

СИСТЕМА ДИНАМІЧНОЇ ЕМУЛЯЦІЇ X64 КОДУ В СЕРЕДОВИЩІ WINDOWS

Вінницький національний технічний університет

Анотація

Запропоновано метод динамічної емуляції виконуваного коду x64 у процесах Windows без створення окремого ізолюваного середовища. Описано підхід, який дозволяє аналізувати поведінку коду без емулювання всієї операційної системи чи процесу, що підвищує ефективність та достовірність аналізу.

Ключові слова: емуляція коду, динамічна емуляція, Windows, аналіз програмного забезпечення, x64.

Abstract

A method for dynamically emulating x64 executable code in Windows processes without creating a separate isolated environment is proposed. An approach is described that allows analyzing code behavior without emulating the entire operating system or process, which increases the efficiency and reliability of the analysis.

Keywords: code emulation, dynamic emulation, Windows, software analysis, x64.

Вступ

Емуляція [1] виконуваного коду є важливим інструментом у галузі аналізу програмного забезпечення, оскільки дозволяє досліджувати поведінку програм, виявляти помилки, аналізувати безпеку та проводити оптимізацію без прямого втручання в їх роботу. Однак класичні підходи до емуляції, як правило, передбачають створення штучного ізолюваного середовища, яке не завжди адекватно відображає реальні умови виконання. Це призводить до значного зниження швидкості роботи, підвищення складності реалізації та зниження точності результатів аналізу.

Метою роботи є розробка методу динамічної емуляції x64-коду, який виконується безпосередньо в реальному середовищі процесів Windows, без створення окремого ізолюваного середовища, що дозволить підвищити продуктивність, зменшити складність реалізації та забезпечити високу достовірність отриманих результатів аналізу.

Основна частина

Класичні емулятори зазвичай створюють повністю ізолюване середовище, в якому відбувається виконання емуляції коду. Таке середовище може бути реалізоване за допомогою віртуальних машин, програмних емуляторів процесорів або спеціальних пісочниць (sandbox). Для підвищення продуктивності деякі з них використовують техніку динамічної перекомпіляції коду JIT (Just-in-Time) [2]. Це означає, що замість повторної емуляції вже виконаних інструкцій емулятор генерує динамічно скомпільований машинний код, який безпосередньо виконується процесором, значно пришвидшуючи процес емуляції. Проте навіть такі емулятори працюють у штучно створеному ізолюваному середовищі, що призводить до певних недоліків: складності емуляції системних викликів, взаємодії з операційною системою, а також зниженої точності та швидкості аналізу через відмінність емульованого середовища від реального.

Запропонована у цій роботі ідея принципово відрізняється тим, що емуляція здійснюється безпосередньо в реальному процесі Windows, без створення окремого ізолюваного середовища. Цей підхід базується на механізмі управління правами доступу до пам'яті, який підтримується операційною системою. Права доступу [3] визначають, які операції (читання, запис або виконання) дозволені для конкретної ділянки пам'яті. Якщо цільовий код спочатку мав права доступу на читання і виконання (RE), то вони змінюються лише на читання (R), якщо ж були доступні всі три права (читання, запис, вико-

нання — RWE), то вони стають читанням і записом (RW), таким чином повністю забороняючи виконання. Це означає, що будь-яка спроба виконати код з цієї області пам'яті спричинить виникнення апаратного винятку, який перехоплюється спеціальним обробником. Після перехоплення винятку зчитуються інструкції, починаючи від тієї, яка спричинила виняток, і аж до першої інструкції, що змінює показник команд (IP) [4], для забезпечення максимальної ефективності роботи. Зчитані інструкції проходять процес рекомпіляції у новий машинний код, і саме в ньому реалізуються всі необхідні перевірки та додатковий функціонал аналізу поведінки. Після завершення рекомпіляції виняток обробляється таким чином, щоб перенаправити потік виконання з оригінальної області пам'яті на новий, рекомпільований код. Абсолютно кожна інструкція проходить через процес рекомпіляції, що дозволяє отримати унікальні переваги, такі як:

1. Можливість детально відслідковувати виконання кожної окремої інструкції. Це може бути реалізовано шляхом вставки спеціальних функцій або перевірок (колбеків), які викликаються перед виконанням оригінальної інструкції.

2. Можливість контролювати доступ до пам'яті. Під час перекомпіляції можна аналізувати інструкції читання або запису пам'яті, вставляючи додаткові перевірки перед їхнім виконанням, які дозволять викликати функцію обробки події у випадку доступу до контрольованих адрес пам'яті.

3. Контроль переходів виконання за межі області емуляції. Оскільки всі інструкції, що змінюють показник команд (IP), також рекомпілюються, можна вставляти додатковий код, який аналізує адресу переходу і, за необхідності, викликає відповідну функцію обробки події.

Завдяки такому підходу суттєво спрощується та прискорюється аналіз виконуваного коду, оскільки не потрібно емуляції системних компонентів та ОС загалом. Метод дозволяє досягти високої достовірності результатів, оскільки аналіз ведеться в умовах реального процесу Windows.

Запропонований принцип динамічної емуляції також може бути ефективно використаний як механізм для захисту програмного забезпечення від несанкціонованого аналізу чи зворотної розробки. Будь-який виконуваний код програми попередньо шифрується і завантажується в пам'ять процесу у зашифрованому вигляді, при цьому пам'ять, у якій знаходиться цей код, має права доступу лише на читання (без можливості виконання). Під час першої ж спроби виконати інструкції з цієї області пам'яті динамічний емулятор перехоплює виняток, дешифрує необхідні інструкції не в місці їх початкового розташування, а в окремому проміжному буфері, після чого виконує рекомпіляцію цього коду з використанням методів обфускації (заплутування логіки коду для ускладнення аналізу). Таким чином, оригінальний код ніколи не присутній у відкритому вигляді в пам'яті процесу, а виконується лише рекомпільований, обфускований варіант коду, що значно ускладнює зворотну розробку та аналіз захищеної програми.

Концепція запропонованого методу є архітектурно-незалежною на рівні ідеї, оскільки механізми управління правами доступу до пам'яті та перехоплення винятків присутні в більшості сучасних процесорних архітектур (таких як ARM, RISC-V та інші). Проте конкретна реалізація такого методу для інших архітектур потребуватиме додаткового аналізу й адаптації з урахуванням специфіки кожної архітектури.

Висновки

Запропонований метод динамічної емуляції виконуваного коду без створення ізольованого середовища дозволяє підвищити ефективність та точність аналізу поведінки програмного забезпечення. Відсутність необхідності емуляції системних компонентів суттєво зменшує складність реалізації, а можливість детального контролю кожної інструкції забезпечує широкі можливості для моніторингу та аналізу виконуваного коду. Крім того, описаний підхід відкриває перспективи для створення ефективних засобів захисту програмного забезпечення, використовуючи динамічну дешифрацію та обфускацію коду під час його виконання. Подальші дослідження будуть спрямовані на практичну реалізацію методу, оптимізацію алгоритмів рекомпіляції та аналіз його ефективності у реальних умовах експлуатації.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. What is emulation? [Електронний ресурс] – Режим доступу до ресурсу: <https://www.imaginationtech.com/glossary/emulation/#:~:text=Emulation%20is%20the%20use%20of,a%20different%20program%20or%20device.>

2. JIT-компіляція [Електронний ресурс] – Режим доступу до ресурсу:
<https://uk.wikipedia.org/wiki/JIT-%D0%BA%D0%BE%D0%BC%D0%BF%D1%96%D0%BB%D1%8F%D1%86%D1%96%D1%8F>
3. Memory protection [Електронний ресурс] – Режим доступу до ресурсу:
https://en.wikipedia.org/wiki/Memory_protection
4. Program counter [Електронний ресурс] – Режим доступу до ресурсу:
https://en.wikipedia.org/wiki/Program_counter

Богомолов Сергій Віталійович — к.т.н., доцент каф. ОТ, Вінницький національний технічний університет, м. Вінниця, e-mail: bogomolovsergiy@vntu.edu.ua

Кожем'яко Андрій Вікторович — кандидат технічних наук, доцент кафедри обчислювальної техніки Вінницького національного технічного університету, Вінниця, e-mail: kvantron@vntu.edu.ua

Стоколос Юрій Миколайович — студент групи ІСП-21б, факультет інформаційних технологій та комп'ютерної інженерії, Вінницький національний технічний університет, Вінниця, e-mail: yura.vntu@gmail.com

Bogomolov Serhiy Vitaliyovych — Ph.D., Associate Professor, Department of OT, Vinnytsia National Technical University, Vinnytsia, e-mail: bogomolovsergiy@vntu.edu.ua

Kozhem'yako Andriy Viktorovich — Candidate of Technical Sciences, Associate Professor, Department of Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: kvantron@vntu.edu.ua

Stokolos Yuriy Mykolayovych — student of group 1SP-21b, Faculty of Information Technologies and Computer Engineering, Vinnytsia National Technical University, Vinnytsia, e-mail: yura.vntu@gmail.com