MCIT Modeling, control & information technologies

Використання методів захисту даних в системах безпеки Unity застосунків

https://doi.org/10.31713/MCIT.2024.062

Максим Поліщук
Поліський національний університет
Житомир, Україна
imyafamillllia2@gmail.com

Анотація **— Зростання ігрової індустрії призвело до ускладнення розробки ігрових застосунків, через їх комплексність. Крім того, ігрові платформи стали ще більш привабливими для кібератак через значні прибутки від мікротранзакцій і обробку конфіденційної інформації. Це стало однією з основних причин зростання потреб в надійних заходах кібербезпеки в даній сфері. Метою даного дослідження є вивчення того, як різні методи захисту даних застосовуються в застосунках Unity, а також оцінка їх ефективності для визначення найкращих практик для реалізації функцій безпеки в даній сфері. Шляхом аналізу існуючих заходів безпеки в даному дослідженні оцінюється ефективність цих методів, визначаються їхні переваги та недоліки в контексті їх використання в системах безпеки застосунків на базі Unity. Результати демонструють, що для досягнення надійного захисту необхідний багаторівневий підхід, який поєднує заходи безпеки на стороні клієнта та на стороні сервера з урахуванням контексту проекту та його потреб.**

Ключові слова **— захист даних; кібербезпека; Unity Engine; шифрування; підробка пам'яті; захист від читів; аутентифікація**

Розвиток ігрової індустрії за останні роки призвів до значного зростання кількості користувачів та збільшення доходів. Це у свою чергу призвело до ускладнення процесу розробки ігрових застосунків, оскільки вони дуже часто являють собою комплексні 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 через 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.

Метою даного дослідження є вивчення того, як різні методи захисту даних застосовуються в застосунках Unity, а також оцінка їх ефективності для визначення найкращих практик для реалізації функцій безпеки в даній сфері.

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

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 [1].

Різниця між Mono та IL2CPP з точки зору безпеки в першу чергу пов’язана з тим, як вони здійснюють виконання та компіляцію 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.

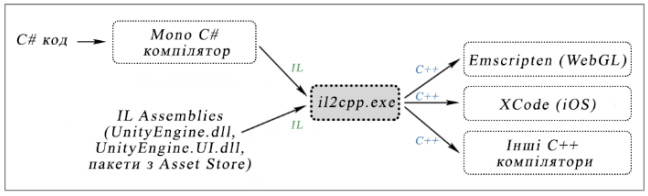

Рисунок 1. Схема роботи IL2CPP

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

1. Загрози на стороні клієнту. Неналежне зберігання чутливих даних на стороні клієнту створює можливість доступу до них, їх витоку, або втручання в ігровий процес. Зазвичай до реалізації цієї загрози призводить використання інструментів для читингу та аналізаторів пам’яті, такиих як Cheat Engine і GameGuardian, що можуть бути використані для пошуку та зміни даних застосунку.

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 [2]. 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.

До методів захисту від даних видів загроз належать:

1) Шифрування конфіденційних даних у пам'яті застосунку. Полягає у шифруванні конфіденційних ігрових даних, таких як прогрес користувача, під час зберігання в пам'яті. Замість зберігання змінних у відкритому вигляді, зберігаються їх зашифровані версії, які розшифровуються лише тоді, коли це необхідно для внутрішніх обчислень [3]. Таким чином, навіть якщо зловмисник отримає доступ до локальних даних, їх буде неможливо прочитати.

Недоліком даного методу є можливий вплив на продуктивність, в залежності від частоти операцій шифрування та дешифрування даних.

2) Інтеграція античитингових інструментів, що дозволяють виявляти ін'єкції пам'яті або несанкціоновані зміни коду. До прикладів таких інструментів належать Easy Anti-Cheat та BattlEye. Вони активно запобігають ін'єкціям пам'яті та змінам коду гри під час виконання програми, але мають низку недоліків, таких як проблеми сумісності з деякими системами та зниження продуктивності застосунку.

2. Загрози на стороні серверу. До таких загроз належать, наприклад, ризик несанкціонованого доступу до конфіденційних даних користувачів, через відсутність належних заходів контролю доступу до БД. Також можлива модифікація пакетів, які надсилаються на сервер. Наприклад, шахрай може надати серверу фальшиві дані про себе або використати вразливості програми, через які сервер сліпо довіряє даним, які надходять від клієнта [4].

До методів захисту від даних видів загроз належать:

1) Принцип недовіри до клієнту. Є невід'ємним принципом розробки онлайн-ігор та будь-яких онлайн-сервісів. Наприклад, якщо клієнт надсилає пакет, який повідомляє серверу, що користувач намагається продати товар у магазині, сервер не повинен сліпо довіряти повідомленню та виконувати операцію продажу, яка дасть користувачу гроші. Замість цього сервер повинен виконати всі необхідні перевірки та валідації, наприклад, чи дійсно користувач має даний предмет і чи знаходиться він у правильному стані для цієї операції [4].

2) Механізми аутентифікації та авторизації користувачів з контролем доступу до БД. Використання надійних механізмів аутентифікації користувачів, таких як OAuth або Firebase Security Rules, гарантує, що лише авторизовані користувачі можуть отримати доступ до даних БД та мають відповідні дозволи для виконання таких дій, як читання та запис.

3) Шифрування мережевого трафіку. Коли дані передаються по мережі, за замовчуванням вони не шифруються, що означає, що будь-хто може отримати дані за допомогою таких інструментів, як Wireshark. Це не є гарантованим рішенням усіх проблем, але це ускладнить роботу для потенційних зловмисників. [4].

3. Незахищеність коду та асетів проекту. Окремо варто виділити 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 [5]. Вона дозволяє отримати доступ до вихідного коду проекту, 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. 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. [6].

Висновок

Отже, розвиток ігрової індустрії призвів до ускладнення процесу розробки ігрових застосунків, через що потреба в надійних заходах кібербезпеки при розробці ігор стала ще більш пріоритетною, ніж раніше. Серед методів захисту даних в системах безпеки програмних застосунків на базі Unity виділено наступні:

- шифрування конфіденційних даних у пам'яті застосунку;
- інтеграція античитингових інструментів;
- принцип верифікації клієнту;
- механізми аутентифікації та авторизації користувачів з контролем доступу до БД;
- шифрування мережевого трафіку;
- використання алгоритмів обфускації коду.

Ефективність описаних заходів безпеки залежить від контексту та потреб проекту, в якому їх застосовують.

Література

1. Scripting backends. Unity Documentation. URL: https://docs.unity3d.com/Manual/scripting-backends.html#:~:text=Mono%20uses%20just-in-time,application%20before%20it%20is%20run
2. Cheat Engine: Memory Scanning. Cheat Engine Wiki. URL: https://wiki.cheatengine.org/index.php?title=Cheat_Engine:Memory_Scanning
3. Anti-Cheat Toolkit. URL: https://codestage.net/uas_files/actk/api/class_code_stage_1_1_anti_cheat_1_1_storage_1_1_obscured_prefs.html
4. Lehtonen S. Comparative study of anti-cheat methods in video games. University of Helsinki, Faculty of Science, 2020. URL: https://helda.helsinki.fi/server/api/core/bitstreams/89d7c14b-58e0-441f-a0de-862254f95551/content
5. Müller H. A., & Kienle H. M. A small primer on software reverse engineering. University of Victoria, 2009. 18 с.
6. 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