HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



NGUYỄN MINH ĐỨC

NGHIÊN CỨU VÀ XÂY DỰNG HỆ THỐNG PHÁT HIỆN HÀNH VI TẤN CÔNG BỘ NHỚ TRONG TRÒ CHƠI ĐIỆN TỬ

CHUYÊN NGÀNH : HỆ THỐNG THÔNG TIN

MÃ SỐ: 8.48.01.04

TÓM TẮT ĐỀ ÁN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI 2025

Luận văn được hoàn thành tại:

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Người hướng dẫn khoa học: **TS. NGUYỄN MINH TUẤN**

Phản biện 1:

Phản biện 2:

Luận văn này được bảo vệ trước Hội đồng chấm luận văn thạc sĩ tại Học viện Công nghệ Bưu chính Viễn thông

Vào lúc:

Có thể tìm hiểu luận văn này tại:

Thư viện của Học viện Công nghệ Bưu chính Viễn thông

MỞ ĐẦU

1. Lý do chọn đề tài

Sự phát triển bùng nổ của ngành công nghiệp trò chơi điện tử toàn cầu đang đối mặt với thách thức nghiêm trọng từ vấn nạn gian lận, đặc biệt là kỹ thuật tấn công bộ nhớ cho phép can thiệp trực tiếp vào RAM để thay đổi các chỉ số quan trọng như máu, tọa độ hay tốc độ nhằm tạo lợi thế bất công. Trước thực trạng các hệ thống phòng thủ truyền thống dựa trên chữ ký hay checksum đã trở nên lạc hậu và dễ bị qua mặt bởi các kỹ thuật tinh vi như làm rối mã hay tiêm DLL, việc phát triển một giải pháp bảo vệ chuyên biệt là vô cùng cấp thiết. Xuất phát từ sự khác biệt căn bản giữa tấn công game so với tấn công phần mềm thông thường, đề tài tập trung nghiên cứu và xây dựng một hệ thống giám sát độc lập hoạt động song song với trò chơi trên nền tảng Windows. Giải pháp này ứng dụng kỹ thuật Hook API để chủ động theo dõi, đánh chặn và phân tích các hành vi truy cập bộ nhớ đáng ngờ trong thời gian thực, giúp phát hiện sớm các gian lận mà không phụ thuộc vào các mẫu nhận diện cố định, qua đó đảm bảo tính công bằng và ổn định cho môi trường game.

2. Tổng quan về vấn đề nghiên cứu

Tấn công bộ nhớ trong trò chơi điện tử là kỹ thuật gian lận phổ biến, cho phép kẻ tấn công can thiệp trực tiếp vào vùng nhớ của tiến trình game để thay đổi các thông số quan trọng như máu, tọa độ hay điểm số, gây ảnh hưởng tiêu cực đến tính công bằng và trải nghiệm người chơi. Mặc dù các hệ thống chống gian lận hiện nay đã áp dụng nhiều biện pháp như kiểm tra chữ ký hay giám sát tài nguyên, chúng vẫn bộc lộ hạn chế đáng kể trước các kỹ thuật che giấu tinh vi như làm rối mã hay thay đổi hành vi tại thời điểm chạy. Trước thực trạng này, đề tài tập trung nghiên cứu và xây dựng một giải pháp giám sát chủ động nhằm phát hiện hành vi tấn công bộ nhớ theo thời gian thực bằng cách ứng dụng kỹ thuật phân tích hành vi hệ thống, cụ thể là Hook API. Hệ thống được thiết kế như một chương trình độc lập hoạt động song song với trò chơi trên nền tảng Windows, cho phép theo dõi các lệnh gọi hàm và truy cập bộ nhớ bất thường từ bên ngoài mà không cần can thiệp trực tiếp vào mã nguồn gốc, qua đó đảm bảo tính linh hoạt và khả năng ứng dụng rộng rãi.

3. Mục đích nghiên cứuMục tiêu cốt lõi của đề tài là thiết kế và xây dựng một hệ thống giám sát thời gian thực nhằm phát hiện các hành vi tấn công bộ nhớ trong trò chơi điện tử, qua đó hỗ trợ nhà phát triển trong công tác kiểm thử và bảo vệ sản phẩm. Để đạt được điều này, nghiên cứu tập trung vào bốn nhiệm vụ chính: thứ nhất là phân tích sâu các kỹ thuật can thiệp bộ nhớ phổ biến như sửa đổi giá trị hay chèn mã thực thi để xác định dấu hiệu nhận biết; thứ hai là phát triển hệ thống giám sát sử dụng kỹ thuật Hook API để theo dõi các hành vi truy cập nhạy cảm; thứ ba là xây dựng cơ chế phát hiện bất thường dựa trên mẫu hành vi nhằm đảm bảo độ chính xác cao và giảm thiểu cảnh báo sai; và cuối cùng là tiến hành thực nghiệm trên các kịch bản tấn công giả lập (như thay đổi thông số máu, tốc độ hay tiêm DLL) để đánh giá toàn diện hiệu năng, độ trễ và tính ổn định của giải pháp trong môi trường thực tế.

4. Đối tượng và phạm vi nghiên cứu

Đối tượng nghiên cứu: Các hành vi tấn công bộ nhớ vào trò chơi điện tử trên nền tảng Windows.

Phạm vi nghiên cứu: Tập trung vào game client-side có thể kiểm soát môi trường chạy có kết nối đến server. Nghiên cứu sử dụng tựa game CS:GO làm đối tượng thử nghiệm.

5. Phương pháp nghiên cứu

Để giải quyết bài toán đặt ra, đề tài vận dụng linh hoạt ba phương pháp nghiên cứu chủ đạo. Đầu tiên, phương pháp thu thập tài liệu được tiến hành thông qua việc tìm hiểu các công trình khoa học, sách chuyên ngành và tài liệu kỹ thuật liên quan để xây dựng cơ sở lý thuyết vững chắc. Kế đến, phương pháp phân tích và tổng hợp được áp dụng để so sánh, đánh giá các mô hình và giải pháp hiện có, từ đó nhận diện những ưu nhược điểm nhằm đề xuất hướng tiếp cận phù hợp nhất cho hệ thống. Cuối cùng, phương pháp thực nghiệm đóng vai trò then chốt trong việc kiểm chứng tính khả thi của giải pháp bằng cách triển khai hệ thống trên môi trường giả lập hoặc thực tế, với kết quả thu được là thước đo chính xác để đánh giá hiệu quả phát hiện hành vi tấn công bộ nhớ so với các mục tiêu ban đầu.

6. Nội dung chính của đề án

Nội dung chính của đề án được cấu trúc thành 3 chương:

- **Chương 1:** Tổng quan về tấn công bộ nhớ và hệ thống bảo vệ trong trò chơi điện tử
- **Chương 2:** Thiết kế và xây dựng hệ thống phát hiện tấn công bộ nhớ.
- **Chương 3:** Thử nghiệm và đánh giá hệ thống

CHƯƠNG 1: TỔNG QUAN VỀ TẤN CÔNG BỘ NHỚ VÀ HỆ THỐNG BẢO VỆ TRONG TRÒ CHƠI ĐIỆN TỬ

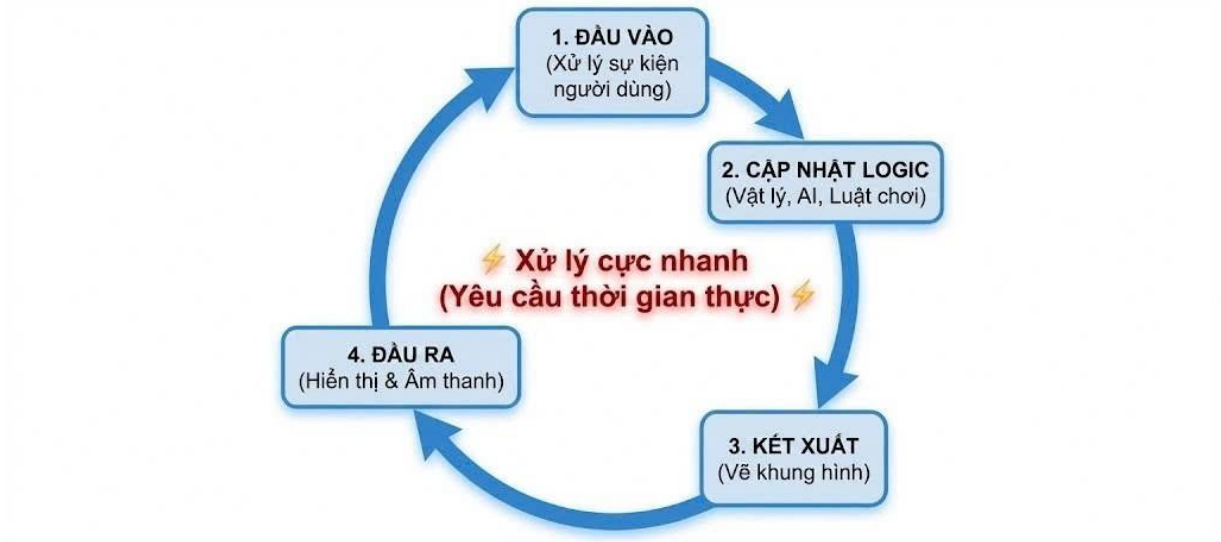
1.1. Giới thiệu chương

Chương này thiết lập cơ sở lý luận cho toàn bộ đề tài. Trong bối cảnh ngành công nghiệp game phát triển với doanh thu hàng trăm tỷ USD, trò chơi điện tử trở thành mục tiêu hấp dẫn của tin tặc. Chương này khẳng định rằng để xây dựng giải pháp phòng thủ hiệu quả, bắt buộc phải nắm vững kiến trúc bộ nhớ và các kỹ thuật tấn công (như sửa đổi giá trị, tiêm mã). Đồng thời, chương cũng phân tích thực trạng các hệ thống chống gian lận hiện hành (VAC, BattlEye) để chỉ ra nhu cầu về một giải pháp giám sát nhẹ nhàng hơn ở tầng người dùng (user-mode).

1.2. Tổng quan về trò chơi điện tử và kiến trúc bộ nhớ

1.2.1. Khái quát về trò chơi điện tử:

- Khác biệt cốt lõi của game so với phần mềm thông thường là **"Vòng lặp trò chơi" (Game Loop)**. Game liên tục cập nhật trạng thái thế giới ảo (vị trí, vật lý) với tần suất cao (60 lần/giây) để đảm bảo tính thời gian thực (Real-time).
- Do yêu cầu xử lý nhanh, bộ nhớ RAM trở thành nơi lưu trữ và biến đổi dữ liệu quan trọng nhất. Bất kỳ sự can thiệp nào vào đây đều ảnh hưởng trực tiếp đến trạng thái game.



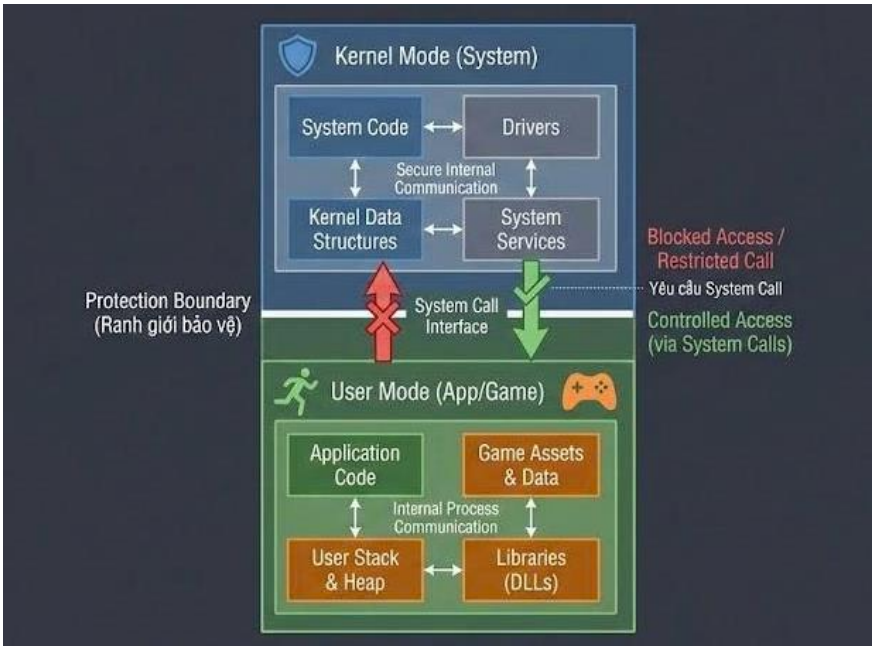
*Hình **Error! No text of specified style in document.**.1: Mô hình vòng lặp trò chơi*

1.2.2. Cấu trúc bộ nhớ trong game:

- Bộ nhớ game chứa dữ liệu nhân vật, vật phẩm, bản đồ. Các biến số quan trọng (máu, đạn, tọa độ) thường nằm trong vùng **Heap** hoặc các vùng cấp phát động.
- Do sử dụng cơ chế cấp phát động (Dynamic Allocation), địa chỉ của các biến thay đổi liên tục sau mỗi lần khởi động. Kẻ tấn công phải sử dụng kỹ thuật quét con trỏ (Pointer Scanning) để tìm ra địa chỉ thực tế.

1.2.3. Kiến trúc bộ nhớ Windows:

- Windows có cơ chế cách ly tiến trình, nhưng lại cung cấp các **API giao tiếp liên tiến trình (IPC)** phục vụ gỡ lỗi. Kẻ tấn công lạm dụng chính các tính năng hợp pháp này.
- Các hàm API nguy hiểm bị khai thác:
  - OpenProcess: Lấy quyền điều khiển (Handle) tiến trình game.
  - Read/WriteProcessMemory: Đọc/Ghi đè giá trị trong RAM.
  - VirtualAllocEx & CreateRemoteThread: Cấp phát nhớ và tạo luồng để tiêm mã độc.
- **Kết luận:** Tấn công bộ nhớ không phải do lỗi hệ điều hành, mà là sự lạm dụng tính năng có sẵn.



- *Hình **Error! No text of specified style in document.**.2: Sơ đồ không gian địa chỉ ảo*

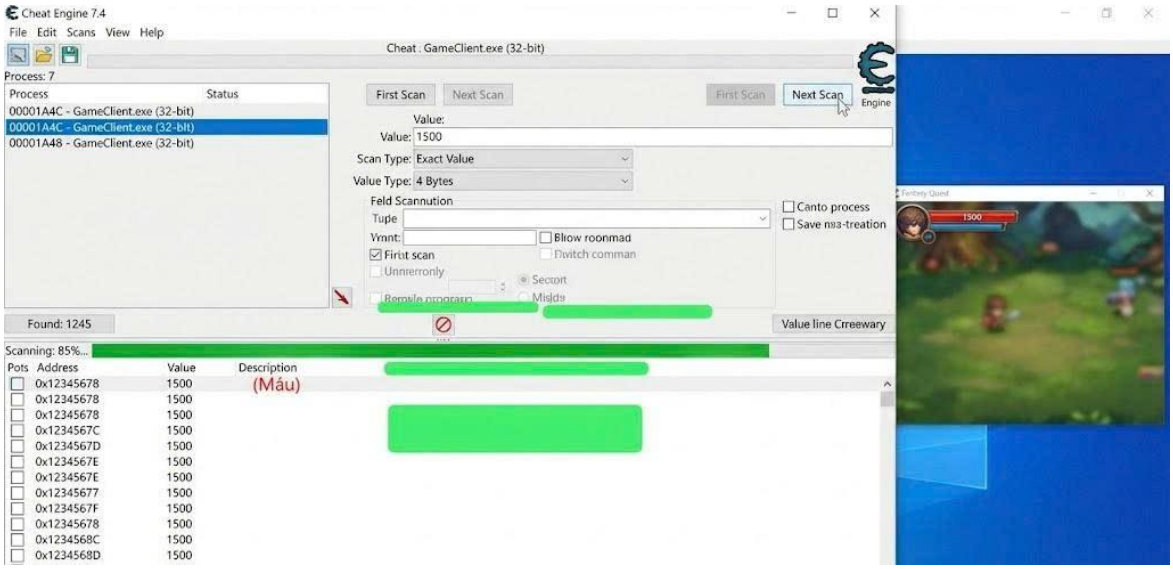
1.3. Các kỹ thuật tấn công bộ nhớ

1.3.1. Các hình thức tấn công phổ biến:

- **Hiệu chỉnh biến số (Variable Modification):** Dùng công cụ quét tìm địa chỉ biến (máu, đạn) và ghi đè giá trị mới (ví dụ: gán máu = 9999).
- **Chèn mã (Code/DLL Injection):** Tiêm thư viện động (.dll) vào tiến trình game. Mã độc chạy trong cùng không gian địa chỉ, cho phép thực hiện các tính năng phức tạp như Aimbot, Wallhack.
- **Can thiệp mã lệnh động (Runtime Patching):** Sử dụng kỹ thuật **Inline Hooking** để ghi đè mã máy ngay khi game đang chạy, nhằm vô hiệu hóa các hàm kiểm tra hoặc thay đổi logic game.

1.3.2. Công cụ tấn công bộ nhớ:

- **Cheat Engine:** Công cụ phổ biến nhất để quét bộ nhớ, tìm con trỏ và chỉnh sửa giá trị. Có khả năng scan đa dạng (chính xác, tăng/giảm, không xác định).
- **DLL Injector (Xenos, GH Injector):** Chuyên dùng để tiêm DLL vào game thông qua CreateRemoteThread hoặc Manual Mapping.
- **Debugger/Disassembler (x64dbg, IDA Pro):** Dùng để phân tích ngược (Reverse Engineering), tìm địa chỉ hàm và logic game để viết hack.



*Hình **Error! No text of specified style in document.**.3: Công cụ Cheat Engine*

1.3.3. Quy trình thực hiện memory hacking:

- **B1 - Xác định mục tiêu:** Quét bộ nhớ để tìm địa chỉ biến hoặc cấu trúc dữ liệu cần hack.

- **B2 - Truy xuất:** Dùng API (OpenProcess) để lấy quyền truy cập vào tiến trình game.
- **B3 - Can thiệp:** Dùng API (WriteProcessMemory, CreateRemoteThread) để ghi đè dữ liệu hoặc chèn mã thay đổi luồng thực thi.

1.4. Tổng quan về các hệ thống chống gian lận hiện nay

1.4.1. Các phương pháp chống gian lận:

- Kiểm tra tính toàn vẹn (Integrity Check), Nhận diện chữ ký (Signature Scanning), Giám sát hành vi (Behavior Monitoring) và Kiểm tra dữ liệu phía máy chủ (Server-side Validation).
- Xu hướng hiện đại là sử dụng **Driver tầng nhân (Kernel-mode)** để có quyền hạn cao nhất.

1.4.2. Một số hệ thống phổ biến:

- **VAC (Valve):** Hoạt động tầng User-mode, dựa trên chữ ký, cấm chậm (delayed ban). Ổn định nhưng yếu trước hack cấp thấp.
- **BattlEye:** Hoạt động tầng Kernel, chặn chủ động các API truy cập bộ nhớ. Hiệu quả cao nhưng xâm lấn sâu.
- **EAC (Easy Anti-Cheat):** Mô hình lai (Hybrid), tập trung chống tiêm mã và driver không ký số.

1.4.3. Hạn chế:

- **Quyền riêng tư:** Can thiệp quá sâu vào máy người dùng.
- **Hiệu năng:** Gây giật lag hoặc màn hình xanh (BSOD) nếu driver lỗi.
- **Kỹ thuật qua mặt:** Bị vô hiệu hóa bởi phần cứng (DMA Attack) hoặc ảo hóa.

1.5. So sánh tấn công game và tấn công ứng dụng

1.5.1. Mục tiêu tấn công:

- **Ứng dụng:** Nhắm vào CIA (Bí mật, Toàn vẹn, Sẵn sàng) để đánh cắp dữ liệu hoặc phá hoại.
- **Game:** Nhắm vào lợi thế cạnh tranh (bất tử, nhìn xuyên tường). Kẻ tấn công cần game hoạt động ổn định để "tận hưởng" lợi thế đó (tính chất ký sinh).

1.5.2. Đặc điểm môi trường:

- Game hoạt động theo thời gian thực, biến động dữ liệu cực nhanh và nhạy cảm với độ trễ. Các giải pháp bảo mật truyền thống (quét sâu, tạm dừng tiến trình) không khả thi vì gây lag.

1.5.3. Đặc thù nhận diện hành vi:

- Bảo mật game phải hiểu **ngữ nghĩa và logic trò chơi** (ví dụ: tốc độ di chuyển có hợp lý không?) và phân biệt hành vi người vs máy (thống kê độ trễ, sai số con người).

1.6. Kết luận chương 1

Chương 1 tổng kết rằng bộ nhớ RAM là chiến trường chính của các hành vi gian lận. Các kỹ thuật tấn công đều dựa trên việc khai thác API hệ điều hành. Từ những hạn chế của các giải pháp hiện tại (quá xâm lấn hoặc kém hiệu quả), chương này định hướng giải pháp của đề tài là xây dựng hệ thống giám sát tại **tầng User-mode**, sử dụng kỹ thuật **Hook API** để kiểm soát các hành vi truy cập bộ nhớ, làm tiền đề cho thiết kế chi tiết ở Chương 2.

CHƯƠNG 2: THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG PHÁT HIỆN TẤN CÔNG BỘ NHỚ

2.1. Giới thiệu chương

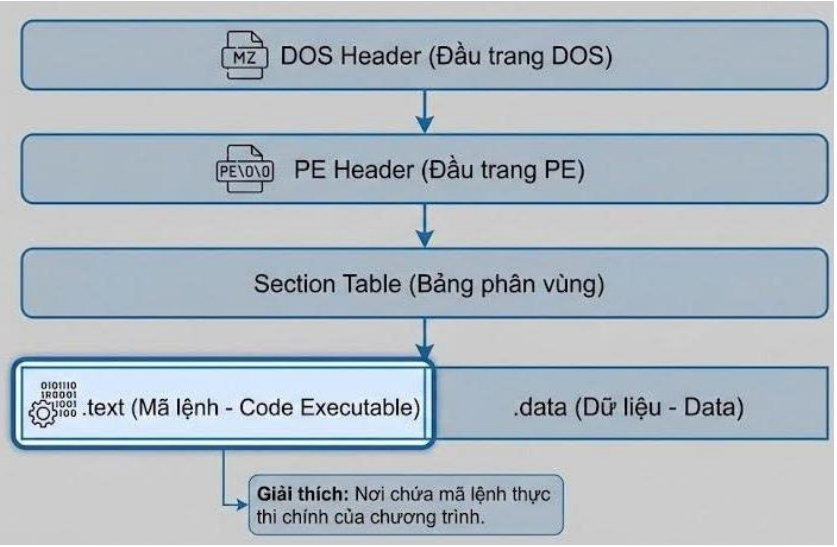
Chương 2 tập trung hiện thực hóa các mục tiêu nghiên cứu bằng việc thiết kế và xây dựng một hệ thống giám sát an ninh hoạt động tại tầng người dùng (User-mode). Thay vì can thiệp vào nhân hệ điều hành (Kernel-mode) đầy rủi ro, hệ thống lựa chọn cách tiếp cận "đứng giữa" (Man-in-the-middle) thông qua kỹ thuật Hook API. Hệ thống được phát triển bằng ngôn ngữ **C/C++** (chuẩn C++17) để đảm bảo tối ưu hóa hiệu năng, giảm thiểu độ trễ xử lý xuống mức micro-giây và tương tác trực tiếp với các API cấp thấp của Windows.

2.2. Cơ sở lý thuyết và công nghệ nền tảng

Hệ thống được xây dựng dựa trên sự am hiểu sâu sắc về kiến trúc nội tại của Windows và các định dạng tập tin thực thi:

2.2.1. Cấu trúc tập tin thực thi PE (Portable Executable):

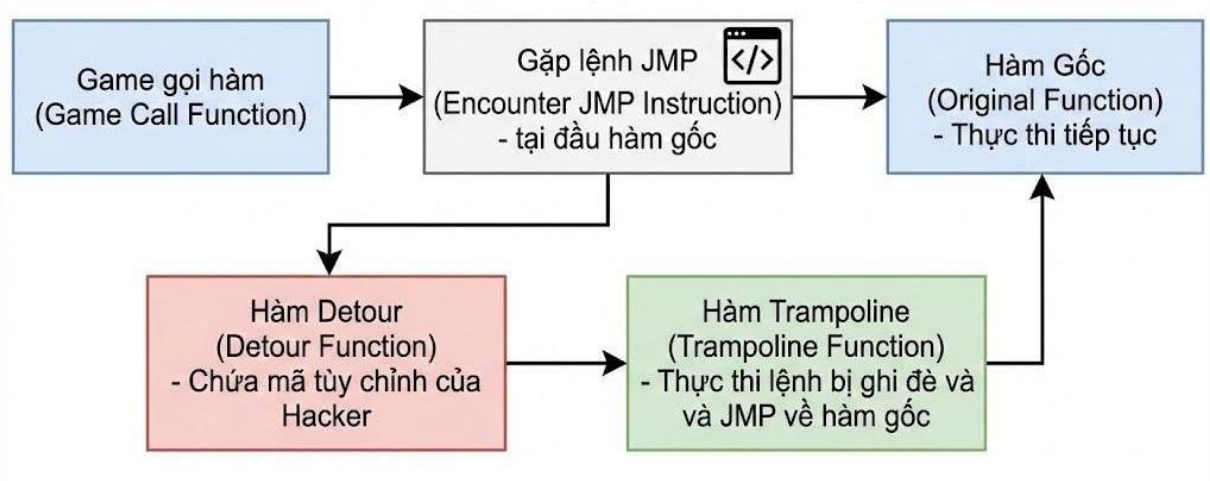
- **Header & Sections:** Phân tích cấu trúc file .exe và .dll. Quan trọng nhất là **NT Headers** (chứa Entry Point) và **Section Table**.
- **Vùng .text (Code Segment):** Đây là nơi chứa mã máy thực thi của trò chơi. Hệ thống xác định vùng này là "bất khả xâm phạm". Mọi hành vi ghi dữ liệu vào đây đều bị coi là dấu hiệu của **Code Patching** (vá mã) hoặc **Inline Hook** trái phép nhằm thay đổi logic game.
- **Bảng địa chỉ nhập (IAT):** Nơi lưu địa chỉ thực của các hàm API. Việc hiểu rõ IAT giúp hệ thống bảo vệ chống lại các kỹ thuật IAT Hooking cổ điển.



*Hình **Error! No text of specified style in document.**.4: Cấu trúc tệp tin PE (Portable Executable)*

2.2.2. Kỹ thuật Hook API:

- **Inline Hooking:** Kỹ thuật ghi đè 5 byte đầu tiên (trên x86) hoặc nhiều hơn (trên x64) của hàm API gốc bằng một lệnh nhảy không điều kiện (JMP) để chuyển hướng luồng thực thi sang hàm giám sát của hệ thống.
- **Thư viện MinHook:** Được lựa chọn làm nòng cốt kỹ thuật nhờ khả năng hỗ trợ kiến trúc x64 vượt trội, tự động tính toán địa chỉ nhảy tương đối và đảm bảo tính nguyên tử (atomic) khi ghi đè bộ nhớ, tránh gây treo ứng dụng.
- **Cơ chế Trampoline:** Để không phá vỡ chức năng của game, hệ thống tạo một vùng nhớ đệm (Trampoline) chứa các lệnh gốc đã bị ghi đè kèm theo lệnh nhảy quay về thân hàm gốc. Nếu hành vi hợp lệ, hệ thống sẽ gọi Trampoline để thực thi tác vụ như bình thường.



*Hình **Error! No text of specified style in document.**.5: Cơ chế Inline Hooking và Trampoline*

2.2.3. Các hàm API Windows nhạy cảm cần giám sát: Hệ thống tập trung vào 3 nhóm hàm chính là "cửa ngõ" của mọi cuộc tấn công:

- **Nhóm Truy cập:** OpenProcess, OpenThread (Bước đầu tiên để hacker lấy quyền điều khiển).
- **Nhóm Thao tác bộ nhớ:**
  - ReadProcessMemory: Dùng cho hack nhìn xuyên tường (ESP), radar.
  - WriteProcessMemory: Dùng để sửa máu, đạn, độ giật.
  - VirtualProtect: Dùng để đổi quyền truy cập vùng nhớ (nhằm chèn mã).
- **Nhóm Tiêm mã:** CreateRemoteThread, LoadLibrary (Dùng để tiêm DLL độc hại).

2.3. Phân tích yêu cầu và kiến trúc hệ thống

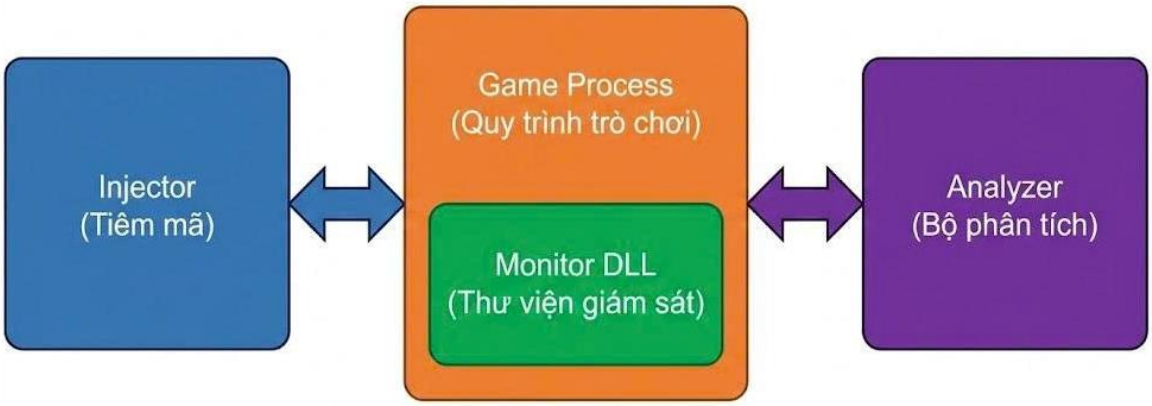
2.3.1. Phân tích yêu cầu bài toán:

- **Chức năng:** Giám sát thời gian thực (Real-time), định danh chính xác tiến trình gọi (Caller ID), phát hiện dựa trên ngữ cảnh (Context-aware), và có cơ chế phản hồi chủ động (Block/Log).
- **Phi chức năng:** Hiệu năng cao (độ trễ < 1ms), ổn định (Thread-safe), tương thích tốt (hỗ trợ x86/x64 và các phần mềm Overlay như Discord).

2.3.2. Mô hình kiến trúc tổng thể: Hệ thống áp dụng mô hình **Hướng tác tử (Agent-based)** hoạt động ngay trong không gian địa chỉ của game:

- **Game Process (Host):** Môi trường chứa.

- **Monitor Agent (Interceptor):** Cảm biến thu thập dữ liệu từ các điểm Hook.
- **Analysis Core (Analyzer):** Bộ não xử lý logic trung tâm.



*Hình **Error! No text of specified style in document.**.6: Mô hình kiến trúc tổng thể hệ thống*

2.3.3. Các thành phần chức năng chính:

- **Module Injector:** Ứng dụng chạy độc lập bên ngoài, nhiệm vụ tìm và tiêm hệ thống bảo vệ vào game.
- **Module Interceptor:** Thư viện DLL giám sát, chịu trách nhiệm thiết lập các bẫy (Hooks).
- **Module Analyzer:** Chứa các luật bảo mật (Security Rules) để phân tích hành vi.

2.4. Xây dựng các phân hệ chức năng

2.4.1. Thiết kế và cài đặt cơ chế tiêm (Injector):

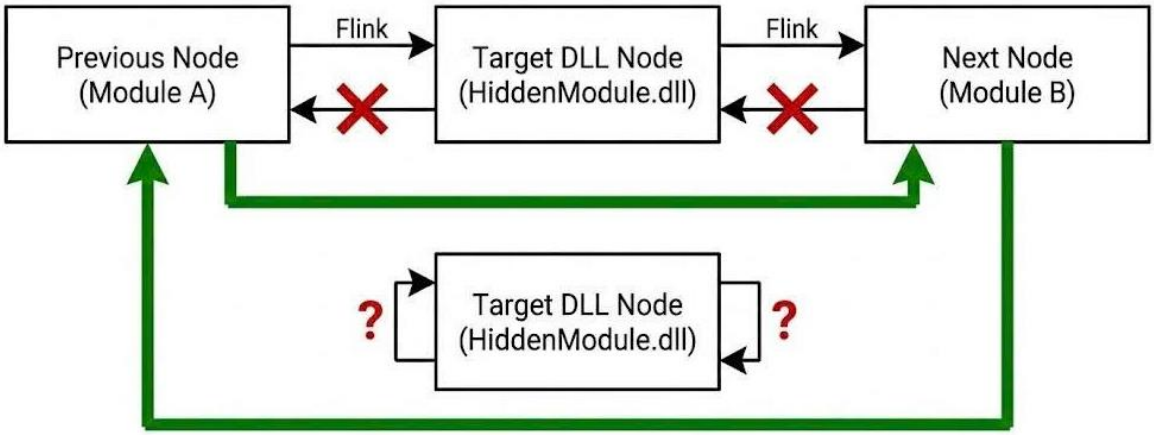
- **Quy trình tiêm 5 bước (Remote Thread Injection):**

1. CreateToolhelp32Snapshot: Tìm PID của tiến trình game.
2. OpenProcess: Mở quyền truy cập.
3. VirtualAllocEx: Cấp phát vùng nhớ trong game để chứa đường dẫn DLL.
4. WriteProcessMemory: Ghi đường dẫn DLL giám sát vào vùng nhớ đó.
5. CreateRemoteThread: Tạo luồng từ xa buộc game gọi LoadLibrary.

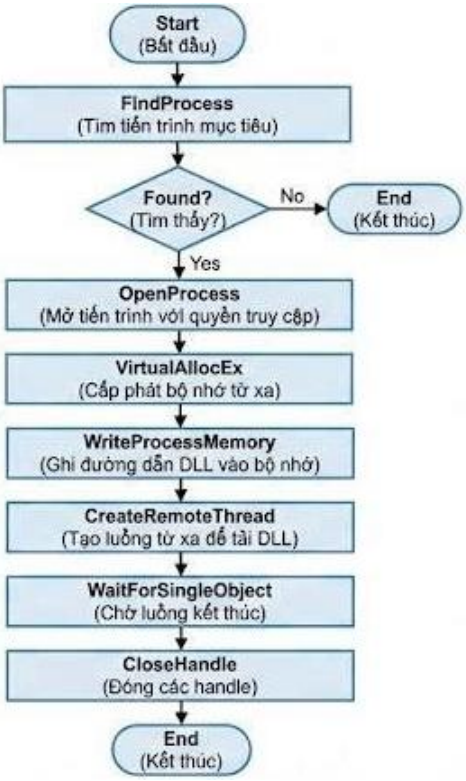
- **Kỹ thuật tàng hình (Stealth):** Sau khi nạp, thực hiện **PEB Unlinking** (thao tác trực tiếp cấu trúc danh sách liên kết InLoadOrderModuleList) để xóa node của DLL giám sát khỏi PEB, giúp ẩn mình trước các công cụ quét module.

2.4.2. Xây dựng cơ chế Hook và giám sát API (Interceptor):

- Khởi tạo môi trường MinHook (MH_Initialize).
- Tạo Hook cho từng hàm mục tiêu (MH_CreateHook): Liên kết hàm gốc (Target) với hàm giám sát (Detour_Function) và lưu địa chỉ Trampoline.
- Kích hoạt đồng loạt (MH_EnableHook) để bắt đầu chế độ bảo vệ.



*Hình **Error! No text of specified style in document.**.7: Sơ đồ danh sách liên kết PEB*



*Hình **Error! No text of specified style in document.**.8: Lưu đồ thuật toán injection*

2.4.3. Xây dựng thuật toán phát hiện bất thường (Analyzer):

- **Kiểm soát quyền truy cập (Access Control):** Tại Detour_OpenProcess, hệ thống kiểm tra tham số dwDesiredAccess. Nếu yêu cầu các quyền nguy hiểm (PROCESS_ALL_ACCESS, PROCESS_VM_WRITE) và PID nguồn không nằm trong Whitelist (Danh sách tin cậy) → Trả về ACCESS_DENIED.
- **Phát hiện hành vi (Behavioral Analysis):** Tại Detour_WriteProcessMemory, hệ thống kiểm tra địa chỉ đích. Nếu địa chỉ nằm trong phạm vi vùng .text → Xác định là tấn công Code Injection → Chặn.
- **Kiểm tra toàn vẹn (Integrity Check):** Một luồng ngầm định kỳ tính **Checksum (CRC32)** của vùng code và so sánh với giá trị gốc để phát hiện các thay đổi lén lút không qua API.

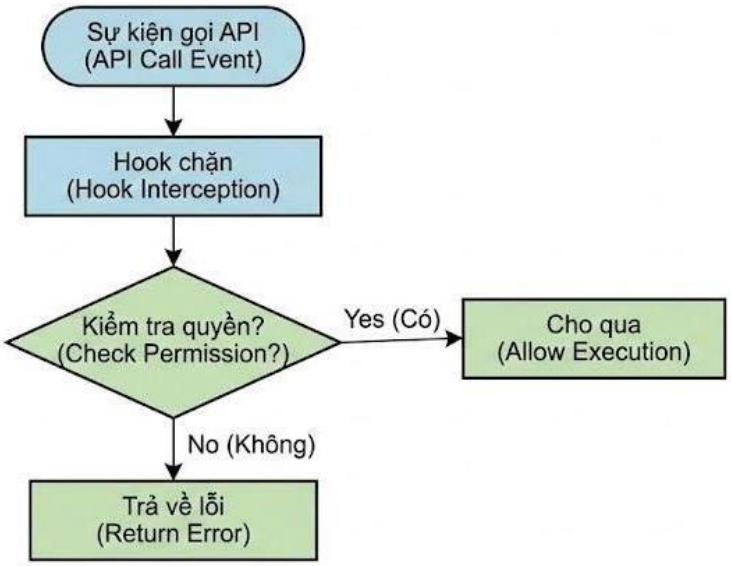
2.5. Quy trình hoạt động của hệ thống

2.5.1. Quy trình khởi tạo:

- Injector chạy → Quét tìm Game → Tiêm DLL giám sát → DLL được nạp, hàm DllMain chạy → Cài đặt Hook → Thực hiện PEB Unlinking → Hệ thống chuyển sang trạng thái thường trực (Idle).

2.5.2. Luồng xử lý sự kiện khi phát hiện can thiệp:

- **Sự kiện:** Hacker/Tool gọi API nhạy cảm.
- **Đánh chặn:** Inline Hook kích hoạt, luồng thực thi bị chuyển sang hàm Detour.
- **Phân tích:** Analyzer trích xuất thông tin: Ai gọi (Caller PID)? Gọi hàm gì? Tham số là gì (Địa chỉ, Quyền)?
- **Quyết định:**
  - **Nếu Vi phạm:** Không gọi Trampoline, trả về mã lỗi giả lập (đánh lừa Tool hack), ghi log cảnh báo.
  - **Nếu Hợp lệ:** Gọi Trampoline để thực thi hàm gốc của Windows, trả kết quả cho game như bình thường.



*Hình **Error! No text of specified style in document.**.9: Luồng xử lý sự kiện*

2.6. Kết luận chương 2

Chương 2 đã hoàn thành việc xây dựng trọn vẹn hệ thống giám sát tại tầng User-mode. Việc kết hợp sức mạnh kiểm soát bộ nhớ của C++ và thư viện MinHook giúp hệ thống đạt được sự cân bằng tối ưu giữa hiệu năng và khả năng bảo mật. Các thuật toán phân tích được thiết kế đa lớp (Quyền hạn - Hành vi - Toàn vẹn) tạo nên một lá chắn chủ động, có khả năng vô hiệu hóa các công cụ gian lận phổ biến ngay từ bước xâm nhập đầu tiên mà không làm ảnh hưởng đến trải nghiệm của người chơi chân chính.

CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ HỆ THỐNG

3.1. Giới thiệu chương

Chương 3 tập trung vào việc kiểm chứng tính khả thi và hiệu quả của hệ thống đã thiết kế trong Chương 2 thông qua môi trường thực tế. Mục tiêu cốt lõi là trả lời ba câu hỏi lớn: Hệ thống có triển khai thành công vào tiến trình game không? Có ngăn chặn được các công cụ tấn công phổ biến không? Và có gây ảnh hưởng tiêu cực đến hiệu năng (FPS) của trò chơi hay không?

3.2. Môi trường và công cụ thử nghiệm

Để đảm bảo tính khách quan, môi trường thử nghiệm được thiết lập sát với điều kiện thực tế của người chơi game.

3.2.1. Cấu hình thử nghiệm:

- **Phần cứng:** CPU Intel Core i7 (đa nhân để test đa luồng), RAM 16GB, GPU RTX 2060 Super (để game chạy ở FPS cao >144, giúp dễ nhận biết độ trễ nếu có).
- **Phần mềm:** Windows 10/11 64-bit, Visual Studio 2019/2022.

3.2.2. Đối tượng thử nghiệm:

- **Tựa game: Counter-Strike: Global Offensive (CS:GO)**.
- **Lý do chọn:** Đây là game bắn súng nhạy cảm với độ trễ, sử dụng kiến trúc x86 tiêu biểu (Source Engine), và hỗ trợ chế độ -insecure cho phép thử nghiệm an toàn mà không bị hệ thống VAC ban nick.



*Hình **Error! No text of specified style in document.**.10: Giao diện game CS:GO và Console Debug*

3.2.3. Bộ công cụ tấn công giả lập:

- **Cheat Engine:** Đại diện cho nhóm công cụ quét bộ nhớ, sửa giá trị và tìm con trỏ.
- **Xenos Injector:** Đại diện cho nhóm công cụ tiêm mã độc (DLL Injection).
- **Custom Exploit Tool:** Tool tự viết để spam lệnh đọc/ghi bộ nhớ tần suất cao (Stress test).

3.3. Các kịch bản thử nghiệm

Hệ thống được kiểm tra qua 5 kịch bản mô phỏng các giai đoạn tấn công khác nhau và kiểm tra độ ổn định.

3.3.1. Kịch bản 1: Tấn công truy cập tiến trình (Access Attack)

- **Mục tiêu:** Kiểm tra khả năng bảo vệ của Hook OpenProcess.
- **Thực hiện:** Dùng Cheat Engine cố gắng mở tiến trình csgo.exe.
- **Kết quả kỳ vọng:** Hệ thống phát hiện tiến trình lạ, trả về mã lỗi ACCESS_DENIED, Cheat Engine không thể đọc được danh sách bộ nhớ.

3.3.2. Kịch bản 2: Tấn công sửa đổi giá trị (Memory Tampering)

- **Mục tiêu:** Kiểm tra Hook WriteProcessMemory.
- **Thực hiện:** Dùng tool spam lệnh ghi giá trị 9999 vào địa chỉ biến Máu (Health) và Đạn (Ammo).
- **Kết quả kỳ vọng:** Lệnh ghi bị chặn đứng, chỉ số máu trong game vẫn giảm bình thường khi nhận sát thương, log hệ thống ghi nhận cảnh báo.

3.3.3. Kịch bản 3: Tấn công tiêm mã (Code Injection)

- **Mục tiêu:** Kiểm tra Hook CreateRemoteThread và LoadLibrary.
- **Thực hiện:** Dùng Xenos Injector cố gắng tiêm file Hack.dll vào game.
- **Kết quả kỳ vọng:** Luồng từ xa không được tạo, DLL hack không được nạp vào bộ nhớ game.

3.3.4. Kịch bản 4: Kiểm thử tính tương thích (Compatibility)

- **Mục tiêu:** Đảm bảo hệ thống không chặn nhầm ("giết nhầm hơn bỏ sót") các phần mềm hợp pháp.
- **Thực hiện:** Chạy game cùng với **Discord** (Overlay), **OBS** (Quay màn hình), **Unikey** (Bộ gõ).

- **Kết quả kỳ vọng:** Hệ thống nhận diện được chữ ký/tên của các phần mềm này trong Whitelist và cho phép chúng hoạt động bình thường.

3.3.5. Kịch bản 5: Kiểm thử độ bền (Stability)

- **Mục tiêu:** Đảm bảo không gây rò rỉ bộ nhớ (Memory Leak).
- **Thực hiện:** Treo game liên tục trong **4 giờ**.
- **Kết quả kỳ vọng:** Mức RAM và CPU ổn định, không tăng dần theo thời gian.

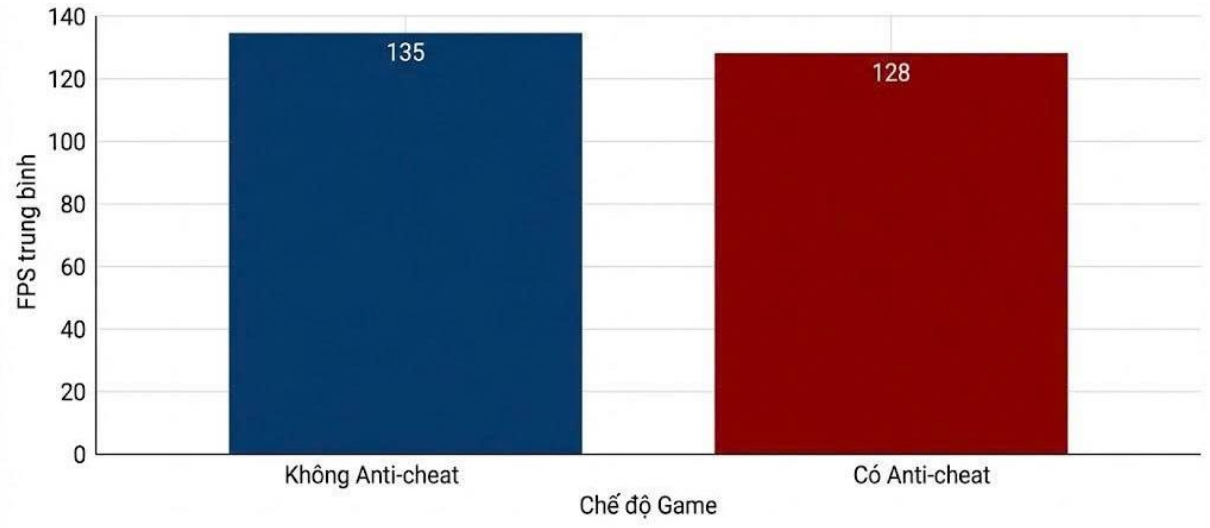
3.4. Kết quả thực nghiệm và Phân tích

3.4.1. Kết quả về chức năng an ninh:

- **Tỷ lệ phát hiện:** Đạt **100%** với các công cụ User-mode thử nghiệm.
- **Chi tiết:**
  - Cheat Engine bị vô hiệu hóa ngay bước truy cập (OpenProcess).
  - Tool ghi bộ nhớ bị chặn, bảo vệ toàn vẹn dữ liệu gameplay.
  - Injector báo lỗi, không thể đưa mã độc vào game.

3.4.2. Đánh giá hiệu năng và tính ổn định:

- **Tác động lên FPS:** Sụt giảm trung bình khoảng **1.65%** (tương đương mất ~4 FPS trên tổng số 240 FPS). Mức giảm này là không đáng kể và khó nhận biết bằng mắt thường.
- **Tài nguyên:** CPU tăng thêm **< 0.5%**, RAM chiếm dụng thêm **4-8 MB**.
- **Độ ổn định:** Game hoạt động mượt mà suốt 4 giờ test, không xảy ra hiện tượng crash hay giật lag (stuttering).



*Hình **Error! No text of specified style in document.**.11: Biểu đồ so sánh FPS (Hiệu năng)*

3.5. Thảo luận về các hạn chế

Mặc dù kết quả khả quan, chương này cũng thẳng thắn nhìn nhận 3 hạn chế kỹ thuật còn tồn tại:

3.5.1. Hạn chế đối với tấn công mức nhân (Kernel Level):

- Hệ thống hoạt động ở User-mode (Ring 3), nên hoàn toàn bất lực trước các công cụ hack sử dụng Driver Kernel-mode (Ring 0) để truy cập trực tiếp bộ nhớ vật lý. Đây là giới hạn kiến trúc.

3.5.2. Nguy cơ bị phát hiện ngược:

- Do sử dụng phương pháp tiêm DLL, mã của hệ thống giám sát nằm trong bộ nhớ game. Các tool hack tinh vi có thể quét chữ ký (Signature Scanning) để phát hiện ra Anti-cheat và tự động tắt hack hoặc tìm cách gỡ bỏ Hook (Unhooking).

3.5.3. Hạn chế về cơ chế khởi tạo (Race Condition):

- Vấn đề "Tranh chấp thời gian": Nếu Hacker mở Tool Hack **trước** khi Anti-cheat kịp khởi chạy và cài đặt Hook, Tool Hack có thể đã lấy được một Handle hợp pháp. Khi đó, các Hook cài sau sẽ không chặn được các kết nối đã thiết lập từ trước.

3.6. Kết luận chương 3

Chương 3 đã hoàn tất quy trình kiểm chứng. Kết quả thực nghiệm khẳng định hệ thống đáp ứng tốt các yêu cầu đề ra: bảo vệ hiệu quả trước các tấn công User-mode phổ biến, tương thích tốt với phần mềm bổ trợ và tối ưu hóa hiệu năng xuất sắc. Các

hạn chế được chỉ ra là cơ sở quan trọng để định hướng nâng cấp hệ thống lên mô hình **Launcher** (khởi chạy trước game) hoặc phát triển Driver trong tương lai.

KẾT LUẬN

Đề tài "Nghiên cứu và xây dựng hệ thống phát hiện hành vi tấn công bộ nhớ trong trò chơi điện tử" xuất phát từ nhu cầu cấp thiết trong việc đảm bảo tính công bằng và toàn vẹn cho môi trường giải trí số, trước sự gia tăng phức tạp của các kỹ thuật gian lận hiện đại. Qua quá trình nghiên cứu từ lý thuyết đến thực nghiệm, đề tài đã hoàn thành các mục tiêu đề ra và đạt được những kết quả cụ thể sau:

Đề tài đã hệ thống hóa được các đặc thù kỹ thuật của tấn công bộ nhớ (cơ chế truy cập, sửa đổi giá trị, tiêm mã). Nghiên cứu đã chỉ ra rằng các giải pháp chống gian lận thương mại hiện hành (như BattlEye, EAC) dù hiệu quả nhưng lại phụ thuộc quá nhiều vào đặc quyền Kernel-mode, gây ra những rào cản lớn về quyền riêng tư, chi phí triển khai và khó tiếp cận đối với các dự án nghiên cứu độc lập.

Đề tài đã xây dựng thành công hệ thống giám sát và phát hiện tấn công hoạt động ở tầng User-mode. Đóng góp cốt lõi của giải pháp là việc ứng dụng kỹ thuật Inline Hooking để chuyển hóa các tương tác hệ thống thô sơ (API Calls) thành các ngữ cảnh bảo mật có ý nghĩa. Cơ chế này cho phép hệ thống chuyển từ thế "quét chữ ký thụ động" sang "ngăn chặn hành vi chủ động", giúp nhận diện và vô hiệu hóa các nỗ lực can thiệp ngay tại thời điểm chúng khởi phát.

Quá trình kiểm thử trên môi trường thực tế với tựa game Counter-Strike: Global Offensive (CS:GO) đã cung cấp những bằng chứng định lượng thuyết phục. Hệ thống đã chứng minh khả năng ngăn chặn hoàn toàn các kịch bản tấn công giả lập từ các công cụ phổ biến (như Cheat Engine, Xenos Injector) mà vẫn duy trì mức độ ổn định cao với độ trễ xử lý không đáng kể (ảnh hưởng chưa tới 2% FPS).

Kết quả của đề tài khẳng định rằng việc áp dụng các kỹ thuật lập trình hệ thống và giám sát API là hướng đi đúng đắn để giải quyết bài toán bảo mật game mà không cần can thiệp sâu vào nhân hệ điều hành. Giải pháp không chỉ khắc phục được hạn chế về tính tương thích của các giải pháp cũ mà còn đáp ứng tốt các yêu cầu khắt khe về hiệu năng trải nghiệm người dùng.

Giải pháp này có ý nghĩa thực tiễn cao, đặc biệt đối với các nhà phát triển game độc lập (Indie) hoặc các môi trường kiểm thử bảo mật, nơi cần một công cụ giám sát nhẹ nhàng, dễ tích hợp và chi phí thấp.

Mặc dù đã có nhiều nỗ lực, song với khuôn khổ giới hạn của một đề án tốt nghiệp và năng lực còn hạn chế, đề tài khó tránh khỏi những thiếu sót (như khả năng phòng thủ trước tấn công Kernel-mode hay vấn đề về điều kiện tranh chấp khi khởi tạo). Tác giả rất mong nhận được những ý kiến đóng góp chân thành từ các thầy cô và anh chị để đề tài được hoàn thiện hơn trong tương lai.