МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ПОЛІСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет інформаційних технологій,
обліку та фінансів
Кафедра комп'ютерних технологій
і моделювання систем

Кваліфікаційна робота
на правах рукопису

Поліщук Максим Віталійович

УДК 004.056:004.4

КВАЛІФІКАЦІЙНА РОБОТА

СИСТЕМА БЕЗПЕКИ ПРОГРАМНОГО ЗАСТОСУНКУ «ZIN: ASTROFORGE»

125 «Кібербезпека»

Подається на здобуття освітнього ступеня магістр

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____
(підпис, ініціали та прізвище здобувача вищої освіти)

Керівник роботи:
Корченко Анна Олександрівна,
доктор технічних наук, професор

Житомир – 2024

Висновок кафедри_____

за результатами попереднього захисту:_____

Протокол засідання кафедри_____

№ ____ від «_____» _____ 20____ р.

Завідувач кафедри_____

_____ _____ _____

(науковій ступінь, вчене звання) (підпис) (прізвище, ім’я, по батькові)

«_____» _____ 20____ р.

Результати захисту кваліфікаційної роботи

Здобувач вищої освіти _____ захистив (ла)

(прізвище ,ім’я, по батькові)

кваліфікаційну роботу з оцінкою:

сума балів за 100-бальною шкалою _____

за шкалою ECT8 _____

за національною шкалою _____

Секретар ЕК

_____ _____ _____

(науковій ступінь, вчене звання) (підпис) (прізвище, ім’я, по батькові)

АНОТАЦІЯ

Поліщук М. В. Система безпеки програмного застосунку «Zin: AstroForge» – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна робота на здобуття освітнього ступеня магістр за спеціальністю 125 – Кібербезпека. – Поліський національний університет, Житомир, 2024.

Кваліфікаційна робота присвячена реалізації системи безпеки для підвищення ефективності захисту даних програмного застосунку «*Zin: AstroForge*», шляхом інтеграції методів кібербезпеки для захисту активів гри, даних користувачів і цілісності ігрового процесу. Для досягнення даної мети у теоретичній частині роботи проведено дослідження особливостей реалізації методів захисту даних в системі безпеки програмних застосунків. Проведено аналіз інформаційних потреб і визначення предметної області дослідження. Проведено порівняльний аналіз аналогів систем безпеки, які використовують в існуючих програмних застосунках конкурентів. У практичній частині здійснено проектування та реалізацію системи безпеки програмного застосунку «*Zin: AstroForge*». Імплементовано такі складові системи безпеки як: архітектура застосунку, система аутентифікації та моніторингу активності користувачів, віддалене сховище даних користувачів з контролем доступу, механізми шифрування локальних даних клієнту, система виявлення несанкціонованої модифікації даних клієнту, механізми обфускації вихідного коду проекту. Проведено експериментальне дослідження ефективності системи безпеки програмного застосунку «*Zin: AstroForge*».

Робота містить 58 сторінок, 48 рисунків, 3 таблиці, 31 літературне джерело.

Ключові слова: захист даних, кібербезпека, Unity Engine, шифрування, підробка пам'яті, захист від читів, аутентифікація.

SUMMARY

Polishchuk M. V. Security system of the software application "Zin: AstroForge" – Qualification work on manuscript rights.

Qualification work for obtaining a master’s degree in specialty 125 – Cybersecurity. – Polissya National University, Zhytomyr, 2024.

The qualification work is dedicated to the implementation of a security system to improve the data protection efficiency of the software application "Zin: AstroForge" by integrating cyber security methods to protect game assets, user data and the integrity of the game process. To achieve this goal, in the theoretical part of the work, a study of the features of the implementation of data protection methods in the security system of software applications was conducted. An analysis of information needs was carried out and the subject area of the study was determined. A comparative analysis of analogues of security systems, which are used in existing software applications of competitors, was carried out. In the practical part, the design and implementation of the security system of the "Zin: AstroForge" software application was carried out. The following security system components have been implemented: application architecture, user authentication and activity monitoring system, remote user data storage with access control, local client data encryption mechanisms, client data modification detection system, project source code obfuscation mechanisms. An experimental study of the effectiveness of the security system of the "Zin: AstroForge" software application was conducted.

The work contains 58 pages, 48 figures, 3 tables, 31 literary sources.

Key words: data protection, cyber security, Unity Engine, encryption, memory tampering, anti-cheat, authentication.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

ОС – операційна система

ПЗ – програмне забезпечення

ПК – персональний комп’ютер

СБ – система безпеки

AOT – ahead-of-time

API – application programming interface

JIT – just-in-time

IL2CPP – intermediate language to C++

ВСТУП

Розвиток ігрової індустрії за останні роки призвів до значного зростання кількості користувачів, що у свою чергу, призвело до ускладнення процесу розробки ігрових застосунків, оскільки вони дуже часто являють собою складні системи, що оперують з великими обсягами даних як в середині своєї інфраструктури, так і з залученням зовнішніх сервісів. Ігрові застосунки, що варіюються від однокористувацьких ігор до багатокористувацьких онлайн-платформ, є привабливими цілями для кібератак через значні доходи, які вони генерують завдяки внутрішньоігровим покупкам і мікротранзакціям, а також конфіденційний характер даних, які вони обробляють, наприклад, особиста інформація користувачів, баланс внутрішньої валюти та історії транзакцій. Тому потреба в надійних заходах кібербезпеки при розробці ігор стала ще більш пріоритетною, ніж раніше.

Отже, актуальність теми кваліфікаційної роботи пов’язана із необхідністю імплементації методів захисту даних в ігрових застосунках для підвищення рівня безпеки даних, що в них обробляються.

Об’єкт дослідження – процес формування механізмів захисту даних користувача, системи аутентифікації, методів забезпечення цілісності даних у програмному застосунку «*Zin: AstroForge*».

Предмет дослідження – методи, моделі та системи захисту даних користувача, аутентифікації, механізмів забезпечення цілісності даних у програмному застосунку «*Zin: AstroForge*».

Метою кваліфікаційної роботи є розробка системи безпеки програмного застосунку «*Zin: AstroForge*», шляхом інтеграції методів безпеки для захисту активів гри, даних користувачів і цілісності ігрового процесу. Для досягнення зазначеної мети необхідно виконати такі завдання:

- провести аналіз систем безпеки існуючих програмних застосунків;
- спроектувати систему безпеки програмного застосунку «Zin: AstroForge»;

- реалізувати систему безпеки програмного застосунку «Zin: AstroForge»;

- провести експериментальне дослідження запропонованої системи безпеки застосунку «*Zin: AstroForge*».

Наукова новизна. Вперше розроблено систему безпеки застосунку «*Zin: AstroForge*», за рахунок імплементації надійної архітектури, системи аутентифікації та моніторингу активності користувачів, віддаленого сховища даних користувачів з контролем доступу, механізмів шифрування локальних даних клієнту, системи виявлення несанкціонованої модифікації даних та механізмів обфускації вихідного коду проекту, що дало можливість запобігти загрозам, пов’язаним з реверсивною інженерією, підробкою даних і неавторизованим доступом, забезпечуючи надійний захист від поширених загроз безпеки в програмних застосунках.

Практична цінність розробленого рішення полягає у розробці алгоритмів для: шифрування локальних даних застосунку, виявлення модифікації критичних змінних програми, аутентифікації користувачів, доступу до віддаленого сховища даних користувачів, виявлення підробки застосунку, обфускації вихідного коду проекту. На базі даних алгоритмів реалізовано програмне рішення у вигляді системи безпеки застосунку «*Zin: AstroForge*».

За темою кваліфікаційної роботи опубліковано наукові публікації, а саме:

- Поліщук М. В. Використання методів захисту даних в системах безпеки Unity застосунків. Моделювання, керування та інформаційні технології (*MCIT*–2024) : збірник праць учасників Міжнародної науково-практичної конференції, 2024. 367 с.

- Поліщук М. В. Застосування шаблонів *MV*x для побудови безпечної архітектури програмного забезпечення. *Litteris et Artibus*: Нові горизонти : збірник матеріалів Всеукраїнської науково-практичної конференції. Випуск

ІХ / за заг. ред. О. В. Тригуби. Кременець : ВЦ КОГПА ім. Тараса Шевченка, 2024. 358 с.

- Поліщук М. В. Використання алгоритмів обфускації коду у програмних застосунках. Безпека, технології, інновації: нові горизонти : збірник праць учасників міжфакультетської науково-практичної інтернет-конференції здобувачів вищої освіти і молодих вчених, 12 листопада 2024 р. Житомир : Поліський національний університет, 2024. 102 с.

Розділ 1. ТЕОРЕТИЧНИЙ АНАЛІЗ ОСОБЛИВОСТЕЙ РЕАЛІЗАЦІЇ СИСТЕМИ БЕЗПЕКИ ПРОГРАМНОГО ЗАСТОСУНКУ «ZIN: ASTROFORGE»

1.1 Особливості реалізації методів захисту даних в системі безпеки програмного застосунку «Zin: AstroForge»

Розробка ігрових застосунків, або *GameDev*, охоплює весь процес створення ПЗ, від концепту до проектування, розробки та випуску програмого продукту. На сьогодні ігрові застосунки стали всесвітньою креативною індустрією, але через багатопрофільну діяльність їх розробка є дуже складним завданням. Мультидисциплінарний характер процесів, які поєднують аудіо, графічний дизайн, системи керування, штучний інтелект і людський фактор, також робить практику розробки програмного забезпечення ігор дещо відмінною від традиційної розробки програмного забезпечення [1].

Оскільки, як і інше програмне забезпечення, ігрові застосунки стають більш комплексними, потреба в надійних заходах кібербезпеки стає все більш пріоритетною. У даному розділі досліджуються особливості застосування та роль кібербезпеки в сфері розробки ігрових застосунків, висвітлюються основні загрози в ігрових середовищах та методи, що використовуються для їх захисту від кібератак.

Кібербезпека в сфері розробки ігрових застосунків служить багатьом цілям, зокрема захисту конфіденційних даних користувачів, запобіганню шахрайським діям, забезпеченню безпеки внутрішніх транзакцій і підтримці цілісності ігрового процесу та економічного балансу шляхом боротьби з застосуванням читів, експлойтів та інших інструментів, що можуть бути використані зловмисниками для отримання несанкціонованого доступу до даних застосунку. Ці дії можуть призвести до значних фінансових втрат компанії та репутаційної шкоди, тим самим підкреслюючи необхідність для розробників віддавати пріоритет надійним практикам розробки ПЗ з ранніх етапів створення програмного продукту.

Система безпеки програмного забезпечення зазвичай визначається як набір заходів, методів та інструментів, призначених для захисту програмного забезпечення та пов’язаних із ним даних від несанкціонованого доступу, модифікації чи пошкодження. Система безпеки спрямована на забезпечення трьох основних принципів безпеки: конфіденційності, цілісності та доступності даних, гарантуючи, що конфіденційна інформація доступна лише авторизованим особам, залишається незмінною та доступною, коли це необхідно [2, 3]. Архітектурні патерни, що застосовуються при побудові системи безпеки програмного забезпечення, описані в табл. 1.1 [4].

Таблиця 1.1 – Архітектурні патерни та їх призначення

Патерн	Призначення
Єдина точка доступу (*Single Access Point*)	Надання модулю безпеки та можливості входу в систему
Точка перевірки (*Check Point*)	Організація перевірок безпеки та їх наслідків
Ролі (*Roles*)	Організація користувачів зі схожими правами безпеки.
Сесія (*Session*)	Локалізація глобальної інформації в багатокористувацькому середовищі
Повний перегляд із помилками (*Full View With Errors*)	Надання користувачам повного перегляду з відображенням помилок, коли це необхідно.
Обмежений перегляд (*Limited View*)	Дозволяє користувачам бачити лише те, до чого вони мають доступ
Захищений рівень доступу (Secure Access Layer)	Інтеграція захисту додатків із низькорівневим захистом

Необхідність реалізації системи безпеки ігрового середовища існує через наявність таких потенційних загроз, що можуть вплинути як на процес користування застосунком, так і на безпеку даних та асетів компанії-розробника:

1) Шахрайство: інструменти для читингу та аналізатори пам’яті, такі як *Cheat Engine* і *GameGuardian*, можуть бути використані для пошуку та зміни даних застосунку [5]. Аналіз пам’яті – це пошук певного значення або шаблону

в пам’яті, наданій відкритому (цільовому) процесу. Кінцевим результатом сканування є список адрес, який може бути уточнений (скорочений) додатковими скануваннями [6]. Використання даних інструментів дозволить зловмисникам отримати певні переваги в межах ігрового середовища, наприклад, економічні, шляхом втручання у внутрішню економічну екосистему. Це може призвести до внутрішньої інфляції, економічних та репутаційних втрат, а також відштовхнути аудиторію від подальшого використання програмного продукту [5].

2) Загрози безпеці облікових записів: оскільки є необхідність у системі аутентифікації користувачів, існує ризик несанкціонованого доступу до конфіденційних даних, якщо не вжити належних заходів безпеки при її імплементації.

3) Атаки на стороні клієнта: неналежне зберігання чутливих даних на стороні клієнту створює можливість доступу до них, їх витоку, або втручання в ігровий процес. Зазвичай до цієї загрози призводять відсутність таких заходів безпеки:

- шифрування конфіденційних даних (прогрес гравця, конфіг-файли, баланс на рахунку). Таким чином, навіть якщо зловмисник отримає доступ до локальних даних, їх буде неможливо прочитати;

- система виявлення несанкціонованої модифікації даних. Оскільки шифрування лише ховає дані від читерського ПЗ, є потреба у повноцінному захисті їх від модифікації, наприклад, впровадивши функції виявлення підозрілої активності або аномалії даних. Це може бути реалізовано шляхом моніторингу зміни даних під час виконання програми, а також з використанням *honey pot* пасток. *Honey pot* в даному контексті являє собою створення фальшивих версій критичних змінних, при спробі модифікації яких викликається тригер, що дозволяє системі реагувати на це контрзаходами, такими як обмеження акаунту зловмисника;

- зберігання критичних даних на стороні серверу замість локального сховища, що дозволяє зменшити вірогідність доступу зловмисників до

захищених даних;

- безпечна аутентифікація та авторизація користувачів.

4) Незахищеність коду та асетів проекту: існує необхідність у захисті проекту від реверсивної інженерії. Реверсивна інженерія програмного забезпечення – це підгалузь розробки програмного забезпечення, що пов’язана з аналізом існуючої системи програмного забезпечення з метою синтезу інформації про її цільові аспекти. Поняття реверсивної інженерії можна краще зрозуміти, якщо поглянути на його контекст і роль у розробці та еволюції програмного забезпечення. У той час як класичний процес розробки програмного забезпечення направлений на вдосконалення системи, проходячи через етапи визначення вимог, проектування та реалізації, реверсивна інженерія починається з аспектів реалізації та переходить до абстракцій вищого рівня, такі як ті, що зустрічаються на етапах проектування та визначення вимог (рис. 1.1) [7].

Рисунок 1.1 – Порівняння класичного процесу розробки ПЗ та реверсивної інженерії

Реверсивна інженерія дозволяє отримати доступ до вихідного коду гри, пропрієтарних технологій та алгоритмів компанії. Це може призвести до крадіжки інтелектуальної власності, копіювання або викрадення коду реалізацій ігрових механізмів або цілих проектів. Крім того, якщо код застосунку легко піддається реверсивній інженерії, зловмисники можуть знаходити вразливі місця та здійснювати маніпуляцій з кодом, шляхом перевизначення існуючих методів у коді, за допомогою написання власних функцій. Даний вид втручання у код застосунку називається «*script hooking*». Одним із способів мінімізації даної загрози є використання алгоритмів

обфускації коду проекту [5]. Обфускатор – це компілятор, який перетворює будь-яку програму в обфусковану програму, яка має ті самі функції вводу-виводу, що й оригінальна програма, але є «нерозбірливою» для особи, що намагається прочитати її код. [8].

1.2 Особливості функціонування Unity Engine як інструмента для розробки програмних застосунків

Unity Engine – це багатоплатформовий інструмент для розробки відеоігор, який використовується для створення та управління інтерактивним *2D* та *3D*-контентом у режимі реального часу. В якості основної мови програмування в *Unity Engine* використовується мова *C#* [9].

Важливим аспектом створення застосунків у *Unity* є скриптинговий бекенд. Існує два його види: *Mono* та *IL2CPP* (*Intermediate Language To C++*), кожен з яких використовує різну техніку компіляції:

- *Mono* використовує компіляцію *JIT* і компілює код за вимогою під час виконання програми.

- *IL2CPP* використовує компіляцію *AOT* і компілює всю програму перед її запуском [10].

Різниця між *Mono* та *IL2CPP* з точки зору безпеки в першу чергу пов’язана з тим, як вони здійснюють виконання та компіляцію коду, що впливає на вразливість застосунку до реверсивної інженерії, захист застосунку від зовнішніх втручань під час його виконання, а також частково впливає на обфускацію коду. *IL2CPP* є більш безпечним, ніж *Mono*, завдяки компіляції *AOT* в нативний машинний код, що ускладнює спроби реверсивної інженерії та втручання під час виконання. На практиці, *Mono* використовується в період розробки та прототипізації проекту, оскільки тестові збірки проекту з його використанням компілюються швидше, тоді як *IL2CPP* використовується для фінальних робочих збірок, де безпека та дотримання вимог платформ цифрової дистрибуції є пріоритетом [5].

При збірці проекту за допомогою *IL2CPP*, *Unity* автоматично виконує

наступні кроки:

1) Здійснюється компіляція *C#* коду програми в бібліотеки *DLL .NET* (керовані збірки).

2) *Unity* здійснює видалення невикористаного байт-коду (code stripping).

3) Бекенд *IL2CPP* перетворює всі керовані збірки на стандартний код C++.

4) Компілятор *C++* компілює згенерований *C++* код і *runtime* частину IL2CPP з використанням компілятора нативної платформи.

5) *Unity* створює або виконуваний файл, або *DLL*, залежно від платформи під яку здійснюється збірка проекту [11].

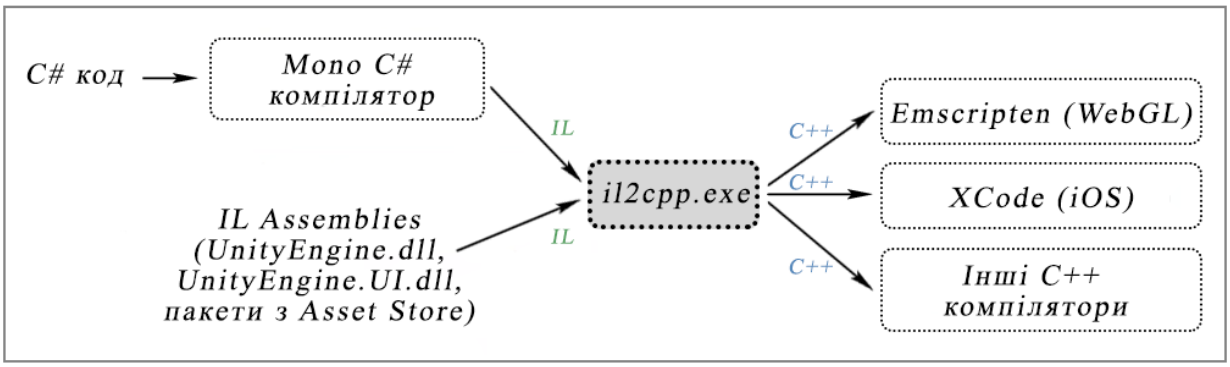

Рисунок 1.2 – Схема роботи *IL2CPP*

1.3 Аналіз програмних застосунків та їх систем безпеки

З метою отримання інформації про сильні та слабкі сторони існуючих рішень на сьогоднішній день, їх методів захисту, та виокремити як найкращі практики, так і прогалини в поточних підходах побудови систем безпеки, було проведено конкурентний аналіз існуючих систем безпеки в ігрових середовищах. Серед існуючих конкурентних рішень можна виділити наступні проекти: *Subway Surfers, Temple Run 2, Among Us, COD: Mobile, Unturned* та *Hay Day*.

Шляхом використання відкритих джерел, а також інструментів сканування, аналізу, декомпіляції коду та інших методів реверсивної інженерії, було здійснено збір інформації про такі характеристики

впроваджених систем безпеки в конкурентних рішеннях:

1) Скриптинговий бекенд (*Mono, IL2CPP*).

2) Захист на стороні клієнту:

- шифрування даних;
- захист коду від аналізаторів пам'яті;
- система виявлення модифікації даних;
- обфускація коду.

3) Захист на стороні серверу:

- система аутентифікації;
- система контролю доступу до БД;
- зберігання даних користувача на сервері;

4) Підтримка ОС (*Android, IOS, MacOS, Windows*).

За результатами проведеного аналізу можна виділити основні переваги та недоліки систем безпеки програмних застосунків.

«Subway Surfers» – мобільний застосунок, створений із використанням скриптингового бекенду *IL2CPP* [12]. Присутнє шифрування даних клієнту, проте воно є лише частковим – шифруються *JSON* файли, що генеруються системою зберігання прогресу користувача, що захищає їх від зовнішнього втручання, але не шифруються змінні всередині коду проекту, що уможливлює їх читання та модифікацію сторонніми програмами [13, 14]. Присутня система аутентифікації, що використовує сервіси *Google Play*, *iCloud* та *Facebook*. Вхід в обліковий запис не є обов'язковим. Дані користувача зберігаються на сервері лише якщо була здійснена реєстрація у застосунку [26]. Це означає, що дані незареєстрованих користувачів, зберігаються локально, що дозволяє здійснювати маніпуляції з ними.

«Temple Run 2» – мобільний застосунок, створений із використанням скриптингового бекенду *IL2CPP* [12]. Шифрування даних відсутнє, архітектура застосунку побудована таким чином, що всі чутливі дані, в тому числі дані прогресу користувача, локально зберігаються в єдиному файлі у відкритому вигляді, що несе високі ризики втручання у роботу застосунку [15].

Присутня система аутентифікації, що використовує сервіси *Google Play* та *iCloud*. Вхід в обліковий запис не є обов'язковим. Дані користувача зберігаються на сервері лише якщо була здійснена реєстрація у застосунку [27]. Враховуючи підхід до зберігання критичних даних користувача, це становить ще вищу загрозу модифікації даних застосунку.

«Among Us» – застосунок, доступний як на мобільних платформах, так і на ПК. Створений із використанням скриптингового бекенду *IL2CPP* [12, 16]. До переходу на *IL2CPP* у версії *«Among Us»* для ОС *Windows* використовувався *Mono*, що дозволило зловмисникам отримати доступ до внутрішньої організації файлів проекту [17]. *Mono* збірки легше піддаються зламу та модифікації, оскільки вони зберігають у собі оригінальний *.NET* код, що дозволяє хакерам легко декомпілювати та змінити внутрішню логіку програми. Це призвело до значних проблем з шахрайством на початку життєвого циклу проекту. Присутня система аутентифікації, що викристовує сервіси *Steam, Epic Games, Google Play, Apple ma Xbox Live*. Авторизація у застосунку є обов'язковою [28]. Також присутні механізми шифрування критичних змінних [18] та обфускація назв структур і функцій у коді [19, 20].

«COD: Mobile» – мобільний застосунок, створений із використанням скриптингового бекенду *IL2CPP* [12]. Має інтегровану власну античитингову систему, що використовує активний моніторинг дій користувача у грі Ця система відстежує аномальну поведінку користувачів і позначає підозрілі дії, які можуть вказувати на шахрайство. Також використовуються методи обфускації та шифрування коду, щоб завадити шахраям легко проаналізувати або змінити його [21, 22]. Недоліком даної системи є великий відсоток помилкових спрацювань, через що навіть легітимні користувачі отримували обмеження та блокування облікових записів [23].

«Unturned» – застосунок, доступний на ПК, створений із використанням скриптингового бекенду *Mono*, що робить його більш вразливим до реверсивної інженерії [12]. Використовує античит систему *Easy Anti-Cheat*, що є широко використовуваним стороннім сервісом, призначеним для

запобігання шахрайству в онлайн-іграх. *Easy Anti-Cheat* зосереджується на виявленні несанкціонованих змін файлів застосунку, підозрілого програмного забезпечення або процесів, запущених на комп’ютері гравця, які можуть бути використані для маніпулювання програмою [29]. Хоча *Easy Anti-Cheat* ефективний у багатьох випадках, він не є рішенням всіх проблем, оскільки досвідчені хакери все одно можуть знайти способи обійти його, особливо в системах із вимкненими засобами безпеки. Крім того, *Easy Anti-Cheat* не захищає від експлойтів на стороні сервера або вразливостей у коді клієнту. Присутній механізм обфускації назв структур і функцій у коді [24]. Система аутентифікації присутня і використовує сервіси *Steam*. Авторизація у застосунку є обов’язковою.

«Hay Day» – мобільний застосунок, створений із використанням скриптингового бекенду *IL2CPP* [12]. Шифрування даних відсутнє, проте є захист ключових даних користувача, шляхом їх обробки та зберігання на стороні серверу. Такий підхід гарантує, що найважливіші змінні програми захищені від втручання на стороні клієнту такими інструментами як аналізатори пам'яті, проте це не стосується інших даних застосунку, які все ще зберігаються локально. Даний підхід вимагає постійного Інтернет-з’єднання, що є одним з його недоліків [25]. Система аутентифікації присутня і використовує власний сервіс *Supercell ID* [30]. Вхід в обліковий запис є обов’язковим.

Результати порівняльного аналізу систем безпеки в існуючих ігрових середовищах відображено у табл. 1.2.

Таблиця 1.2 – Порівняння систем безпеки в існуючих ігрових середовищах

Система безпеки	Скрипт. бекенд		Клієнт				Сервер			ОС			
	Mono	IL2CPP	Шифрування даних	Захист коду від аналізаторів пам'яті	Система виявлення модифікації даних	Обфускація коду	Система аутентифікації	Система контролю доступу до БД	Зберігання даних користувача на сервері	Android	IOS	MacOS	Windows
Subway Surfers	-	+	+	-	-	-	+	+	+	+	+	-	-
Temple Run 2	-	+	-	-	-	-	+	+	+	+	+	-	-
Among Us	+	+	+	-	-	+	+	+	+	+	+	+	+
COD: Mobile	-	+	+	-	+	+	+	+	+	+	+	-	-
Unturned	+	-	-	-	+	-	+	+	+	-	-	-	+
Hay Day	-	+	-	+	-	-	+	+	+	+	+	-	-

Висновки до першого розділу

У першому розділі кваліфікаційної роботи проведено аналіз систем безпеки існуючих програмних застосунків, що дало можливість визначити загальні практики, сильні сторони та вразливі місця в поточних рішеннях. Аналіз також виявив прогалини в певних системах, такі як відсутність шифрування локальних даних, залежність від застарілих технологій і відсутність захисту від зворотного проектування.

Розділ 2. РЕАЛІЗАЦІЯ СИСТЕМИ БЕЗПЕКИ ПРОГРАМНОГО ЗАСТОСУНКУ «ZIN: ASTROFORGE»

2.1 Проектування системи безпеки програмного застосунку «Zin: AstroForge»

До компонентів спроектованої системи безпеки програмного застосунку «*Zin: AstroForge*» належать:

- надійна архітектура застосунку для забезпечення масштабованості та зручності обслуговування;
- система аутентифікації та моніторингу активності користувачів;
- віддалене сховище даних користувачів з контролем доступу;
- механізми шифрування локальних даних клієнту;
- система виявлення несанкціонованої модифікації даних клієнту;
- механізми обфускації вихідного коду та захист від підробки проекту.

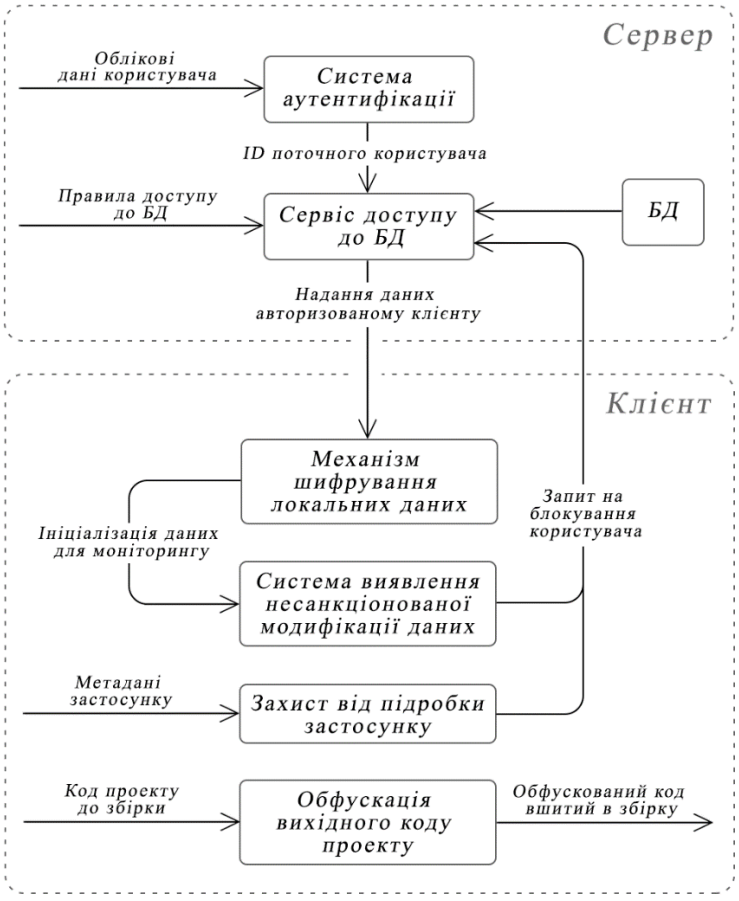

Рисунок 2.1 – Структурна схема системи безпеки

Основою надійної системи безпеки є надійна архітектура ПЗ. Безпечна архітектура застосунку необхідна для забезпечення масштабованості та зручності обслуговування проекту. Модульні компоненти та методи безпечного кодування дозволять легко оновлювати та розширювати проект, зменшуючи ймовірність появи вразливостей при додаванні нових функцій. Основою архітектури даного застосунку є сервісно-орієнтований підхід проектування і організації інформаційної архітектури, який полягає у використанні розподілених, слабко пов'язаних замінних компонентів (сервісів), оснащених стандартизованими інтерфейсами. До використаних принципів при застосуванні даного підходу належать наступні:

1) Слабка зв’язаність. Сервіси слабо пов’язані між собою, що не заважає їх використанню в межах однієї сутності. Основна ідея даного принципу полягає у зменшенні взаємозалежності сервісів до рівня, на якому все ще зберігається їх сумісність.

2) Стандартизовані інтерфейси. Однією з основних вимог даного підходу є необхідність використання стандартизованих інтерфейсів, оскільки інші сервіси чи компоненти системи взаємодіють із потрібним їм сервісом через його інтерфейс. Інтерфейс в даному випадку є контрактом, в якому прописані правила, згідно з якими оперують класи, що його реалізують. Наприклад, інтерфейс може декларувати те, які дані необхідні даному сервісу, а також для чого та за якими правилами він може використовуватися.

3) Повторне використання. Полягає у можливості повторного використання сервісів. З цього слідує те, що вони мають бути реалізовані таким чином, що їх внутрішня логіка не залежала від конкретної технології, що використовується у проекті.

4) Доступ до сервісів. Ще однією вимогою є можливість компонентів системи легко знайти потрібний сервіс, щоб ним користуватися. Усім споживачам сервісів має бути доступним реєстр сервісів. Реєстр сервісів – це сховище сервісів, що доступне у межах системи, завдяки якому компоненти системи можуть отримувати доступ до необхідних їх сервісів.

5) Автономність. Кожен сервіс повинен мати можливість працювати та функціонувати незалежно та бути здатним самостійно керувати виділеними йому ресурсами, логікою та середовищем.

Іншим важливим аспектом даної архітектури є процес керування станами та поведінкою об’єктів в системі, оскільки від цього залежить зручність контролю, підтримки проекту та здатність до його масштабування. Одним із популярних інструментів, який використовується для даної задачі є машина кінцевих станів, або «*Finite State Machine (FSM)*». *FSM* – це математична модель, яка використовується для проектування систем, які можуть перебувати в одному з обмеженої кількості станів у певний момент часу. На рис. 2.2 наведено *UML*-діаграму станів спроектованого застосунку.

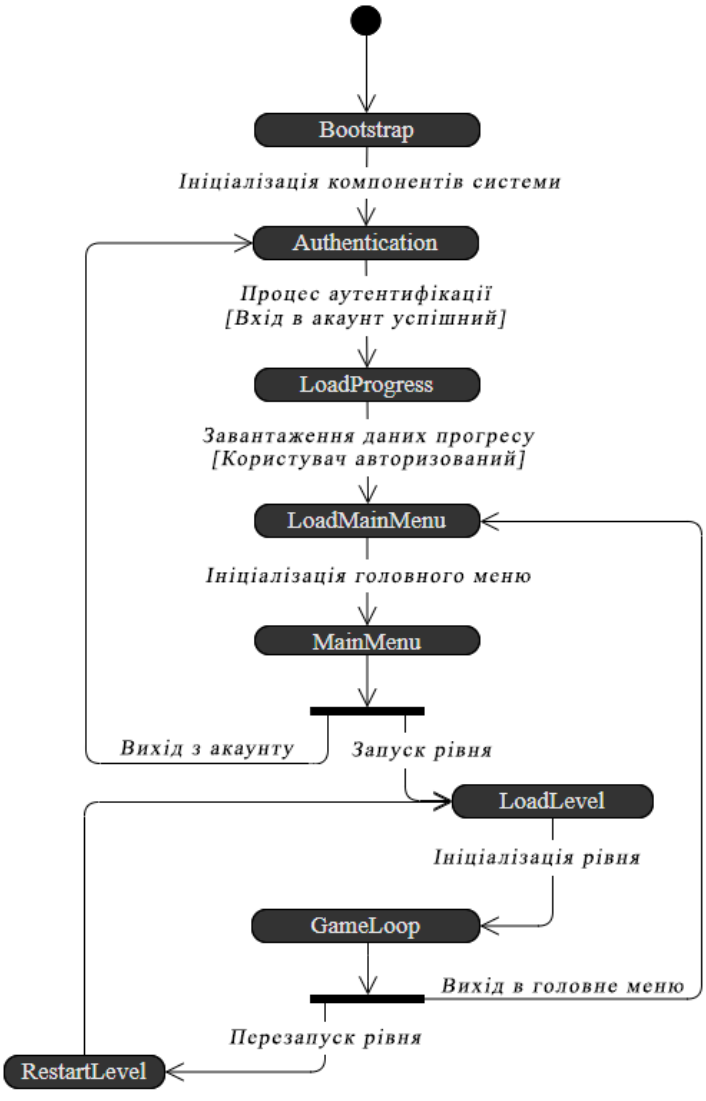

Рисунок 2.2 – Діаграма станів

При запуску застосунку відбувається процес ініціалізації та реєстрації всіх сервісів у системі. Після цього відбувається процес аутентифікації користувача в системі, у якого є можливість увійти в свій обліковий запис з використанням електронної пошти та паролю, або створити новий запис шляхом реєстрації. При реєстрації здійснюється валідація введених даних як на стороні клієнту, так і на сервері, для запобігання обходу вимог до оформлення паролів та інших даних входу. Внаслідок цього, кожному користувачу присвоюється унікальний ідентифікатор, який зберігається на сервері разом з іншими даними для входу, такими як ім’я, пошта та пароль користувача. Після авторизації у застосунку, відбувається синхронізація з сервером, на якому зберігаються дані прогресу користувача. Вони також зберігаються і локально, але у зашифрованому вигляді, що дозволяє користуватися застосунком в офлайн режимі. Після авторизації та завантаження даних прогресу користувача відбувається перехід до головного меню застосунку, в якому можна користуватись усіма функціями, що стосуються самої гри, а саме перегляд і редагування даних профілю, запуск рівнів, здійснення внутрішньоігрових покупок і т. д. Для закінчення взаємодії з застосунком його можна закрити у будь-який момент часу без наслідків для даних прогресу користувача, оскільки їх збереження та вивантаження на сервер відбувається автоматично до закриття програми. Порядок взаємодії між користувачем та об’єктами системи з плином часу можна побачити на *UML*-діаграмі послідовності, що наведена на рис. 2.3.

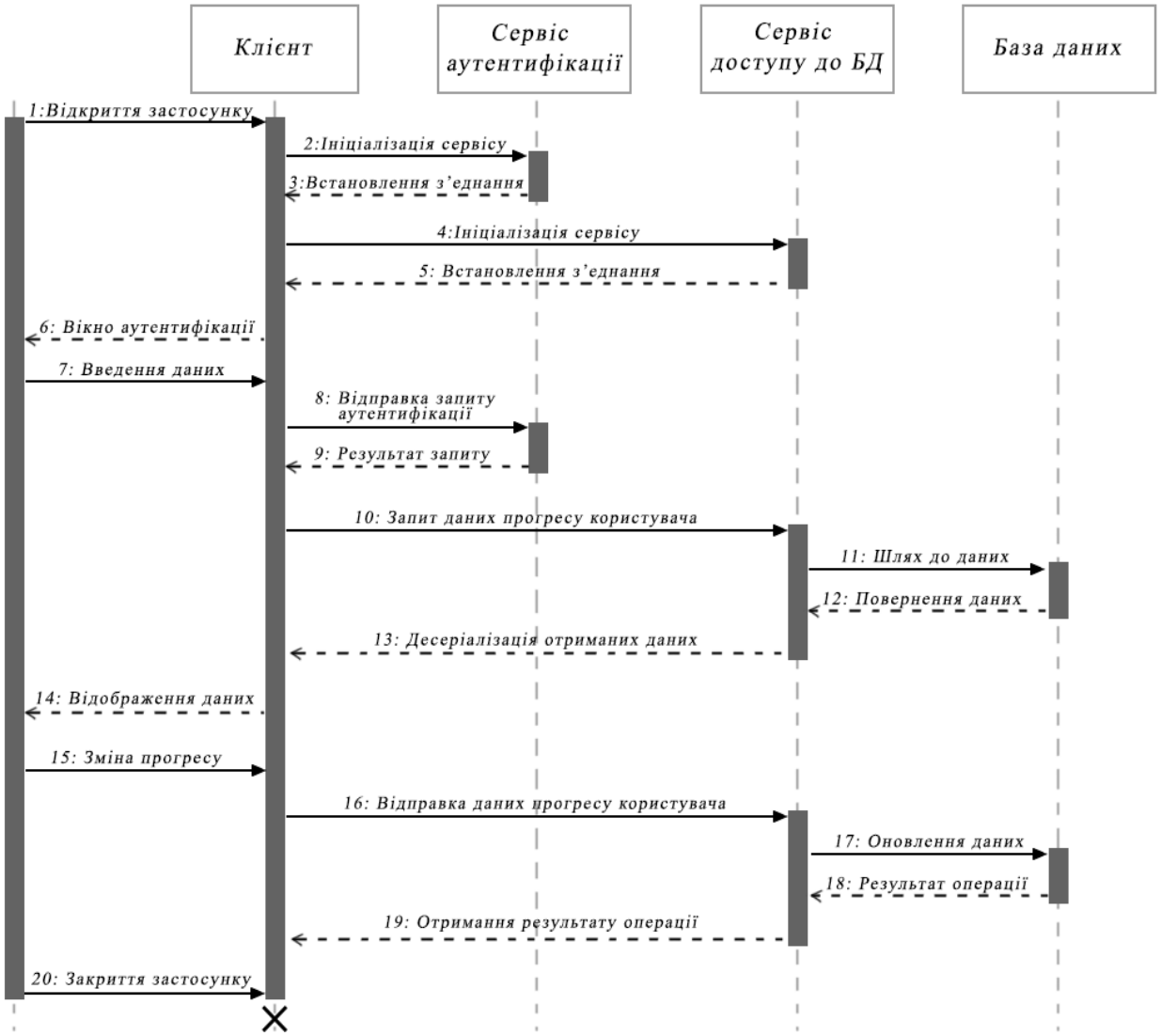

Рисунок 2.3 – Діаграма послідовності

2.2 Реалізація системи безпеки програмного застосунку «Zin: AstroForge»

В основі сервісної архітектури створеного застосунку лежить патерн проектування «*Service Locator*». Було реалізовано машину кінцевих станів. Запуск гри відбувається з окремої сцени «*Initial*» та стану «*BootstrapState*», в якому відбувається реєстрація сервісів, що дозволяє явно визначити порядок їх ініціалізації. Це дозволяє реалізувати єдину точку входу в застосунок, розділити процеси завантаження компонентів програми, а також визначити порядок конструювання об'єктів за допомогою *Dependency Injection*. Код методу «*RegisterServices()*» для реєстрації сервісів з використанням патерну

«*Service Locator*» наведено на рис. 2.4.

```
private void RegisterServices()
{
    RegisterStaticDataService();
    services.RegisterSingle<IGameStateMachine>(stateMachine);
    services.RegisterSingle<IInputService>(InputService());
    services.RegisterSingle<IAssets>(new AssetProvider());
    services.RegisterSingle<IPersistentProgressService>(new PersistentProgressService());
    services.RegisterSingle<IGamePauseService>(new GamePauseSevice());
    services.RegisterSingle<IAuthenticationService>(new AuthenticationService(
        services.Single<IGameStateMachine>()));
    services.RegisterSingle<IRealtimeDatabaseService>(new RealtimeDatabaseService(
        services.Single<IAuthenticationService>(),
        services.Single<IPersistentProgressService>()));
    services.RegisterSingle<IPurchaseService>(new PurchaseService(
        services.Single<IPersistentProgressService>(),
        services.Single<IStaticDataService>()));
    services.RegisterSingle<IScoreService>(new ScoreService(
        services.Single<IPersistentProgressService>()));
    services.RegisterSingle<IUIFactory>(new UIFactory(
        services.Single<IAssets>(),
        services.Single<IStaticDataService>(),
        services.Single<IPersistentProgressService>(),
        services.Single<IGameStateMachine>(),
        services.Single<IGamePauseService>(),
        services.Single<IScoreService>(),
        services.Single<IPurchaseService>()));
    services.RegisterSingle<IWindowService>(new WindowService(
        services.Single<IUIFactory>()));
    services.RegisterSingle<IGameFactory>(new GameFactory(
        services.Single<IAssets>(),
        services.Single<IStaticDataService>(),
        services.Single<IWindowService>(),
        services.Single<IPersistentProgressService>(),
        services.Single<IScoreService>()));
    services.RegisterSingle<ISaveLoadService>(new SaveLoadService(
        services.Single<IPersistentProgressService>(),
        services.Single<IGameFactory>(),
        services.Single<IRealtimeDatabaseService>()));
}
```

Рисунок 2.4 – Реєстрація сервісів в системі

Систему аутентифікації у застосунку було реалізовано з використанням сервісу *Firebase Authentication*. Він надає комплексний набір інструментів для керування процесами входу користувачів в облікові записи і контролю доступу но них. В даному проекті було реалізовано аутентифікацію за допомогою електронної пошти та пароля. Інтерфейс вікон для реєстрації та входу в обліковий запис користувача наведено на рис. 2.5.

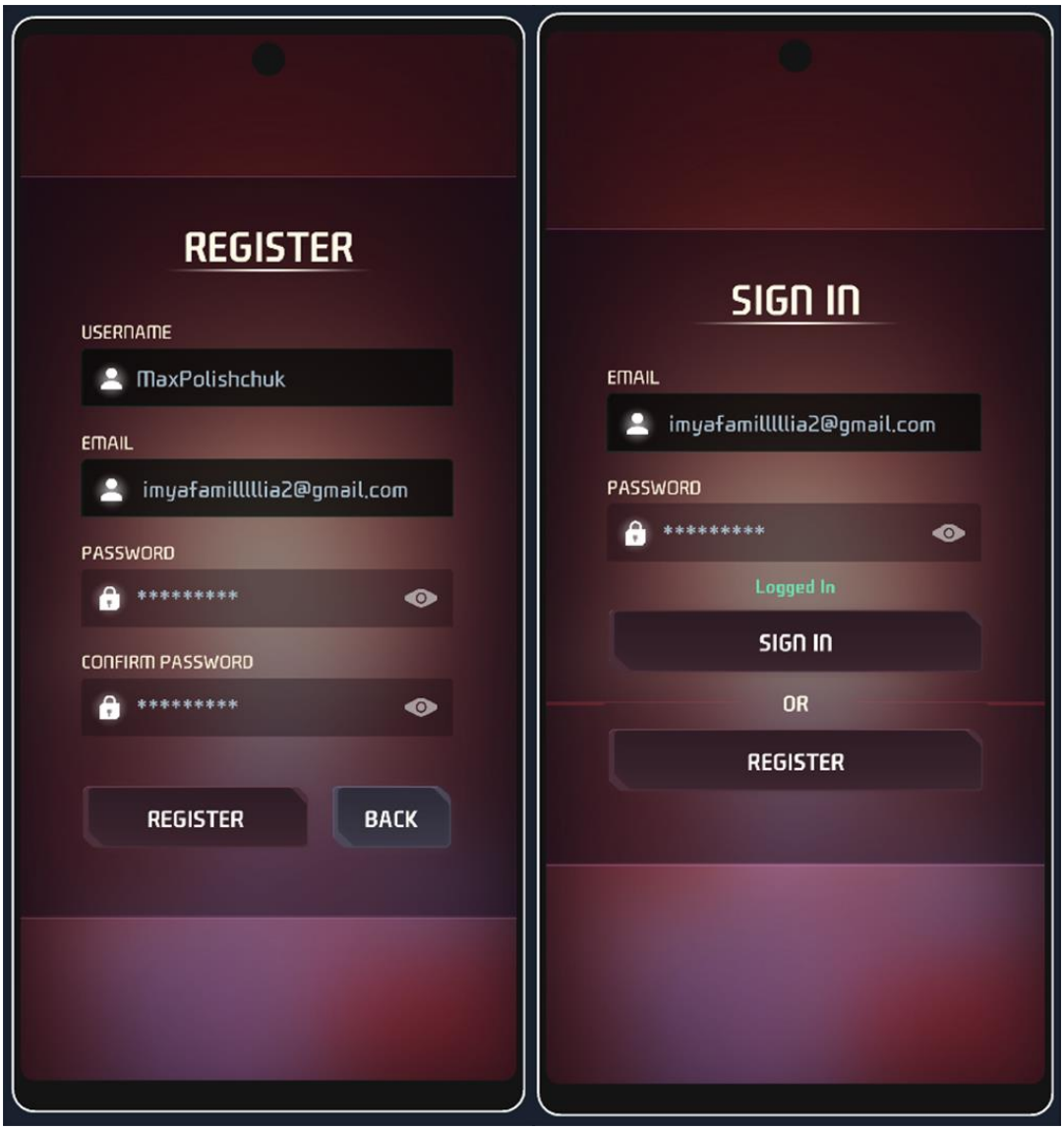

Рисунок 2.5 – Вікна реєстрації та входу в обліковий запис

На даному етапі відбувається первинна валідація полів вводу на стороні клієнту (рис. 2.6), а також повторна перевірка дотримання вимог оформлення паролів на стороні серверу (рис. 2.7). При підтвердженні входу відбувається виклик методу «*Login()*» через сервіс «*AuthenticationService*» (рис. 2.8). У ньому надсилається запит на сервер, на якому відбувається перевірка існування даного користувача в системі і його статус, щоб не допустити авторизацію користувачів з обмеженим акаунтом. В якості додаткового захисту від атак, було встановлено обмеження кількості нових облікових записів, які можна створити за день з однієї *IP*-адреси до 100.

```
if (string.IsNullOrWhiteSpace(username))
{
    DisplayWarning(warningRegisterText, "Missing Username");
}
else if (string.IsNullOrWhiteSpace(email))
{
    DisplayWarning(warningRegisterText, "Missing Email");
}
else if (string.IsNullOrWhiteSpace(password))
{
    DisplayWarning(warningRegisterText, "Missing Password");
}
else if (IsWeakPassword(password))
{
    DisplayWarning(warningRegisterText, "Password must be at least 8 characters");
}
else if (password != confirmPassword)
{
    DisplayWarning(warningRegisterText, "Passwords Do Not Match");
}
```

Рисунок 2.6 – Валідація полів для введення даних

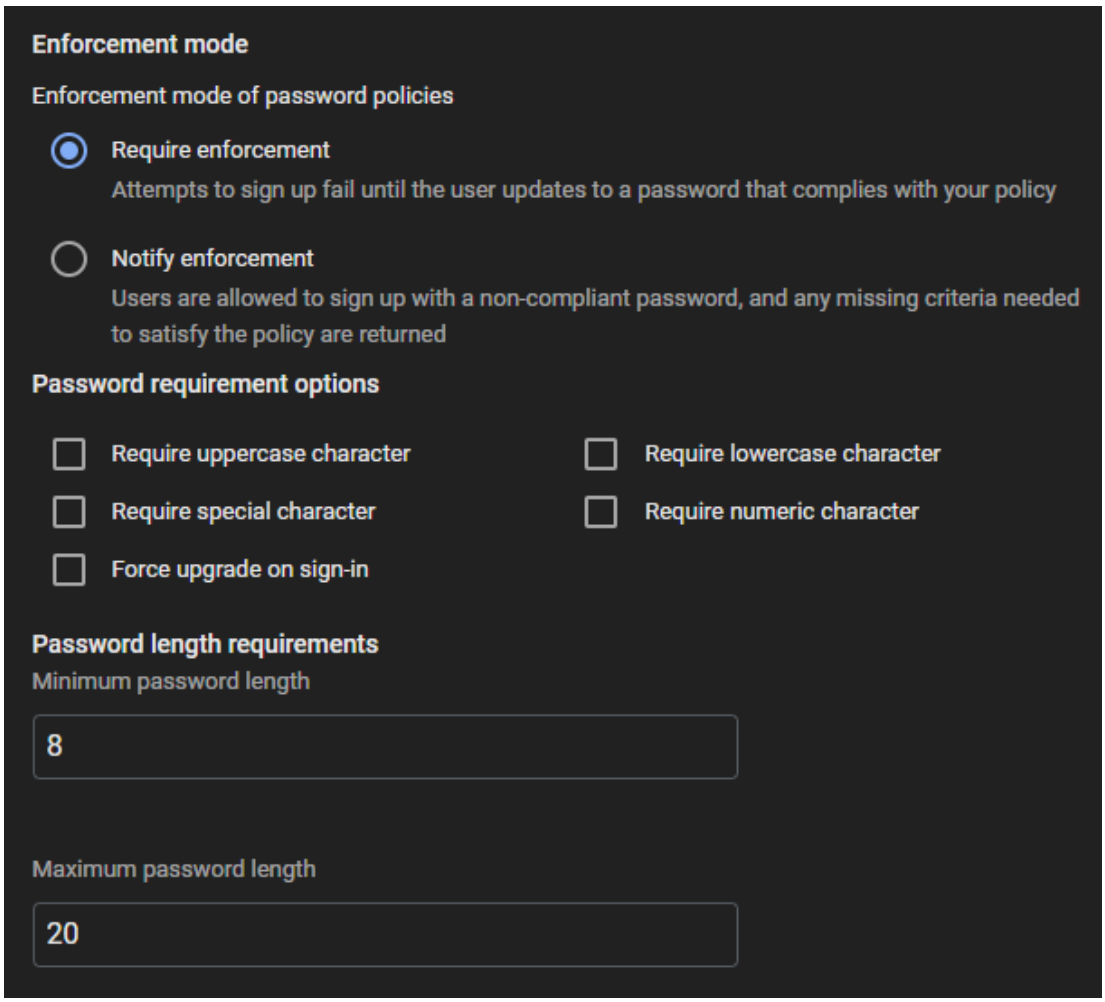

Рисунок 2.7 – Вимоги до паролю користувачів

```
Frequently called  0+1 usages
public IEnumerator Login(string email, string password, TMP_Text warningLoginText, TMP_Text confirmLoginText,
    VerticalLayoutGroup contentLoginLayoutGroup)
{
    if (string.IsNullOrWhiteSpace(email))
    {
        DisplayWarning(warningLoginText, "Missing Email");
    }
    else if (string.IsNullOrWhiteSpace(password))
    {
        DisplayWarning(warningLoginText, "Missing Password");
    }
    else
    {
        Task<AuthResult> LoginTask = auth.SignInWithEmailAndPasswordAsync(email, password);
        yield return new WaitUntil(() => LoginTask.IsCompleted);

        if (LoginTask.Exception != null)
        {
            Debug.LogWarning(message: $"Failed to register task with {LoginTask.Exception}");
            DisplayWarning(warningLoginText, "Wrong Password or Email");
        }
        else
        {
            //User is now logged in. Get the LoginTask result
            user = LoginTask.Result.User;
            Debug.LogFormat("User signed in successfully: {0} ({1})", user.DisplayName, user.Email);
            warningLoginText.text = "";
            confirmLoginText.text = "Logged In";

            stateMachine.Enter<LoadProgressState>();
        }
    }

    LayoutRebuilder.ForceRebuildLayoutImmediate((RectTransform) contentLoginLayoutGroup.transform);
}
```

Рисунок 2.8 – Метод для входу в обліковий запис

Список усіх користувачів, зареєстрованих у системі можна побачити в адмін-панелі сервісу *Firebase Authentication* (рис. 2.9). Крім того, також є можливість моніторингу активності користувачів (рис. 2.10).

Identifier	Providers	Created ↓	Signed In	User UID
testuser@gmail.com	✉	Nov 13, 2024	Nov 13, 2024	ipteTxzGHOWPVYoE49f6SUd...
myusername@gmail.co...	✉	Nov 13, 2024	Nov 13, 2024	0SqhUetObXX7KIKYjAwbLzxq...
cooluser2@gmail.com	✉	Nov 12, 2024	Nov 12, 2024	lM6r4jYeJkWQFPAsTuKMM4Y...
cooluser@gmail.com	✉	Nov 11, 2024	Nov 12, 2024	dbTLzxuvGJWvetFg0VX4Cfld...
imyafamillllia2@gmail....	✉	Nov 10, 2024	Nov 10, 2024	bPVmobLEzwfKyjZ95gm3pqe...

Рисунок 2.9 – Список користувачів в системі

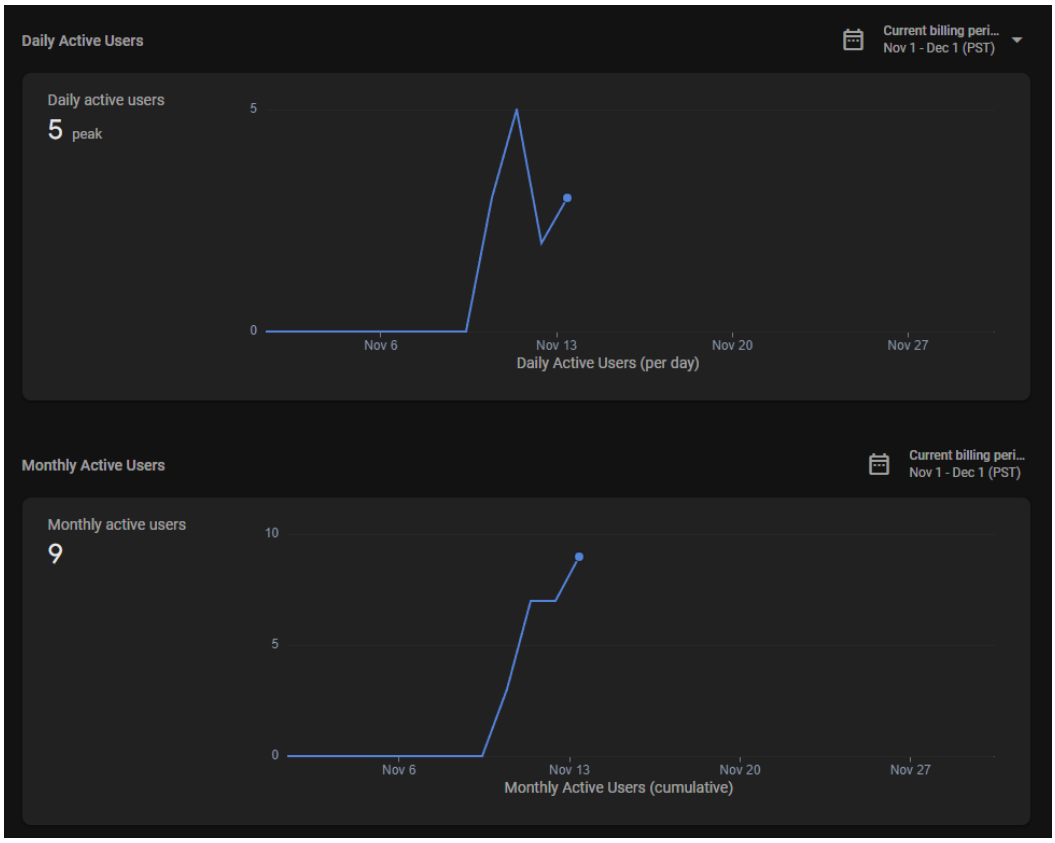

Рисунок 2.10 – Інтерфейс для моніторингу активності користувачів

В якості сервісу доступу до БД виступає *Firebase Realtime Database*. При вході в стан «*LoadProgressState*», через сервіс «*RealtimeDatabaseService*» застосунок здійснює спробу встановлення з'єднання з сервером, на якому зберігаються дані користувача (рис. 2.11).

```
1 usage  Gwyllym *
private async void LoadProgressOrInitNew(int delay)
{
    await Task.Delay(delay);

    if (saveLoadService.LoadLocalProgress() != null)
    {
        progressService.Progress = saveLoadService.LoadLocalProgress();
    }
    else
    {
        bool remoteDataExists = await realtimeDatabaseService.DoesUserDataExistAsync();

        if (remoteDataExists)
            progressService.Progress = await realtimeDatabaseService.LoadProgressFromDatabaseAsync();
        else
            progressService.Progress = NewProgress();
    }

    await realtimeDatabaseService.UpdateDatabaseFromLocalProgressAsync(progressService.Progress.ToJson());
    await saveLoadService.SaveProgress();
    gameStateMachine.Enter<LoadMainMenuState, string>("MainMenu");
}
```

Рисунок 2.11 – Метод синхронізації даних користувача з сервером

Якщо спроба успішна, відбувається синхронізація даних. Якщо спроба невдала, наприклад у випадку відсутності Інтернет з'єднання, відбувається вхід в офлайн режим. В обох випадках дані користувача зберігаються у зашифрованому вигляді у класі «*PlayerProgress*». При завантаженні на сервер весь клас серіалізується і надсилається у форматі *JSON*. Загальну структуру даних прогресу користувача наведено на рис. 2.12

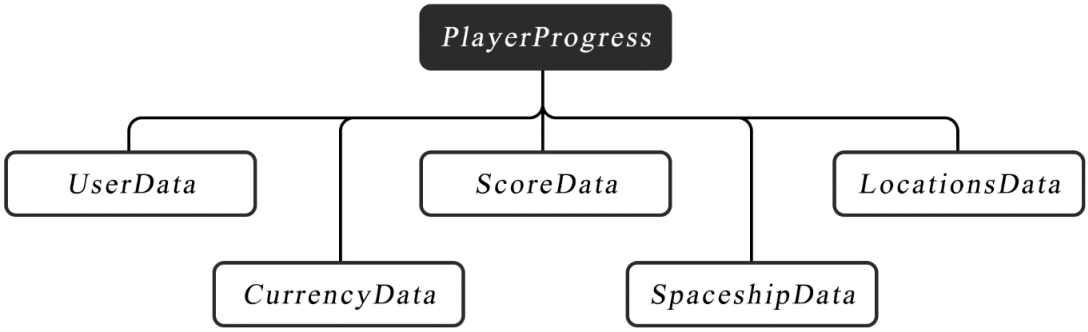

Рисунок 2.12 – Структура даних користувача

Для забезпечення контролю доступу до записів у базі даних було додано правила безпеки:

- перевірка аутентифікації користувача (*auth != null*): декларує, що тільки аутентифіковані користувачі мають доступ до читання та запису в БД.

- ізоляція користувачів (*auth.uid === $uid*): декларує, що користувачі можуть читати та змінювати лише ті дані, що прив'язані до їхнього облікового запису.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".read": "auth != null && auth.uid === $uid",
        ".write": "auth != null && auth.uid === $uid",
      }
    }
  }
}
```

Рисунок 2.13 – Правила для контролю доступу до БД

При реалізації системи шифрування локальних даних користувача було застосовано бібліотеку *CodeStage.AntiCheat.ObscuredTypes* (рис. 2.14).

```
using UnityEngine;
using CodeStage.AntiCheat.ObscuredTypes;
```

Рисунок 2.14 – Підключення бібліотеки *CodeStage.AntiCheat.ObscuredTypes*

Механізм шифрування даних користувача працює наступним чином: конфіденційні дані користувача, такі як баланс внутрішньоігрових валют, дані прогресу та особисті налаштування протягом всього циклу роботи застосунку зберігаються у зашифрованому вигляді за допомогою алгоритму шифрування на основі *XOR*. Шифр *XOR* – це симетричний алгоритм шифрування, що використовує логічну операцію *XOR*, або виняткову диз'юнкцію для шифрування даних. Він не є настільки надійним алгоритмом шифрування, як, наприклад *AES*, але в контексті даного проекту це є найбільш оптимальним рішенням, через його використання в комплексі з іншими заходами безпеки, а також через потенційно велику кількість фрагментів даних прогресу користувача, які необхідно динамічно шифрувати та дешифрувати під час виконання застосунку, що може вплинути на продуктивність його роботи, особливо на мобільних платформах.

Шифрування поширюється на основні типи даних, такі як *int, float, bool* та ін., гарантуючи, що будь-яка спроба змінити їх за допомогою інструментів аналізу пам’яті призведе до неможливості їх прочитати. Це реалізовано шляхом використання прихованих типів даних, або «*ObscuredTypes*» (рис. 2.15).

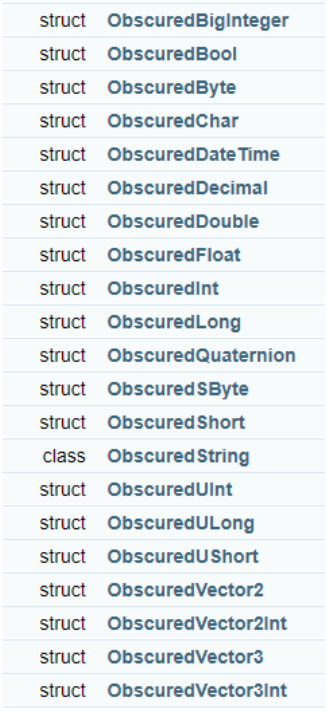

Рисунок 2.15 – *ObscuredTypes*

Такі типи як «*ObscuredInt*», «*ObscuredFloat*» та ін. є структурами, що реалізують інтерфейс *IObscuredType*. Вони функціонують як обгортка над звичайними типами даних, при ініціалізації яких автоматично відбувається шифрування оригінального значення за допомогою щойно згенерованого ключа шифрування. Кожен прихований тип надає публічний *API*, через який інші компоненти системи можуть отримувати доступ на читання та зміну зашифрованих змінних цього типу. Код для надання інструкцій для читання та зміни захищених змінних іншими компонентами системи є тим же, що і у випадку зі звичайними змінними, що не вимагає додаткових модифікацій та налаштувань для їх сумісності з основною логікою застосунку. Головною перевагою прихованих типів, таких як «*ObscuredInt*» над звичайними, такими як «*int*» є те, що зловмисник не зможе легко знайти та змінити потрібне йому значення типу «*ObscuredInt*» в пам'яті, чого не можна сказати про незахищені типи даних.

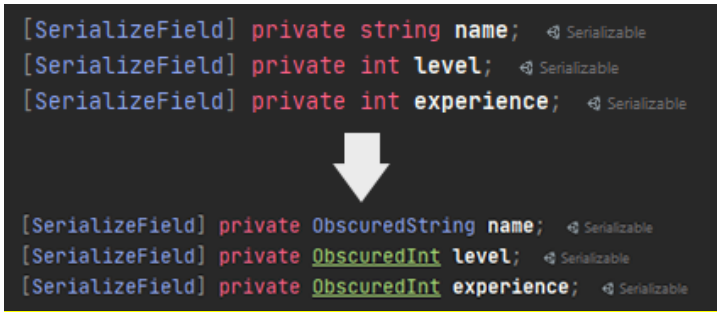

Рисунок 2.16 – Використання *ObscuredTypes* замість звичайних типів даних

Загальний алгоритм шифрування полягає у наступних кроках:

1) Відбувається виклик методу для шифрування значення змінної.

2) Генерується випадковий ключ (рис. 2.17).

Рисунок 2.17 – Методи генерації ключа для *ObscuredString* та *ObscuredInt*

2) За допомогою ключа виконується операція *XOR*, щоб створити зашифровані дані (рис. 2.18). Оператор *XOR* (^) порівнює бінарне представлення значення та ключа. Порівнюється кожен біт:

- якщо біти значення та ключа однакові (0 і 0 або 1 і 1), результат дорівнює 0.

- якщо біти різні (0 і 1 або 1 і 0), результат дорівнює 1.

На виході повертається результат «*value ^ key*», у вигляді зашифрованого цілого числа.

```
internal static char[] InternalEncryptDecrypt(char[] value, char[] key)
{
    if (value == null || value.Length == 0)
        return value;

    if (key.Length == 0)
        return value;

    var keyLength :int  = key.Length;
    var valueLength :int  = value.Length;
    var result = new char[valueLength];

    for (var i = 0; i < valueLength; i++)
        result[i] = (char) (value[i] ^ key[i % keyLength]);

    return result;
}
```

ObscuredString

```
public static int Encrypt(int value, int key)
{
    return value ^ key;
}
```

ObscuredInt

Рисунок 2.18 – Методи шифрування для *ObscuredString* та *ObscuredInt*

4) Для розшифрування даних, знову застосовується той самий ключ і повторно виконується операція *XOR*.

Також для посилення захисту застосовано метод «*RandomizeCryptoKey()*» (рис. 2.19). Він дозволяє змінювати поточний ключ шифрування на нове випадкове значення під час виконання програми та повторно зашифрувати змінну з його використанням, що додатково ускладнює процес отримання доступу до даної змінної, оскільки її зашифроване значення постійно змінюється.

```
public void RandomizeCryptoKey()
{
    hiddenValue = InternalDecrypt();
    currentCryptoKey = GenerateKey();
    hiddenValue = Encrypt(hiddenValue, currentCryptoKey);
}
```

Рисунок 2.19 – Метод *RandomizeCryptoKey()*

Дані користувача до та після шифрування на прикладі секції «*UserData*» наведені на рис. 2.20.

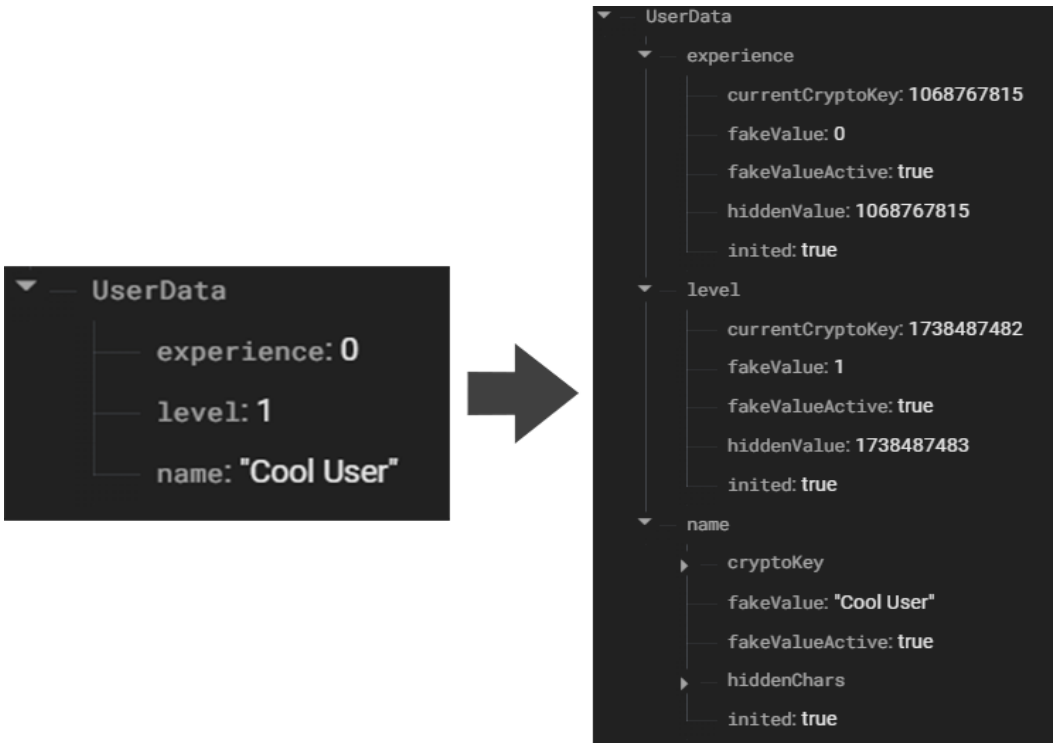

Рисунок 2.20 – Дані «*UserData*» до та після шифрування

Оскільки шифрування лише ховає дані від аналізаторів пам’яті, є потреба у повноцінному захисті від їх модифікації, тому також було реалізовано систему виявлення несанкціонованої зміни даних застосунку, підозрілої активності та аномалії даних. Для цього було додано можливість створення *honey pot* пасток у коді за допомогою класу «*ObscuredCheatingDetector*» – у всіх захищених змінних створюються фальшиві копії, що не шифруються і доступні для модифікації сторонніми інструментами. Це дозволяє шахраям знаходити бажані (фальшиві) значення в пам’яті та змінювати їх, зберігаючи оригінальні значення змінних в безпеці. При спробі модифікації фальшивих змінних викликається метод «*OnCheaterDetected()*», що дозволяє карати зловмисників (рис. 2.21). В даному випадку, через сервіс «*RealtimeDatabaseService*» відбувається виклик методу «*CurrentCurrnetUserAsync*». В ньому надсилається запит до серверу для зміни властивості користувача «*IsSuspended*», а також відбувається інформування користувача про обмеження його облікового запису (рис. 2.22).

```
public void OnCheaterDetected()
{
    Debug.Log("!!!Cheater Detected!!!");
    cheaterDetected = true;
    realtimeDatabaseService.SuspendCurrentUserAsync();
    windowService.Open(WindowId.AccountSuspended);
}
```

Рисунок 2.21 – Метод «*OnCheaterDetected()*»

```
public async Task SuspendCurrentUserAsync()
{
    try
    {
        DatabaseReference userRef = databaseReference
            .Child("users")
            .Child(user.UserId);

        await userRef.Child("isSuspended").SetValueAsync(true);
        Debug.Log("User account has been suspended.");
    }
    catch (System.Exception ex)
    {
        Debug.LogError($"Failed to suspend user: {ex.Message}");
    }
}
```

Рисунок 2.22 – Метод для обмеження облікового запису користувача

Також є можливість калібрування параметрів виявлення змін у пам’яті для зменшення ймовірності помилкових спрацювань (рис 2.23):

- *Double Epsilon*, *Float Epsilon*: максимальна допустима різниця між зашифрованими та підробленими значеннями типу «*ObscuredDouble*» та «*ObscuredFloat*». Поширюється на змінні з плаваючою крапкою;

- *Vector2 Epsilon*, *Vecor3 Epsilon*: максимальна допустима різниця між зашифрованими та підробленими значеннями типу «*ObscuredVector2*» та «*ObscuredVector3*». Поширюється на векторні змінні;

- *Quaternion Epsilon*: максимальна допустима різниця між зашифрованими та підробленими значеннями типу «*ObscuredQuaternion*». Поширюється на змінні для розрахунку кутів у просторі.

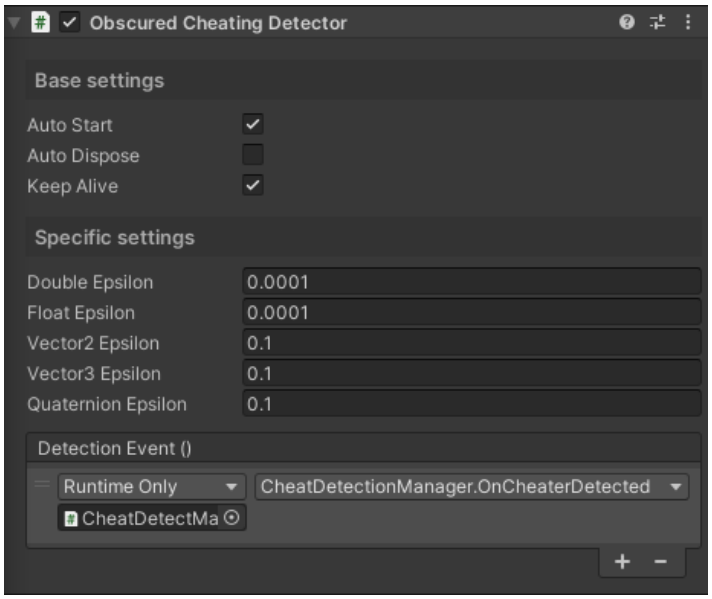

Рисунок 2.23 – Параметри виявлення змін у пам’яті

Незважаючи на імплементований захист від аналізаторів пам’яті та систему виявлення модифікації захищених змінних, все ще існує загроза підробки застосунку. Для захисту від даної загрози було реалізовано додаткові заходи для ускладнення реверсивної інженерії та підробки застосунку. Базовим заходом безпеки для захисту від реверсивної інженерії в сучасних *Unity* застосунках є використання скриптингового бекенду *IL2CPP*. Для цього достатньо обрати його у вікні налаштувань проекту (рис. 2.24).

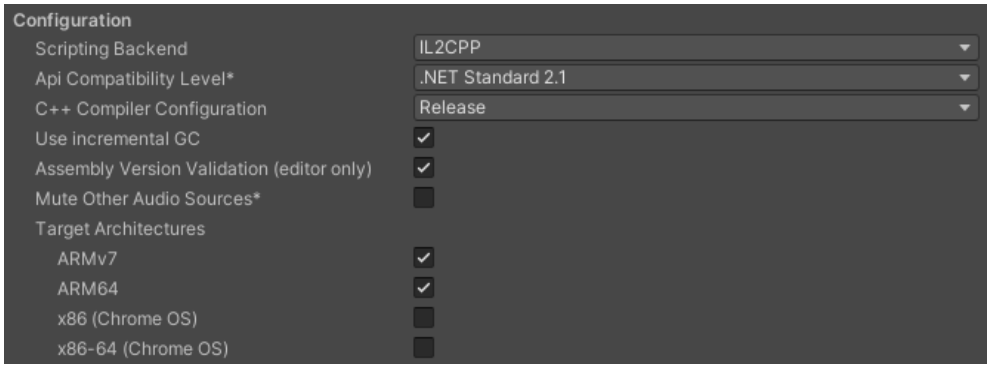

Рисунок 2.24 – Вікно налаштувань компіляції проекту

Оскільки використання *IL2CPP* як бекенду для скриптингу замість *Mono* лише частково обфускує код, також було реалізовано механізм обфускації коду. За основу було взято пакет «*BeeByte*», що надає базові можливості обфускації коду з мінімальним впливом на продуктивність застосунку. Після його додаткової модифікації, обфускатор реалізує такі методи обфускації

коду:

1. Лексичні трансформації. Полягає у зміні назв змінних, функцій, класів та інших ідентифікаторів у коді, щоб приховати їхнє призначення та взаємозв'язок у програмі. Наприклад, назви змінних, такі як «*customerID*» або «*calculateTotal*», можуть бути замінені на випадкові назви «*AEGGFFNPEDG*» або «*AKMHIJNNOAK*», які ускладнюють визначення їхньої ролі у роботі програми.

2. Техніки зміни потоку керування (*control flow*). Це метод, який перешкоджає ефективному аналізу програми, шляхом вставки фіктивного коду. Це дозволяє змінити структуру коду, щоб його було важче відстежити [31].

В якості додаткового захисту було додано перевірку джерела походження застосунку: перед переходом в стан аутентифікації викликається метод «*GetAppInstallationSource()*», який перевіряє з якого джерела було встановлено застосунок (*Gogle Play Store*, *Package Installer* та ін.) (рис. 2.25). Дозвіл на продовження користування застосунком надається тільки якщо його було встановлено з легітимних джерел, в даному випадку – *Gogle Play Store.*

```
public string GetAppInstallationSource()
{
    string installationSource = "Unknown";

    try
    {
        using AndroidJavaClass unityPlayer = new AndroidJavaClass("com.unity3d.player.UnityPlayer");
        using AndroidJavaObject context = unityPlayer.GetStatic<AndroidJavaObject>("currentActivity");
        using AndroidJavaObject packageManager = context.Call<AndroidJavaObject>("getPackageManager");

        string packageName = context.Call<string>("getPackageName");
        string installerPackageName = packageManager.Call<string>("getInstallerPackageName", packageName);

        if (string.IsNullOrEmpty(installerPackageName))
            installationSource = "Other";
        else if (installerPackageName == "com.android.vending")
            installationSource = "Google Play Store";
        else
            installationSource = "Package Installer";
    }
    catch (System.Exception ex)
    {
        Debug.LogError($"Error while fetching installation source: {ex.Message}");
    }

    return installationSource;
}
```

Рисунок 2.25 – Метод для перевірки джерела походження застосунку

Загальний алгоритм функціонування створеної системи безпеки застосунку «*Zin: AstroForge*» наведено на рис. 2.26.

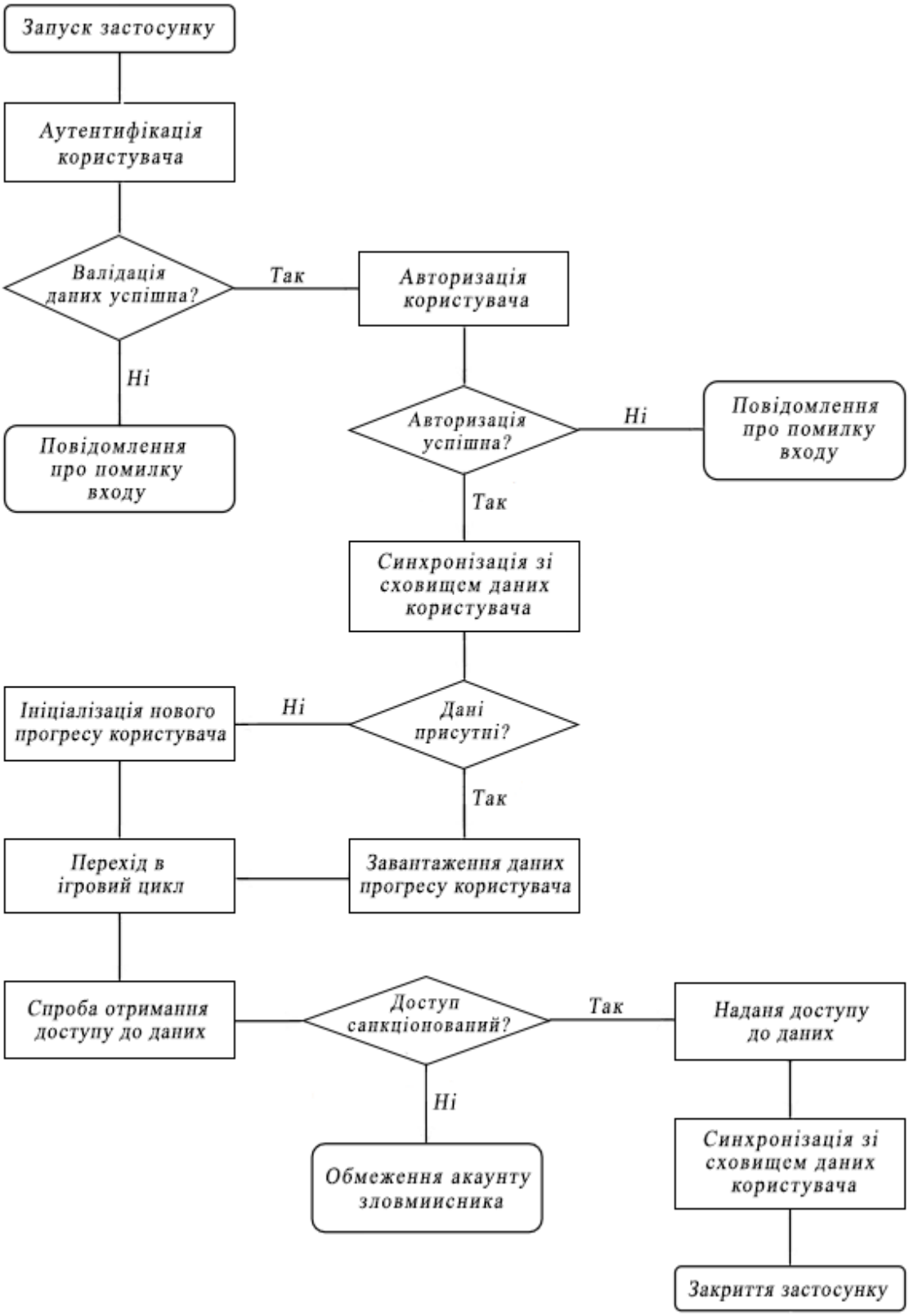

Рисунок 2.26 – Алгоритм системи безпеки застосунку

Висновки до другого розділу

У другому розділі кваліфікаційної роботи було розроблено систему безпеки програмного застосунку «*Zin: AstroForge*», що надає можливість запобігати загрозам, пов’язаним з реверсивною інженерією, підробкою даних і неавторизованим доступом, забезпечуючи надійний захист від поширених загроз безпеки в програмних застосунках.

Розділ 3. ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ СИСТЕМИ БЕЗПЕКИ ПРОГРАМНОГО ЗАСТОСУНКУ «ZIN: ASTROFORGE»

3.1 Тестування ефективності захисту від аналізаторів пам’яті програмного застосунку «Zin: AstroForge»

Тестування ефективності захисту від аналізаторів пам’яті було проведено з використанням наступних інструментів:

- «*Game Guardian*», для аналізу пам’яті в ОС *Android*;
- «*Cheat Engine*», для аналізу пам’яті в ОС *Windows*;

Найбільш популярним інструментом для аналізу та модифікації пам’яті застосунків для ОС *Android* є «*Game Guardian*». Цей інструмент працює лише на пристроях з *root*-доступом або у віртуальному середовищі. В даному випадку для тестування було використано віртуальне середовище «*Virtual Master*». Для цього було створено 64-бітну віртуальну версію *Android* 7.1.2, в якій було встановлено інструмент «*Game Guardian*» та імпортовано застосунок «*Zin:AstroForge*».

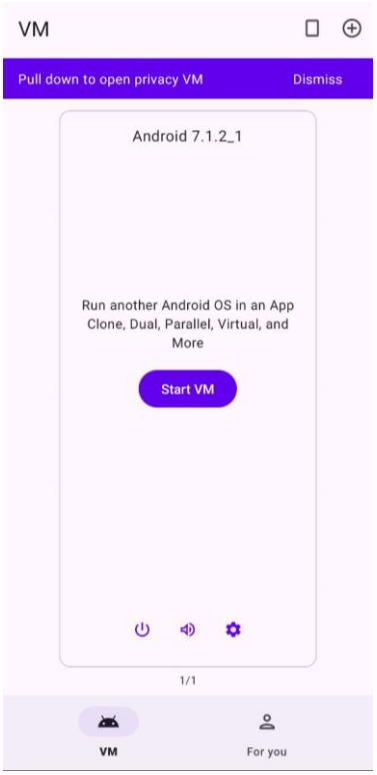

Рисунок 3.2 – Віртуальне середовище «*Virtual Master*»

Здійснено вхід в обліковий запис користувача «*Cool User*» (рис. 3.3).

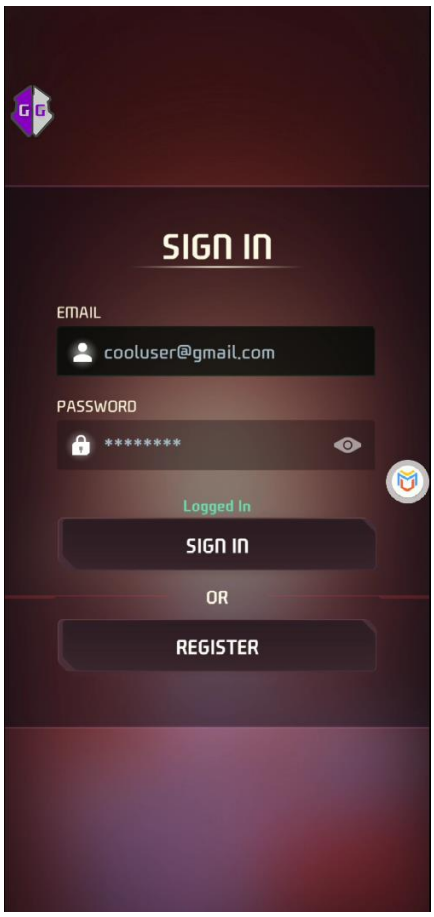

Рисунок 3.3 – Вхід в обліковий запис користувача «*Cool User*»

Було здійснено пошук змінної, що з першого погляду відповідає за баланс внутрішньоігрової валюти. ЇЇ початкове значення дорівнює «26» (рис. 3.4).

Рисунок 3.4 – Баланс валюти в головному меню

Виконано початковий пошук. Введено поточне значення «26» в поле пошуку і запущено сканування. Інструмент повернув список з 368 402 адресами пам’яті у застосунку, які містять це значення (рис. 3.5).

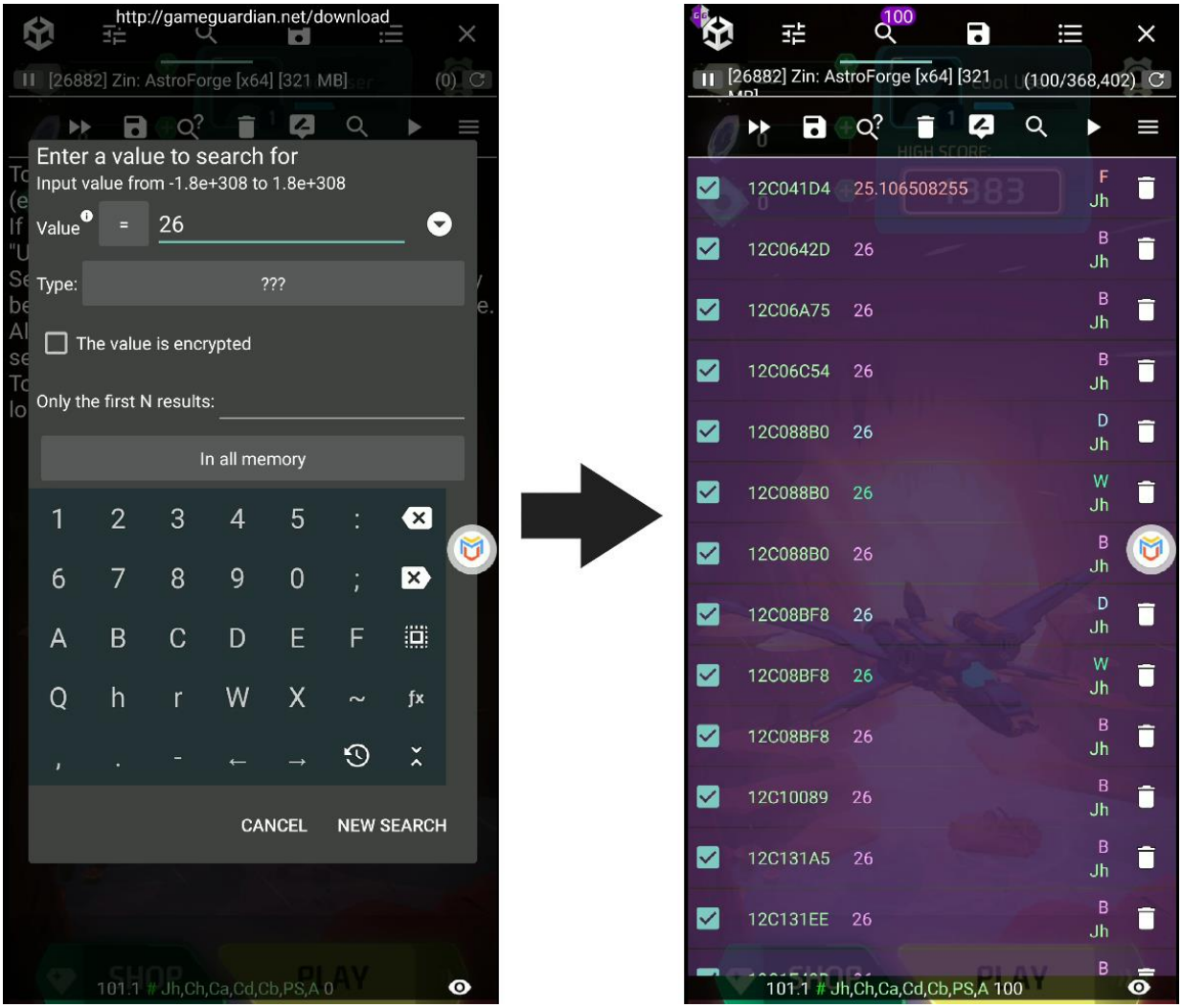

Рисунок 3.5 – Початковий пошук змінної

Після цього необхідно звузити результат пошуку. Це можна зробити шляхом уточнення даної змінної, змінивши її значення у самому застосунку і запустити повторне сканування для нового значення змінної. Цей процес повторюється, доки не залишиться мінімальна кількість адрес. Внаслідок другого пошуку список зменшився до 129 адрес, після третього – до 3 адрес (рис. 3.6).

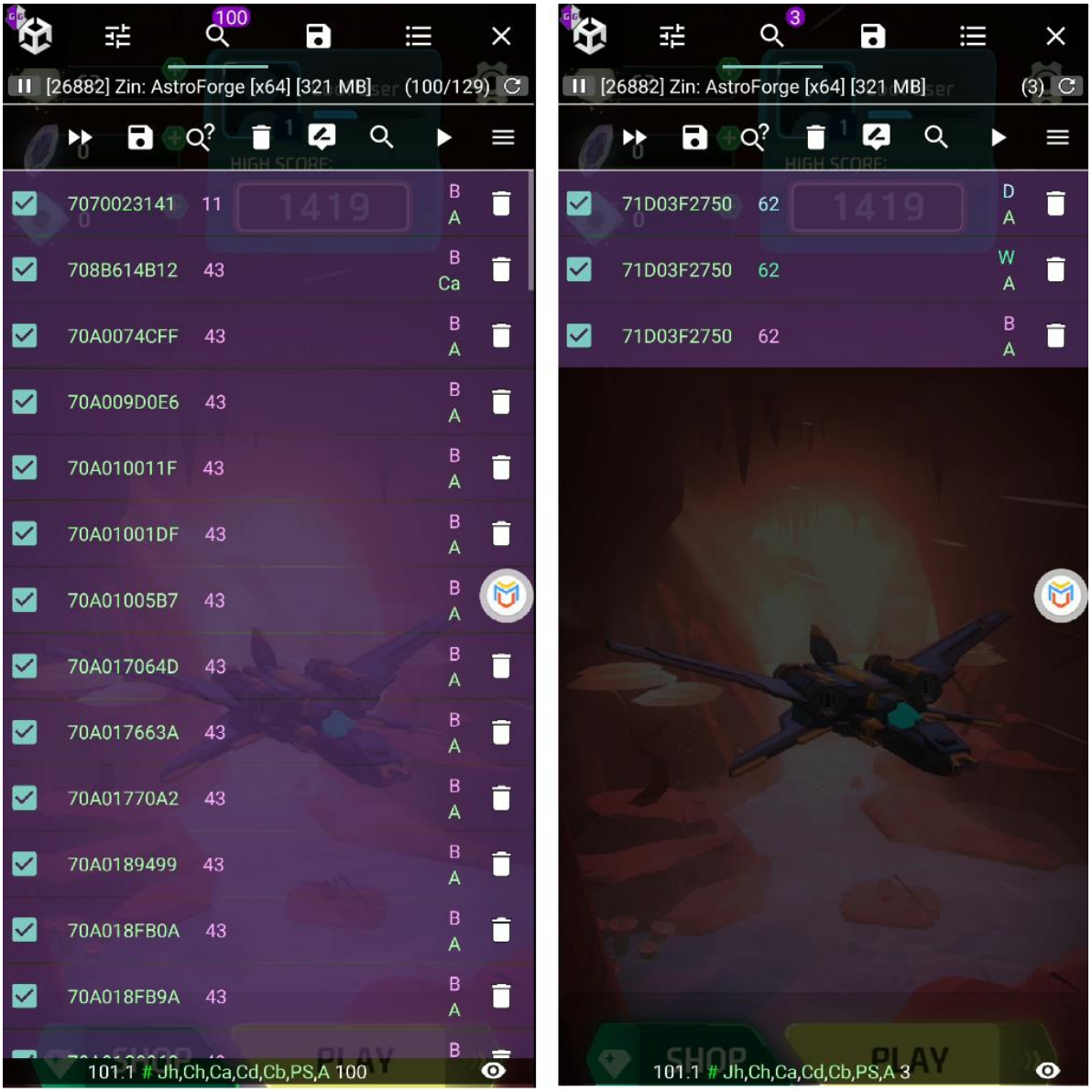

Рисунок 3.6 – Етапи звуження результатів пошуку

Після останнього четвертого уточнення даних список складався з 3 адрес зі значенням «92», які було модифіковано, змінивши їх значення на «9999». Як показано на рис. 3.7, система розпізнала несанкціоновану зміну значень змінних і відповіла обмеженням облікового запису користувача.

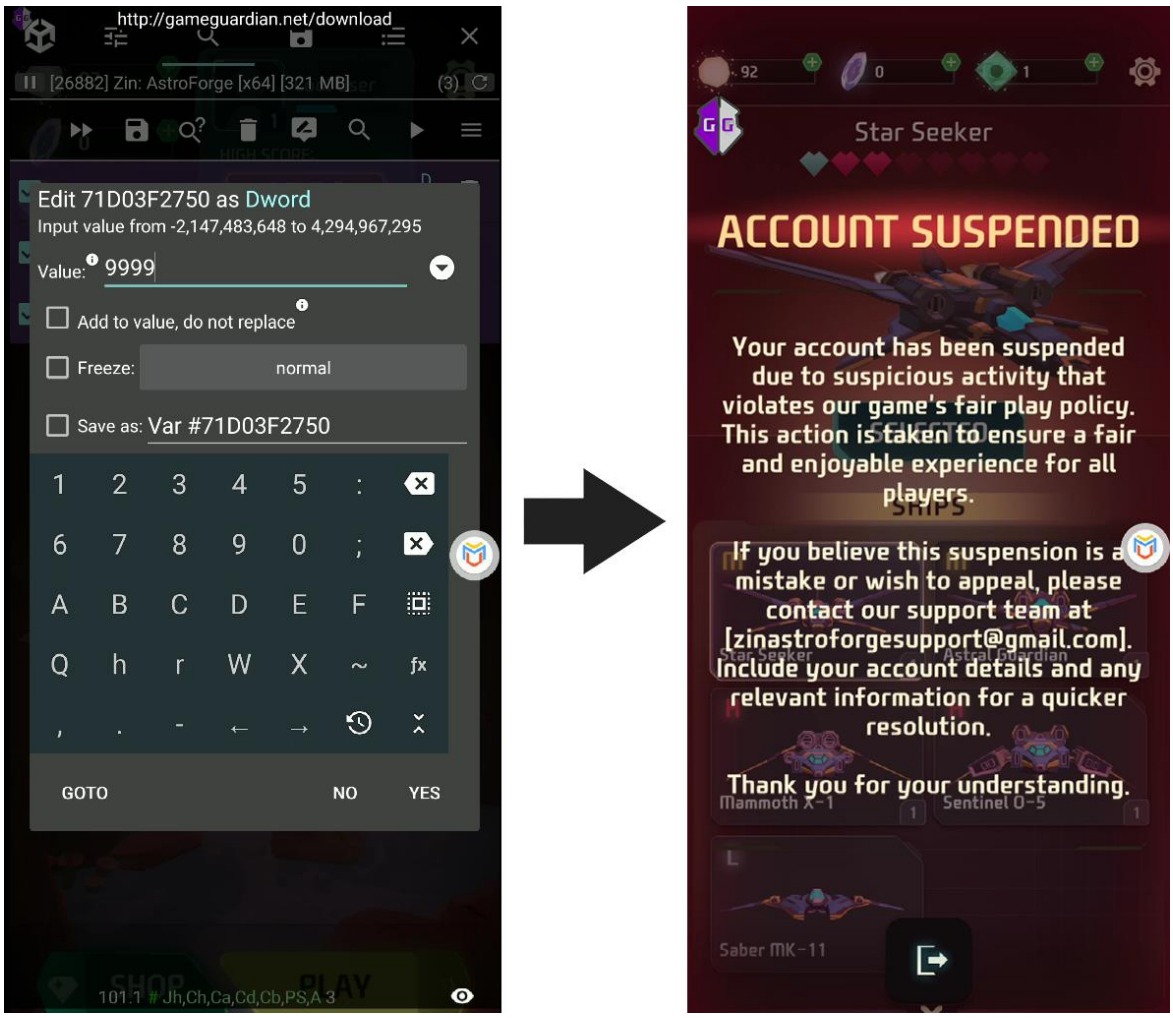

Рисунок 3.7 – Зміна значень змінних і її наслідки

Як можна бачити на рис. 3.8, після синхронізації з сервером, оригінальне значення змінної було завантажено в БД в незміненому вигляді.

Рисунок 3.8 – Оригінальне значення не змінилося

В ОС *Windows* для модифікації пам’яті застосунків використовується програма *Cheat Engine*. Вона працює за схожим принципом і має більш

широкий функціонал для аналізу процесів у *Windows*. Було здійснено пошук змінної балансу внутрішньоігрової валюти користувача аналогічними методами, використаними в *Game Guardian*. В даному випадку кінцеве значення змінної дорівнює «245» (рис. 3.9).

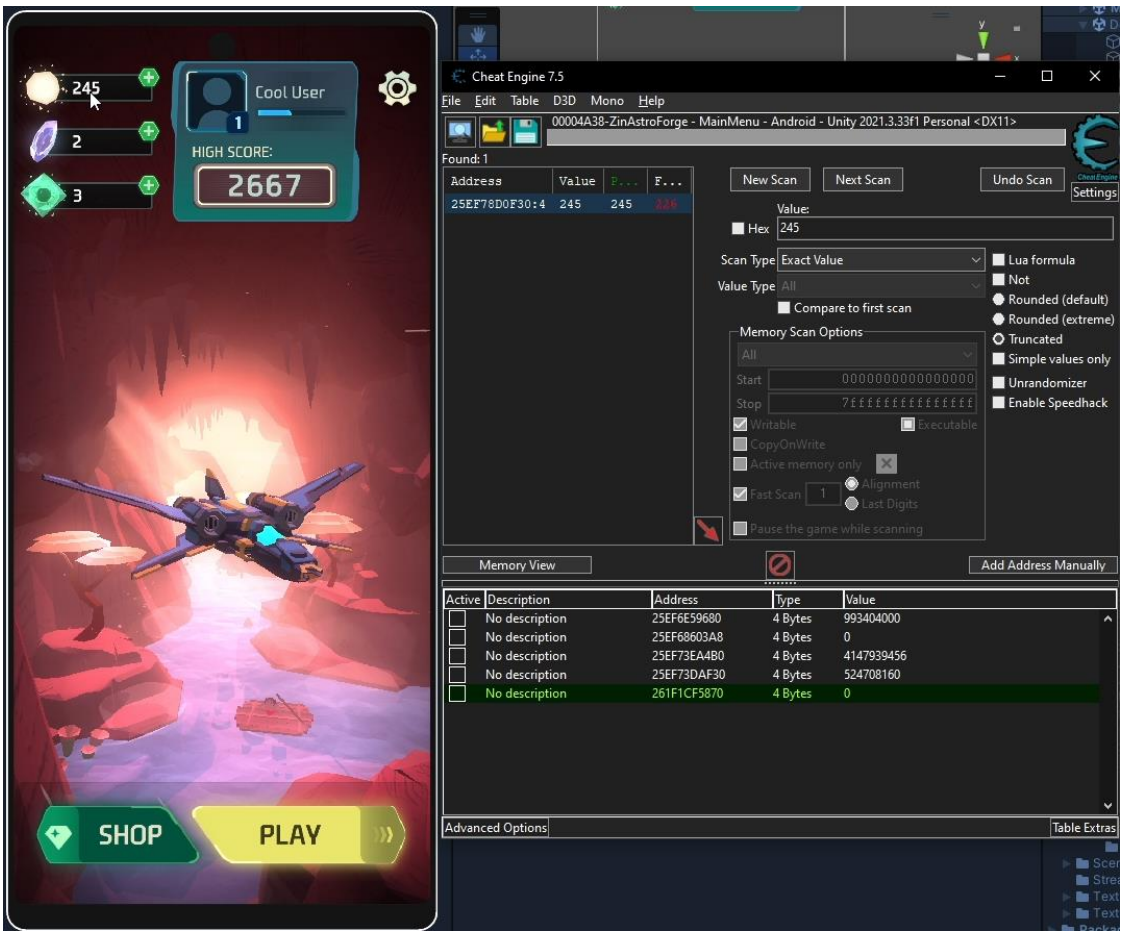

Рисунок 3.9 – Знайдено фальшиву змінну

Після цього було здійснено спробу зміни значення даної змінної на «9999» (рис. 3.10). Аналогічно з версією застосунку «*Zin: AstroForge*» для ОС *Android*, система виявила спробу втручання в пам’ять застосунку і відповіла обмеженням облікового запису зловмисника (рис. 3.11).

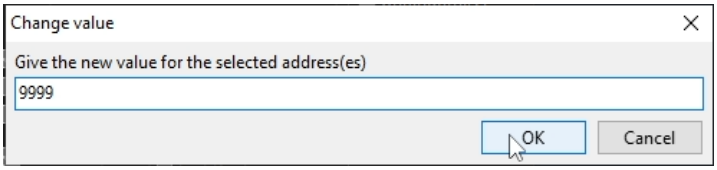

Рисунок 3.10 – Зміна значення фальшивої змінної в пам’яті

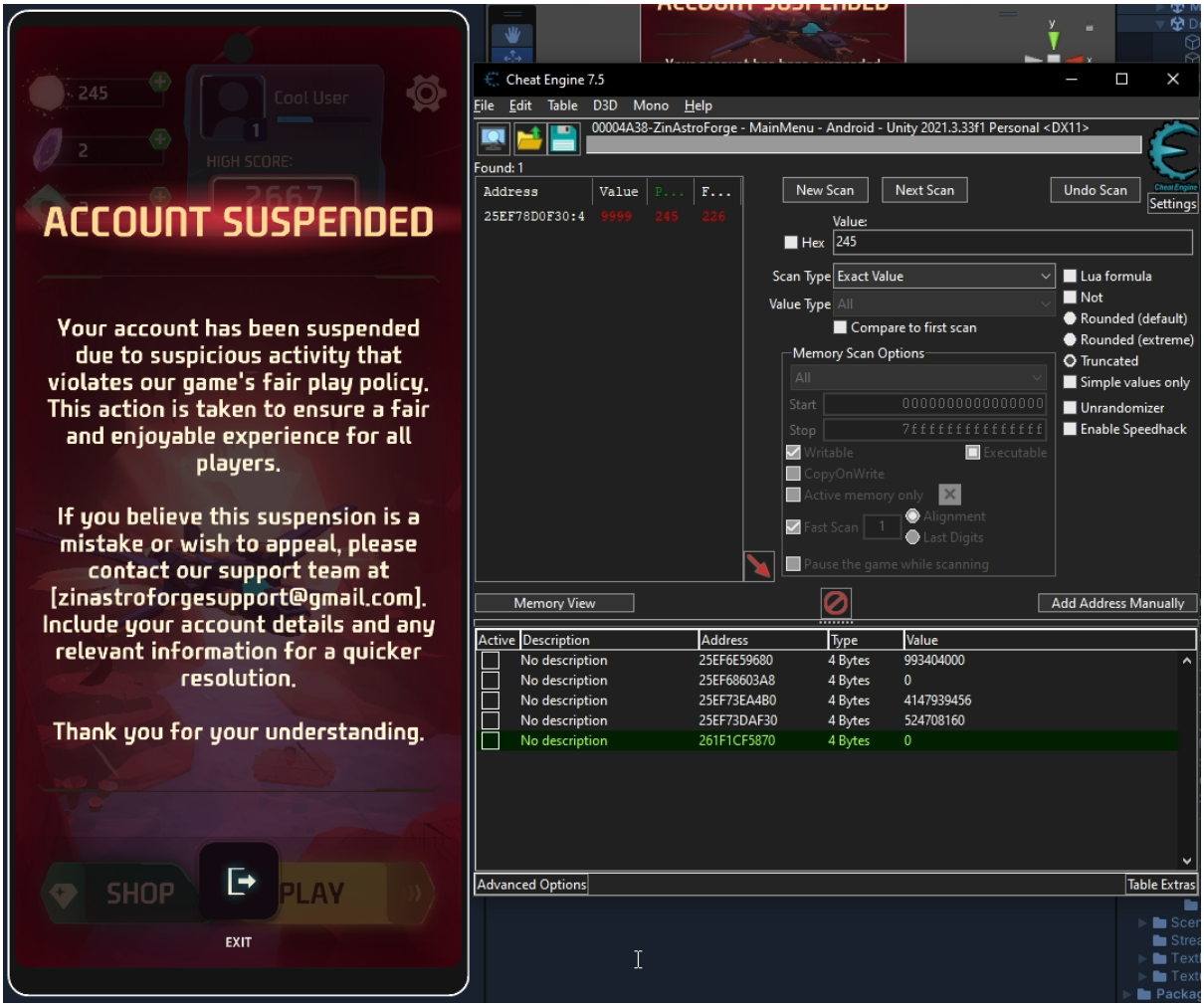

Рисунок 3.11 – Повідомлення про обмеження акаунту

Як можна бачити на рис. 3.12, спроба модифікації оригінального значення змінної була невдалою.

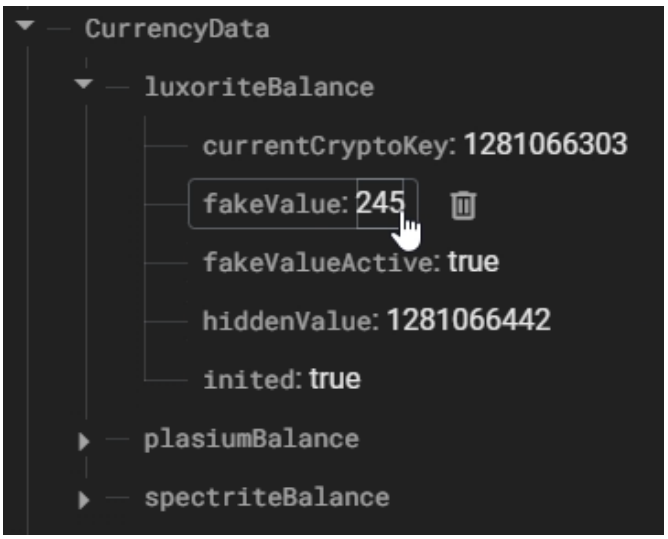

Рисунок 3.12 – Значення змінної не було змінено

3.2 Тестування ефективності захисту від реверсивної інженерії та підробки застосунку «Zin: AstroForge»

Для підробки застосунку зловмиснику необхідно виконати наступні кроки:

1) Отримати *.apk* файл зі встановленого застосунку. Це можна зробити за допомогою програми «*MT Manager*» та команди «*Extract .apk*».

2) Відкрити *.apk* файл, після чого експортувати з нього файли «*global-metadata.dat*» та «*libil2cpp.so*»:

- файл «*global-metadata.dat*» є частиною середовища виконання *Unity* і містить метадані, які використовуються для роботи зі скомпільованим кодом і керованими збірками. Він містить відомості про типи, методи, поля, властивості, збірки та інші елементи коду програми.

- файл «*libil2cpp.so*» – це скомпільована бібліотека «*shared objects*» (*.so*), яка використовується для виконання коду в середовищі *Unity*.

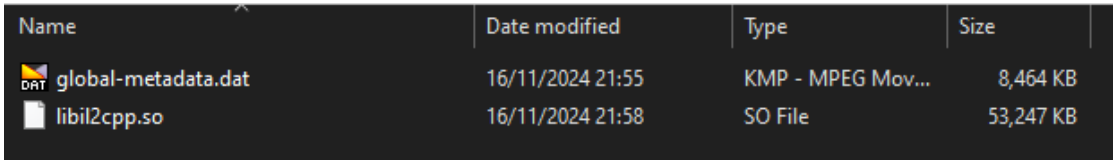

Рисунок 3.14 – Файли «*global-metadata.dat*» та «*libil2cpp.so*»

3) Використовуючи 2 отримані раніше файли, створити дамп проекту за допомогою інструменту «*IL2CPPDumper*» рис. 3.15.

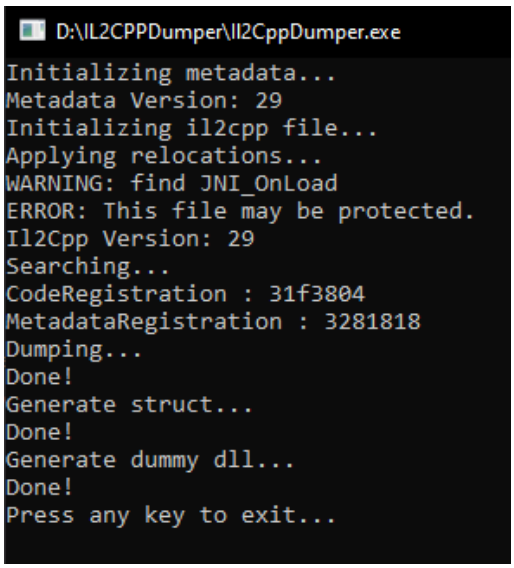

Рисунок 3.15 – Створення дампу проекту в «*IL2CPPDumper*»

4) У папці «*DummyDll*», що містить створений дамп проекту необхідно знайти файл «*Assembly-CSharp.dll*» та відкрити його в програмі для декомпіляції *.dll* файлів, наприклад, «dnSpy». У випадку з незахищеним застосунком, незважаючи на застосування *IL2CPP*, зловмисник зможе відносно легко знайти необхідні йому сигнатури методів та адреси значень у декомпільованому коді застосунку. Як показано на рис. 3.16 та рис. 3.17, більшість змінних та методів мають оригінальні назви, що дозволяє визначити, яку функцію виконує той чи інший метод.

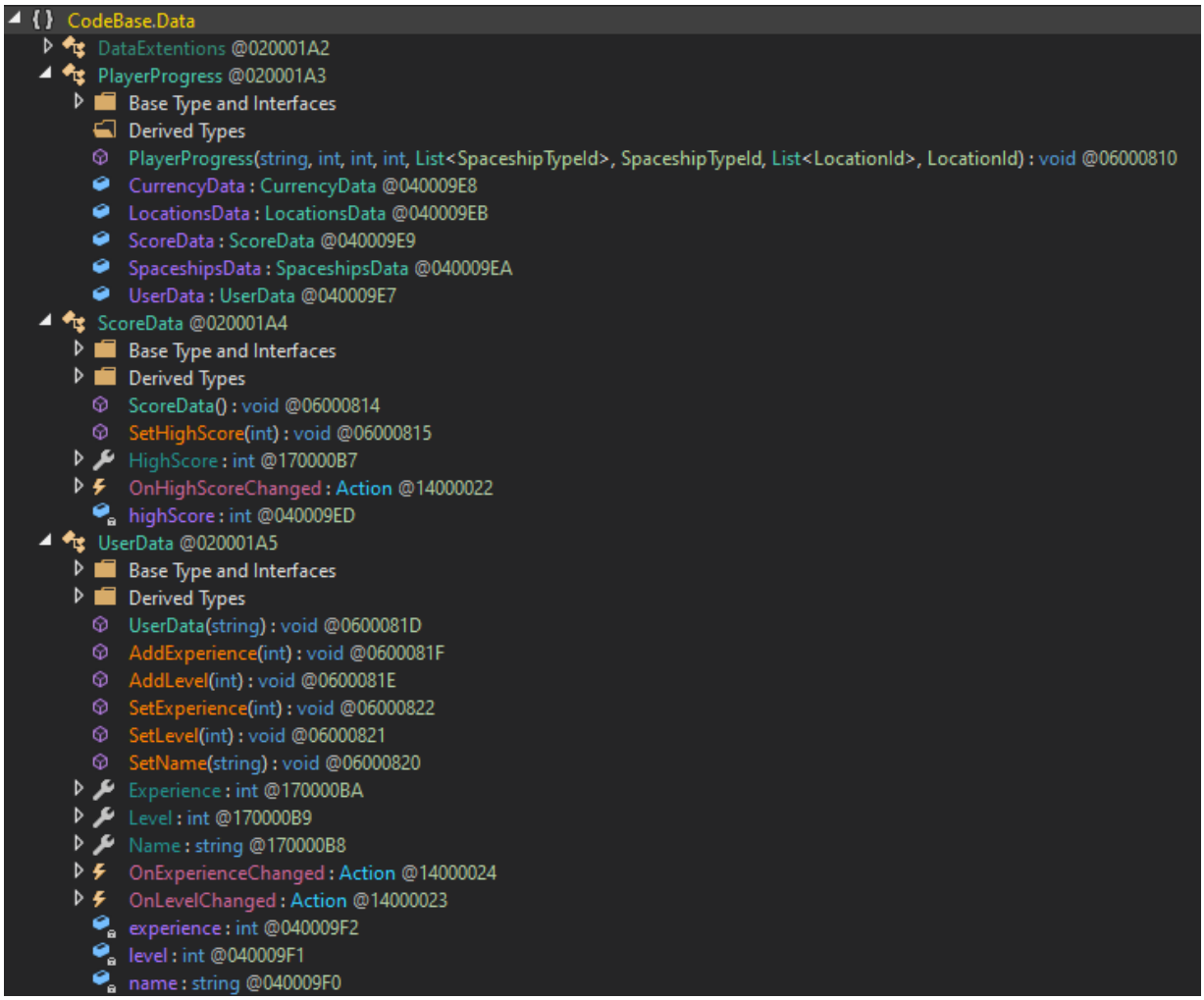

Рисунок 3.16 – Вміст простору імен «*Codebase.Data*»

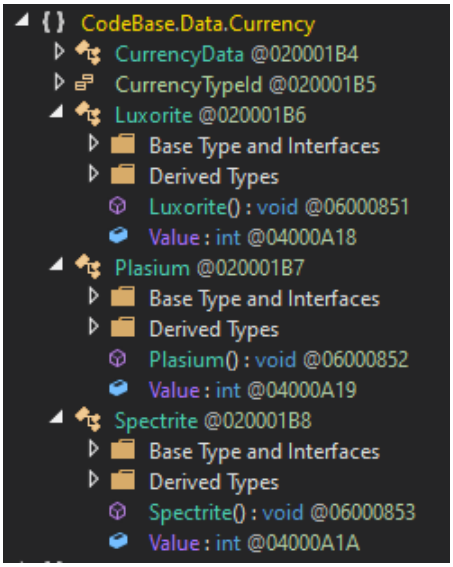

Рисунок 3.17 – Вміст простору імен «*Codebase.Data.Currency*»

5) Маючи дані адреси змінної, а також знаючи за що вона відповідає, зловмисник може модифікувати її у файлі «*libil2cpp.so*» за допомогою програми «*HexEditor*», здійснивши пошук адреси знайденої змінної та змінивши її значення на потрібне (рис. 3.18).

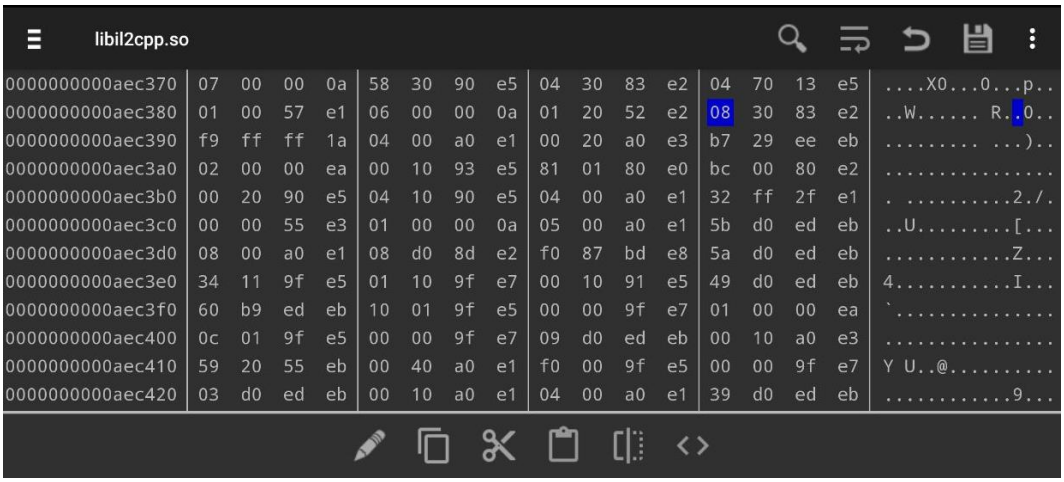

Рисунок 3.18 – Редагування пам’яті в «*HexEditor*»

6) Після цього потрібно запакувати модіфіковані файли в окремий *.apk* файл, тим самим створивши його модифіковану версію.

Оскільки в системі безпеки було реалізовано обфускацію вихідного коду, зловмисник зустрінеться з труднощами вже на етапі декомпіляції коду, оскільки для захищеного застосунку буде набагато важче знайти необхідні значення та методи для модифікації (рис. 3.19, рис. 3.20).

```
// Token: 0x060035E5 RID: 13797 RVA: 0x00002050 File Offset: 0x00000250
[Token(Token = "0x60035E5")]
[Address(RVA = "0xAE034C", Offset = "0xAE034C", VA = "0xAE034C")]
public void CNPLMAALLOG(JHANNAEBBBC HAKJENADFGB)
{
}
```

Рисунок 3.19 – Сигнатура методу після обфускації

Рисунок 3.20 – Назви класів та їх полів після обфускації

На випадок, якщо зловмиснику все ж вдалося модифікувати застосунок, при його встановленні відбувається зміна метаданих застосунку, що дозволяє визначити, що його джерелом походження не є *Google Play Store*. В даному випадку система відповіла обмеженням облікового запису зловмисника (рис. 3.21).

Рисунок 3.21 – Виявлено спробу підробки застосунку

3.3 Порівняння системи безпеки програмного застосунку «Zin: AstroForge» з існуючими рішеннями

Внаслідок проведеного експериментального дослідження ефективності створеної системи безпеки можна виділити її переваги над існуючими рішеннями, а саме:

1) Шифрування даних прогресу користувача з використанням алгоритмів, зосереджених на балансі між швидкодією та надійністю.

2) Захист коду від аналізаторів пам'яті, що унеможливлює отримання доступу до даних користувача під час виконання застосунку.

3) Система виявлення модифікації даних, що автоматично виявляє спроби модифікації критичних змінних програми та реагує на це обмеженням облікового запису зловмисника.

4) Обфускація коду, що виступає додатковим рівнем захисту від реверсивної інженерії та ускладнює процес підробки застосунку.

Результати порівняння систем безпеки наведено у табл. 3.1.

Таблиця 3.1 – Порівняння системи безпеки програмного застосунку «*Zin: AstroForge*» з існуючими рішеннями

Система безпеки	Скрипт. бекенд		Клієнт				Сервер			ОС			
	Mono	IL2CPP	Шифрування даних	Захист коду від аналізаторів пам'яті	Система виявлення модифікації даних	Обфускація коду	Система аутентифікації	Система контролю доступу до БД	Зберігання даних користувача на сервері	Android	IOS	MacOS	Windows
Subway Surfers	-	+	+	-	-	-	+	+	+	+	+	-	-
Temple Run 2	-	+	-	-	-	-	+	+	+	+	+	-	-
Among Us	+	+	+	-	-	+	+	+	+	+	+	+	+
COD: Mobile	-	+	+	-	+	+	+	+	+	+	+	-	-
Unturned	+	-	-	-	+	-	+	+	+	-	-	-	+
Hay Day	-	+	-	+	-	-	+	+	+	+	+	-	-
Zin: AstroForge	-	+	+	+	+	+	+	+	+	+	-	-	-

Висновки до третього розділу

У третьому розділі кваліфікаційної роботи проведено експериментальне дослідження ефективності системи безпеки програмного застосунку «*Zin: AstroForge*», що включає тестування ефективності захисту від аналізаторів пам’яті, а також тестування ефективності захисту від реверсивної інженерії та підробки застосунку. Проведено порівняння системи безпеки програмного

застосунку «*Zin: AstroForge*» з існуючими рішеннями, на основі якого було виділено такі переваги створеної системи, як шифрування даних прогресу користувача, захист коду від аналізаторів пам'яті, система виявлення модифікації даних та механізми обфускації коду.

ВИСНОВКИ

Проведено аналіз систем безпеки існуючих програмних застосунків, що дало можливість визначити загальні практики, сильні сторони та вразливі місця в поточних рішеннях. Аналіз також виявив прогалини в певних системах, такі як відсутність шифрування локальних даних, залежність від застарілих технологій і відсутність захисту від зворотного проектування.

Розроблено систему безпеки програмного застосунку «*Zin: AstroForge*», що надає можливість запобігати загрозам, пов’язаним з реверсивною інженерією, підробкою даних і неавторизованим доступом, забезпечуючи надійний захист від поширених загроз безпеки в програмних застосунках.

Проведено експериментальне дослідження ефективності системи безпеки програмного застосунку «*Zin: AstroForge*», що включає тестування ефективності захисту від аналізаторів пам’яті, а також тестування ефективності захисту від реверсивної інженерії та підробки застосунку. Проведено порівняння системи безпеки програмного застосунку «*Zin: AstroForge*» з існуючими рішеннями, на основі якого було виділено такі переваги створеної системи, як шифрування даних прогресу користувача, захист коду від аналізаторів пам'яті, система виявлення модифікації даних та механізми обфускації коду.

ПЕРЕЛІК ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Aleem S., Capretz L. F., & Ahmed F. Game development software engineering process life cycle: a systematic review. *Journal of Software Engineering Research and Development*, 2016. № 4. URL: https://link.springer.com/article/10.1186/s40411-016-0032-7 (дата звернення: 05.10.2024)

2. Fernandez E. B., Yoshioka N., Washizaki H., & Yoder J. Abstract security patterns and the design of secure systems. *Cybersecurity*, 2022. № 5. URL: https://cybersecurity.springeropen.com/articles/10.1186/s42400-022-00109-w (дата звернення: 15.10.2024)

3. Lundgren B., & Möller N. Defining information security. *Science and engineering ethics*, 2019. № 25. URL: https://link.springer.com/article/10.1007/s11948-017-9992-1 (дата звернення: 15.10.2024)

4. Yoder J., Barcalow J. Architectural patterns for enabling application security. In Proceedings of the 4th Conference on Patterns Language of Programming, 1997. 31 с.

5. Поліщук М. В. Використання методів захисту даних в системах безпеки Unity застосунків. Моделювання, керування та інформаційні технології (МСІТ–2024) : збірник праць учасників Міжнародної науково-практичної конференції, 2024. 367 с.

6. Cheat Engine: Memory Scanning. *Cheat Engine Wiki*. URL: https://wiki.cheatengine.org/index.php?title=Cheat_Engine:Memory_Scanning (дата звернення: 05.10.2024)

7. Müller H. A., & Kienle H. M. A small primer on software reverse engineering. *University of Victoria,* 2009. 18 с.

8. Goldwasser S., Rothblum G. N. On Best-Possible Obfuscation. *Weizmann Institute of Science*. 2007. 20с. URL: https://link.springer.com/chapter/10.1007/978-3-540-70936-7_11 (дата звернення: 10.10.2024)

9. Поліщук М. В. Unity Software: бізнес модель та її особливості. Інформаційні технології та моделювання систем : збірник праць учасників Всеукраїнської науково-практичної конференції, присвяченої 100-річчю Поліського національного університету, 12 травня 2022 р. Житомир : Поліський національний університет, 2022. 92 с.

10. Scripting backends. *Unity Documentation*. URL: https://docs.unity3d.com/Manual/scripting-backends.html#:~:text=Mono%20uses%20just-in-time,application%20before%20it%20is%20run (дата звернення: 10.10.2024)

11. IL2CPP Overview. *Unity Documentation*. URL: https://docs.unity3d.com/2019.4/Documentation/Manual/IL2CPP.html (дата звернення: 10.10.2024)

12. Mono/IL2CPP Game List. *Unknown Cheats*. URL: https://www.unknowncheats.me/forum/unity/590622-mono-il2cpp-game-list.html (дата звернення: 14.10.2024)

13. Subway Surfers Decrypted. *GitHub*. URL: https://github.com/Noobgamer0111/SubwaySurfersDecrypted (дата звернення: 14.10.2024)

14. Subway Surfers ezGUI, API Refference. *GitHib*. URL: https://github.com/lea0o0oo/subwaySurfers-ezGUI/tree/main/server#api-reference (дата звернення: 14.10.2024)

15. Temple run 2 Android cheat. *GitHub*. URL: https://gist.github.com/tk120404/8665136 (дата звернення: 14.10.2024)

16. Undetectable Among Us Cheat. *Unknown Cheats*. URL: https://www.unknowncheats.me/forum/among-us/637492-undetectable-cheat.html (дата звернення: 14.10.2024)

17. A brief history of major technical changes in Among Us. *Unknown Cheats*. URL: https://www.unknowncheats.me/forum/among-us/624161-wrote-cheat-minimal-effort-2024-a.html (дата звернення: 14.10.2024)

18. Among Us Reversal, Structs and Offsets. *Unknown Cheats*. URL: https://www.unknowncheats.me/forum/among-us/418272-reversal-structs-offsets.html (дата звернення: 14.10.2024)

19. Among Us Deobfuscation. *Unknown Cheats*. URL: https://www.unknowncheats.me/forum/among-us/426859-deobfuscation.html (дата звернення: 14.10.2024)

20. Is it possible to deobfuscate IL2CPP game? *AndnixSH*. URL: https://www.andnixsh.com/2023/05/is-it-possible-to-deobfuscate-il2cpp.html (дата звернення: 14.10.2024)

21. Ricochet Anti-Cheat: Call of Duty's Anti-Cheat Initiative. *Activision Support*. URL: https://support.activision.com/articles/ricochet-overview (дата звернення: 14.10.2024)

22. Call of Duty: Mobile Security & Enforcement Policy. *Activision Support*. URL: https://support.activision.com/cod-mobile/articles/call-of-duty-mobile-security-enforcement-policy (дата звернення: 14.10.2024)

23. My Year-Long Struggle Against a Call of Duty False Permanent Ban. *Mike Swanson's Blog*. URL: https://blog.mikeswanson.com/my-year-long-struggle-against-a-call-of-duty-false/ (дата звернення: 14.10.2024)

24. Unturned Documentation. *Smartly Dressed Games*. URL: https://docs.smartlydressedgames.com/en/stable/ (дата звернення: 15.10.2024)

25. HayDay Project. *GitHub*. URL: https://github.com/NatalieC001/HayDay-Project (дата звернення: 15.10.2024)

26. Subway Surfers Support. Sybo Helpshift. URL: https://sybo.helpshift.com/hc/en/5-subway-surfers/ (дата звернення: 16.10.2024)

27. Imangi Studios Privacy Policy. Imangi Studios. URL: https://imangistudios.com/privacy-policy/#:~:text=Imangi%20has%20a%20policy%20of,no%20longer%20is%20personally%20identifiable. (дата звернення: 16.10.2024)

28. Account Management. Innersloth. URL: https://accounts.innersloth.com (дата звернення: 16.10.2024)

29. Easy Anti-Cheat. Easy Anti-Cheat. URL: https://www.easy.ac/en-US (дата звернення: 16.10.2024)

30. Account protection. Supercell Suport. URL: https://support.supercell.com/hay-day/en/articles/ap.html (дата звернення: 16.10.2024)

31. Поліщук М. В. Використання алгоритмів обфускації коду у програмних застосунках. Безпека, технології, інновації: нові горизонти : збірник праць учасників міжфакультетської науково-практичної інтернет-конференції здобувачів вищої освіти і молодих вчених, 12 листопада 2024 р. Житомир : Поліський національний університет, 2024. 102 с.