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

ĐỀ ÁN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

HÀ NỘI 2025

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

ĐỀ ÁN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. NGUYỄN MINH TUẤN

HÀ NỘI 2025

LỜI CAM ĐOAN

Tôi xác nhận rằng Đề án này là kết quả nghiên cứu của riêng tôi, được thực hiện dưới sự hướng dẫn của TS. Nguyễn Minh Tuấn. Các thông tin, kết quả phân tích được thu thập và chúng phản ánh đúng thực tế của Học viện Công nghệ Bưu chính Viễn thông.

Tôi hiểu về các quy định liên quan đến sự trung thực trong học thuật và xin cam đoan rằng nghiên cứu này đã được tôi thực hiện mà không vi phạm bất kỳ yêu cầu nào về sự trung thực trong học thuật.

Hà Nội, ngày tháng năm 2025

Người cam đoan

Nguyễn Minh Đức

LỜI CẢM ƠN

Để hoàn thành luận án này, trước tiên em xin gửi đến các Quý Thầy, Cô giáo Học viện Công nghệ Bưu chính Viễn thông lời cảm ơn chân thành và sâu sắc nhất, đã truyền dạy kiến thức và tạo điều kiện giúp đỡ em trong suốt quá trình theo học tại Học viện.

Đặc biệt, em xin chân thành cảm ơn Thầy TS. Nguyễn Minh Tuấn đã hướng dẫn và giúp đỡ em trong suốt quá trình thực hiện luận án tốt nghiệp. Em rất biết ơn vì sự tận tâm và kiến thức chuyên môn của Thầy đã chỉ dạy em trong suốt quá trình nghiên cứu và hoàn thiện đồ án.

Vì kiến thức bản thân còn nhiều hạn chế, trong quá trình hoàn thiện đồ án tốt nghiệp này, em không tránh khỏi những sai sót. Kính mong nhận được những góp ý từ các Thầy, Cô để hoàn thiện hơn công trình nghiên cứu của mình.

Em xin chúc Quý Thầy, Cô giáo có được thật nhiều sức khỏe, hạnh phúc và gặt hái nhiều thành công trong sự nghiệp trồng người cao cả của mình.

Em xin chân thành cảm ơn!

Hà Nội, ngày tháng năm 20

Sinh viên thực hiện

Nguyễn Minh Đức

MỤC LỤC

LỜI CAM ĐOAN ..... **i**

LỜI CẢM ƠN ..... **ii**

DANH MỤC VIẾT TẮT ..... **vi**

DANH MỤC CÁC BẢNG ..... **viii**

DANH MỤC HÌNH VẼ ..... **ix**

LỜI MỞ ĐẦU **1**

CHƯƠNG 1: TỔNG QUAN VỀ TẤN CÔNG BỘ NHỚ VÀ HỆ THỐNG BẢO VỆ TRONG TRÒ CHƠI ĐIỆN TỬ **5**

1.1. Giới thiệu chương **5**

1.2. Tổng quan về trò chơi điện tử và kiến trúc bộ nhớ **5**

1.2.1. Khái quát về trò chơi điện tử 5

1.2.2. Cấu trúc bộ nhớ trong game 6

1.2.3. Kiến trúc bộ nhớ Windows 8

1.3. Các kỹ thuật tấn công bộ nhớ **9**

1.3.1. Các hình thức tấn công phổ biến 9

1.3.2. Công cụ tấn công bộ nhớ 11

1.3.3. Quy trình thực hiện memory hacking 13

1.4. Tổng quan về các hệ thống chống gian lận hiện nay **15**

1.4.1. Các phương pháp chống gian lận 15

1.4.2. Một số hệ thống phổ biến 17

1.4.3. Hạn chế 19

1.5. Các nghiên cứu liên quan **20**

1.5.1. Phương pháp nhận diện dựa trên chữ ký 20

1.5.2. Phương pháp bảo vệ cấp độ nhân 21

1.5.3. Phương pháp giám sát hành vi tại tầng ứng dụng 23

1.6. So sánh tấn công game và tấn công ứng dụng 24
1.6.1. Mục tiêu tấn công 25
1.6.2. Đặc điểm môi trường 26
1.6.3. Đặc thù nhận diện hành vi bất thường 27
1.7. Kết luận chương 1 28
CHƯƠNG 2: THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG PHÁT HIỆN TẤN CÔNG BỘ NHỚ 30
2.1. Giới thiệu chương 30
***2.2.* Cơ sở lý thuyết và công nghệ nền tảng. 31**
2.2.1. Cấu trúc tập tin thực thi PE 31
2.2.2. Kỹ thuật Hook API 33
2.2.3. Các hàm API Windows nhạy cảm cần giám sát 36
2.3. Phân tích yêu cầu và kiến trúc hệ thống 38
2.3.1. Phân tích yêu cầu bài toán 39
2.3.2. Mô hình kiến trúc tổng thể 41
2.3.3. Các thành phần chức năng chính 42
2.3.4. Kiến trúc hệ thống 44
2.4. Xây dựng các phân hệ chức năng 47
2.4.1. Thiết kế và cài đặt cơ chế tiêm 48
2.4.2. Xây dựng cơ chế Hook và giám sát API 53
2.4.3. Xây dựng thuật toán phát hiện bất thường 55
2.5. Quy trình hoạt động của hệ thống 59
2.5.1. Quy trình khởi tạo 60
2.5.2. Luồng xử lý sự kiện khi phát hiện can thiệp 61
2.6. Kết luận chương 2 63
CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ HỆ THỐNG 65

3.1. Giới thiệu chương.....65

3.2. Môi trường và công cụ thử nghiệm 65

3.2.1. Cấu hình thử nghiệm.....66

3.2.2. Đối tượng thử nghiệm.....67

3.2.3. Bộ công cụ tấn công giả lập.....67

3.3. Các kịch bản thử nghiệm.....68

3.3.1. Tấn công truy cập tiến trình.....68

3.3.2. Tấn công sửa đổi giá trị.....69

3.3.3. Tấn công tiêm mã.....70

3.3.4. Kịch bản kiểm thử tính tương thích.....71

3.3.5. Kịch bản kiểm thử độ bền và tính ổn định hệ thống.....72

3.4. Kết quả thực nghiệm và Phân tích 73

3.4.1. Kết quả về chức năng an ninh.....73

3.4.2. Đánh giá hiệu năng và tính ổn định.....75

3.5. Thảo luận về các hạn chế.....76

3.5.1. Hạn chế đối với tấn công mức nhân.....77

3.5.2. Nguy cơ bị phát hiện ngược.....77

3.5.3. Hạn chế về cơ chế khởi tạo.....78

3.6. Kết luận chương 3 79

KẾT LUẬN.....80

TÀI LIỆU THAM KHẢO.....82

DANH MỤC VIẾT TẮT

Từ viết tắt	Từ đầy đủ	Nghĩa Tiếng Việt
API	Application Programming Interface	Giao diện lập trình ứng dụng
ASLR	Address Space Layout Randomization	Kỹ thuật ngẫu nhiên hóa bố cục không gian địa chỉ
AV	Antivirus	Phần mềm diệt virus
BE	BattlEye	Một hệ thống chống gian lận phổ biến
BSOD	Blue Screen of Death	Màn hình xanh chết chóc (Lỗi hệ thống nghiêm trọng)
CIA	Confidentiality, Integrity, Availability	Ba trụ cột bảo mật: Tính bí mật, Toàn vẹn, Sẵn sàng
CPU	Central Processing Unit	Bộ vi xử lý trung tâm
CS:GO	Counter-Strike: Global Offensive	Tựa game bắn súng chiến thuật góc nhìn thứ nhất
DLL	Dynamic Link Library	Thư viện liên kết động
DMA	Direct Memory Access	Truy cập bộ nhớ trực tiếp (Kỹ thuật phần cứng)
DoS	Denial of Service	Tấn công từ chối dịch vụ
EAC	Easy Anti-Cheat	Một hệ thống chống gian lận phổ biến
FPS	Frames Per Second	Số khung hình trên giây (Chỉ số hiệu năng đồ họa)
HP	Health Points / Hit Points	Chỉ số máu/sinh mệnh của nhân vật trong game

IAT	Import Address Table	Bảng địa chỉ nhập (trong cấu trúc file PE)
IDE	Integrated Development Environment	Môi trường phát triển tích hợp (ví dụ: Visual Studio)
OEP	Original Entry Point	Điểm nhập gốc (Địa chỉ bắt đầu thực thi của chương trình)
OS	Operating System	Hệ điều hành
PE	Portable Executable	Định dạng tập tin thực thi chuẩn trên Windows (.exe, .dll)
PEB	Process Environment Block	Khối môi trường tiến trình (Cấu trúc dữ liệu của Windows)
PID	Process Identifier	Mã định danh tiến trình
PoC	Proof of Concept	Bằng chứng khái niệm (Mã demo kỹ thuật)
QA	Quality Assurance	Đảm bảo chất lượng (Kiểm thử phần mềm)
RAM	Random Access Memory	Bộ nhớ truy cập ngẫu nhiên
RPG	Role-Playing Game	Trò chơi nhập vai
SDK	Software Development Kit	Bộ công cụ phát triển phần mềm
VAC	Valve Anti-Cheat	Hệ thống chống gian lận của Valve
VAS	Virtual Address Space	Không gian địa chỉ ảo
VM	Virtual Machine	Máy ảo

DANH MỤC CÁC BẢNG

Bảng 2.1: Kỹ thuật DLL Injection để nạp module giám sát 50

Bảng 2.2: Logic thao tác con trỏ 52

Bảng 2.3: Đoạn mã cài đặt 54

Bảng 2.4: Đoạn mã phát hiện dựa trên truy cập.....56

Bảng 2.5: Đoạn mã kiểm tra vùng nhớ 58

Bảng 2.6: Đoạn mã kiểm tra tính toàn vẹn vùng nhớ 59

Bảng 3.1: Tổng hợp kết quả thử nghiệm chức năng an ninh 74

Bảng 3.2: So sánh chỉ số FPS trung bình.....75

DANH MỤC HÌNH VẼ

Hình 2.1: Mô hình vòng lặp trò chơi.....6
Hình 2.2: Sơ đồ không gian địa chỉ ảo.....8
Hình 2.3: Công cụ Cheat Engine.....12
Hình 2.4: Quy trình DLL Injection.....13
Hình 2.5: So sánh kiến trúc Anti-cheat (Kernel và User).....18
Hình 2.6 Kiến trúc hoạt động của phương pháp nhận diện dựa trên ký số.....21
Hình 2.7: Kiến trúc hoạt động của phương pháp bảo vệ mức độ nhân.....23
Hình 2.8 Kiến trúc hoạt động của phương pháp giám sát hành vi tại ứng dụng 24
Hình 2.1: Cấu trúc tệp tin PE (Portable Executable).....33
Hình 2.2: Cơ chế Inline Hooking và Trampoline.....36
Hình 2.3: Mô hình kiến trúc tổng thể hệ thống.....42
Hình 2.4: Kiến trúc hệ thống.....47
Hình 3.1: Giao diện game CS:GO và Console Debug.....67
Hình 3.2: Biểu đồ so sánh FPS (Hiệu năng).....76

LỜI MỞ ĐẦU

1. Lý do chọn đề tài

Ngành công nghiệp trò chơi điện tử đang phát triển mạnh mẽ với quy mô toàn cầu, thu hút hàng trăm triệu người chơi và mang lại doanh thu hàng tỷ đô la mỗi năm [2]. Sự phổ biến của các trò chơi trực tuyến, đặc biệt là những trò chơi có tính cạnh tranh cao, đã kéo theo nhiều vấn đề liên quan đến gian lận, trong đó nổi bật là các hình thức can thiệp vào tiến trình trò chơi để thay đổi trạng thái hoạt động theo hướng có lợi cho người chơi gian lận. Một trong những kỹ thuật phổ biến và nguy hiểm nhất hiện nay là **tấn công bộ nhớ (memory hacking)**, cho phép người chơi chỉnh sửa trực tiếp các vùng nhớ của trò chơi để thay đổi giá trị các biến như máu, tọa độ, vũ khí, tốc độ, hoặc hành vi của đối tượng trong game [5], [15].

Các hành vi này không những làm suy giảm trải nghiệm công bằng cho người chơi mà còn ảnh hưởng nghiêm trọng đến uy tín của nhà phát hành và gây tổn thất về mặt kinh tế. Nhiều hệ thống chống gian lận hiện nay vẫn còn phụ thuộc vào các phương pháp phát hiện truyền thống như kiểm tra chữ ký hoặc so sánh checksum, vốn dễ bị qua mặt bởi các kỹ thuật thay đổi mã tại thời điểm chạy, DLL injection hoặc obfuscation.

Trước thực trạng trên, việc nghiên cứu và phát triển một **hệ thống giám sát hành vi truy cập và chỉnh sửa bộ nhớ trong quá trình thực thi trò chơi**, sử dụng các kỹ thuật như **hook API, phân tích truy cập bộ nhớ**, là một hướng đi có giá trị thực tiễn cao. Hệ thống như vậy không chỉ giúp phát hiện sớm các hành vi gian lận mà còn có thể tích hợp vào các công cụ kiểm thử trò chơi, công cụ hỗ trợ QA, hoặc làm nền tảng cho các hệ thống chống gian lận.

Tấn công bộ nhớ là vấn đề nghiêm trọng trong an toàn thông tin, nhưng các cuộc tấn công nhằm vào trò chơi điện tử có những điểm khác biệt so với tấn công phần mềm thông thường. Trong khi tấn công phần mềm thường nhắm vào việc leo thang đặc quyền hay đánh cắp dữ liệu, thì tấn công trong trò chơi điện tử lại tập trung vào việc thay đổi các giá trị như điểm số, tiền tệ, máu hoặc vị trí của nhân vật với mục đích tạo lợi thế so với những người chơi khác. Sự khác biệt này đòi hỏi một

phương pháp tiếp cận chuyên biệt, và đó chính là lý do đề tài này tập trung vào việc xây dựng một hệ thống phát hiện dành riêng cho những trò chơi điện tử.

Đề tài này hướng đến việc nghiên cứu và triển khai một hệ thống nhằm phát hiện hành vi tấn công bộ nhớ trong trò chơi điện tử. Mục tiêu là hỗ trợ quá trình giám sát và kiểm thử bảo mật cho trò chơi, đồng thời góp phần nâng cao tính ổn định và công bằng trong trải nghiệm của người chơi.

2. Tổng quan về vấn đề nghiên cứu.

Tấn công bộ nhớ trong trò chơi điện tử là kỹ thuật khai thác khả năng truy cập vào vùng nhớ của tiến trình trò chơi để thay đổi các giá trị điều khiển trạng thái bên trong hệ thống. Những hành vi này có thể tác động trực tiếp đến kết quả và trải nghiệm chơi game, điển hình như thay đổi giá trị điểm số, dịch chuyển tức thời nhân vật, tự động thực hiện thao tác tấn công, điều chỉnh tốc độ hoặc vị trí, mở khóa vật phẩm bị giới hạn và nhiều thao tác khác. Các hình thức tấn công này thường được thực hiện thông qua các công cụ có khả năng đọc và ghi dữ liệu vào vùng nhớ hoặc can thiệp vào quá trình thực thi của chương trình dưới dạng các đoạn mã tùy chỉnh hoặc trình điều khiển cấp thấp.

Để giảm thiểu rủi ro từ các hành vi gian lận kể trên, nhiều hệ thống chống gian lận đã được xây dựng nhằm kiểm tra, giám sát và ngăn chặn các hành vi bất thường xảy ra trong quá trình vận hành trò chơi. Các hệ thống này chủ yếu dựa vào các kỹ thuật như phát hiện theo mẫu chữ ký, giám sát hành vi truy cập tài nguyên hệ thống, hoặc kiểm tra tính toàn vẹn của mã chương trình. Tuy nhiên, những phương pháp này vẫn gặp nhiều thách thức trong việc phát hiện các hình thức tấn công được che giấu kỹ lưỡng. Các kỹ thuật như làm rối mã, thay đổi hành vi khi chạy, hoặc khai thác cơ chế hoạt động của hệ điều hành khiến cho một số hành vi gian lận không bị phát hiện kịp thời, từ đó làm giảm hiệu quả của các cơ chế bảo vệ hiện có.

Trước thực trạng đó, đề tài tập trung vào việc xây dựng một hệ thống giám sát và phát hiện hành vi tấn công bộ nhớ trong quá trình thực thi trò chơi. Giải pháp được là áp dụng các kỹ thuật phân tích hành vi ở mức hệ thống, như hook API để quan sát các hành vi truy cập và chỉnh sửa bộ nhớ.

Dữ liệu tấn công bộ nhớ là tập hợp các thông tin được thu thập trong thời gian thực. Các thông tin này bao gồm các lệnh ghi bất thường vào vùng nhớ của game, các lệnh đọc bất thường, các lệnh gọi hàm (API calls) đáng ngờ từ các tiến trình bên ngoài, và những thay đổi giá trị trong bộ nhớ game (như HP, tốc độ) không tuân theo logic.

Hệ thống phát hiện tấn công bộ nhớ sẽ được triển khai như một chương trình giám sát độc lập, hoạt động song song với tiến trình trò chơi trên hệ điều hành Windows. Hệ thống này không tích hợp trực tiếp vào mã nguồn của game mà sử dụng các kỹ thuật hook API để theo dõi các hành vi đáng ngờ từ bên ngoài. Cách tiếp cận này giúp đảm bảo tính linh hoạt, cho phép hệ thống áp dụng cho nhiều trò chơi khác nhau

Việc triển khai hệ thống theo hướng này giúp tăng tính chủ động trong việc phát hiện sớm các hành vi gian lận mà không phụ thuộc vào mẫu cố định hay thuật toán phức tạp, đồng thời vẫn đảm bảo phù hợp với định hướng ứng dụng của đề tài.

3. Mục đích nghiên cứu

Mục tiêu của đề tài là thiết kế và xây dựng một hệ thống có khả năng giám sát và phát hiện hành vi tấn công bộ nhớ trong quá trình thực thi trò chơi điện tử. Hệ thống nhằm hỗ trợ các nhà phát triển trong việc kiểm thử, bảo vệ và nâng cao tính bảo mật cho sản phẩm trò chơi, đồng thời cung cấp một công cụ kỹ thuật có thể ứng dụng thực tiễn trong môi trường phát triển game hoặc nghiên cứu về an toàn phần mềm. Cụ thể, đề tài hướng đến các mục tiêu sau:

- **Nghiên cứu kỹ thuật tấn công bộ nhớ trong game:** Tìm hiểu, phân tích và phân loại các phương pháp can thiệp vào vùng nhớ của trò chơi, bao gồm các hình thức phổ biến như sửa giá trị biến, ghi đè bộ nhớ, chèn mã thực thi, hoặc điều khiển hành vi đối tượng trong trò chơi thông qua truy cập bộ nhớ trái phép. Trên cơ sở đó, xác định các dấu hiệu đặc trưng để phục vụ cho việc phát hiện.
- **Xây dựng hệ thống giám sát bộ nhớ:** Phát triển một hệ thống có khả năng theo dõi hành vi truy cập và ghi dữ liệu vào vùng nhớ nhạy cảm của tiến trình game trong thời gian thực. Hệ thống sẽ sử dụng các kỹ thuật như hook API để ghi nhận các hành vi nghi vấn.

- **Thiết kế cơ chế phát hiện hành vi bất thường:** Đề xuất và triển khai một cơ chế phát hiện các hành vi không hợp lệ dựa trên mẫu truy cập bộ nhớ, thứ tự các lời gọi API, hoặc thay đổi dữ liệu tại các vùng nhớ quan trọng. Cơ chế này phải đảm bảo tính hiệu quả trong việc phát hiện, đồng thời hạn chế tối đa các cảnh báo sai.
- **Đánh giá hiệu quả hệ thống trong môi trường thử nghiệm:** Tiến hành kiểm thử hệ thống trên một hoặc một số trò chơi mẫu, mô phỏng các tình huống tấn công bộ nhớ phổ biến để đánh giá khả năng phát hiện, độ trễ xử lý và tính ổn định của hệ thống. Các kịch bản này bao gồm: thay đổi giá trị đơn giản (giả lập việc sử dụng công cụ can thiệp bộ nhớ để chỉnh sửa giá trị của biến như máu hoặc tốc độ), tấn công bằng cách tiêm mã độc (DLL injection), và tấn công bằng cách ghi đè bộ nhớ (giả lập việc lợi dụng các lỗ hổng để ghi đè dữ liệu vào vùng nhớ quan trọng). Từ đó, đưa ra các nhận định về khả năng ứng dụng và đề xuất cải tiến trong tương lai.

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

- **Phương pháp thu thập tài liệu:** Nghiên cứu và tìm hiểu các tài liệu liên quan đã được công bố trước đó nhằm thu thập thông tin và kiến thức phục vụ cho đề tài. Các nguồn tài liệu bao gồm sách chuyên ngành, báo khoa học, tạp chí chuyên đề, tài liệu kỹ thuật và các công trình nghiên cứu có liên quan.
- **Phương pháp phân tích và tổng hợp:** Thực hiện so sánh, đánh giá các giải pháp, mô hình và ứng dụng đã thu thập được từ quá trình nghiên cứu tài liệu. Qua đó rút ra các đặc điểm, ưu điểm và hạn chế để làm cơ sở đề xuất hướng tiếp cận phù hợp cho hệ thống được xây dựng trong đề tài.
- **Phương pháp thực nghiệm:** Triển khai thử nghiệm hệ thống trên phần mềm nhằm kiểm tra tính khả thi, hiệu quả và khả năng phát hiện hành vi tấn công bộ nhớ trong môi trường giả lập hoặc thực tế. Kết quả thực nghiệm sẽ là cơ sở để đánh giá mức độ đáp ứng của hệ thống so với mục tiêu ban đầu.

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 1 trình bày cơ sở lý thuyết và bối cảnh tổng quan về vấn đề tấn công bộ nhớ trong trò chơi điện tử – kỹ thuật gian lận phổ biến nhất hiện nay. Sự bùng nổ của ngành công nghiệp game với hơn 3 tỷ người chơi và doanh thu hàng trăm tỷ USD (Newzoo, 2023) đã biến trò chơi điện tử thành mục tiêu hấp dẫn cho các hành vi can thiệp trái phép. Do đó, việc nắm vững kiến trúc bộ nhớ và cơ chế hoạt động của các kỹ thuật tấn công (như sửa đổi giá trị HP/tọa độ, tiêm mã độc) là tiền đề bắt buộc để xây dựng các giải pháp phòng thủ hiệu quả.

Bên cạnh việc phân tích kỹ thuật, chương này cũng khảo sát các hệ thống chống gian lận tiêu chuẩn công nghiệp như VAC, BattlEye và Easy Anti-Cheat. Mặc dù đạt hiệu quả cao, các hệ thống này chủ yếu hoạt động ở tầng nhân (kernel-mode) với độ can thiệp sâu, gây khó khăn cho việc nghiên cứu và triển khai độc lập [12]. Thực tế cho thấy, nhiều kỹ thuật gian lận vẫn có thể được phát hiện hiệu quả ngay tại tầng người dùng (user-mode) nếu áp dụng đúng cơ chế giám sát API và truy cập bộ nhớ.

Tóm lại, nội dung Chương 1 cung cấp nền tảng lý thuyết vững chắc bao gồm: tổng quan về game, kiến trúc bộ nhớ Windows, phân loại các hình thức tấn công và công cụ hỗ trợ. Những kiến thức này không chỉ định vị phạm vi nghiên cứu mà còn tạo cơ sở logic cho việc thiết kế và triển khai hệ thống giám sát trong các chương tiếp theo.

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

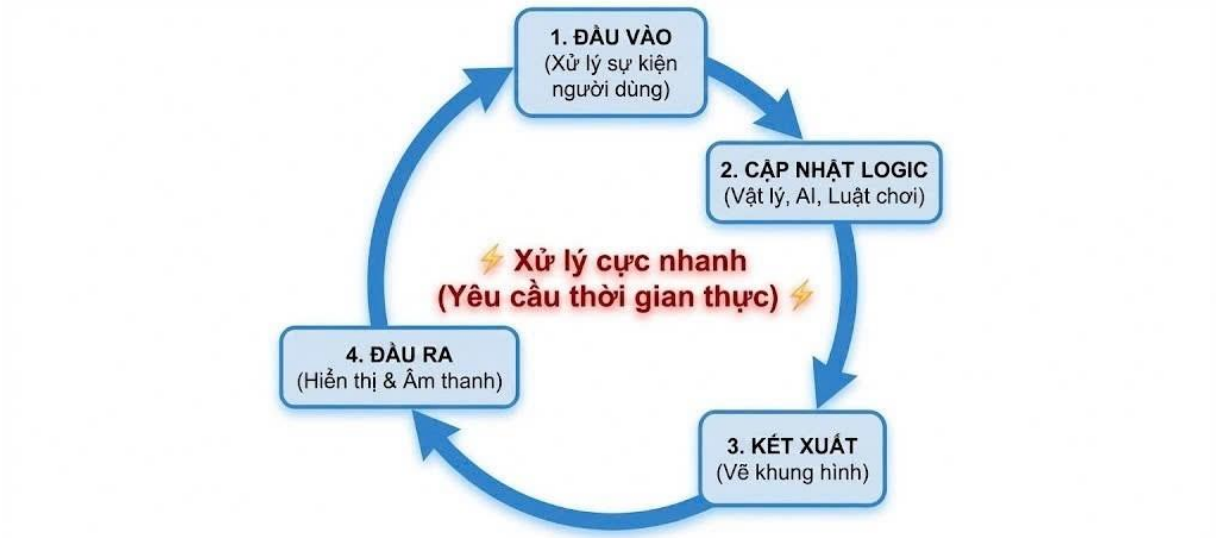
1.2.1. Khái quát về trò chơi điện tử

Trò chơi điện tử là một dạng phần mềm tương tác cho phép người chơi điều khiển các đối tượng trong môi trường ảo mô phỏng. Trải qua quá trình phát triển cùng công nghệ phần cứng và mạng Internet, trò chơi điện tử hiện nay đã trở thành những hệ thống phần mềm có quy mô lớn và độ phức tạp cao, tích hợp đa dạng các thành phần từ đồ họa, âm thanh đến trí tuệ nhân tạo và xử lý mạng.

Về mặt kỹ thuật, đặc điểm cốt lõi phân biệt trò chơi với các ứng dụng thông thường nằm ở cơ chế vận hành dựa trên "vòng lặp trò chơi" (Game Loop). Khác với

mô hình hướng sự kiện thụ động (event-driven), game loop liên tục cập nhật trạng thái của thế giới ảo—bao gồm vị trí nhân vật, chỉ số vật lý, thông số môi trường—với tần suất rất cao (thường là 60 lần/giây hoặc hơn) để đảm bảo tính thời gian thực (real-time).

Chính yêu cầu khắt khe về tốc độ xử lý này khiến cho bộ nhớ RAM trở thành nơi lưu trữ và biến đổi dữ liệu quan trọng nhất trong quá trình vận hành. Đồng thời, tính thời gian thực cũng làm cho các trò chơi, đặc biệt là game trực tuyến nhiều người chơi, trở nên cực kỳ nhạy cảm với độ trễ (latency). Bất kỳ sự can thiệp nào vào bộ nhớ không chỉ ảnh hưởng đến tính toàn vẹn dữ liệu mà còn có nguy cơ phá vỡ sự ổn định của vòng lặp xử lý, đặt ra thách thức lớn cho việc xây dựng các giải pháp bảo mật vừa hiệu quả vừa không gây ảnh hưởng đến hiệu năng trải nghiệm.



Hình 2.1: Mô hình vòng lặp trò chơi

1.2.2. Cấu trúc bộ nhớ trong game

Bộ nhớ của trò chơi điện tử là nơi lưu trữ toàn bộ trạng thái hoạt động của game trong quá trình thực thi, bao gồm dữ liệu về nhân vật, vật phẩm, bản đồ, quy tắc xử lý và nhiều thành phần khác. Khác với các ứng dụng thông thường, game yêu cầu xử lý liên tục theo thời gian thực, do đó các biến, đối tượng và cấu trúc dữ liệu trong bộ nhớ luôn thay đổi với tốc độ cao. Điều này khiến bộ nhớ game trở thành mục

tiêu hấp dẫn cho các kỹ thuật gian lận, bởi chỉ cần thay đổi một phần nhỏ dữ liệu cũng có thể ảnh hưởng trực tiếp đến trạng thái của trò chơi.

Về cơ bản, cấu trúc bộ nhớ của game được xây dựng dựa trên mô hình bộ nhớ của hệ điều hành, trong đó mỗi tiến trình của game có một không gian địa chỉ ảo riêng. Không gian này được chia thành nhiều vùng chức năng như **text/code segment**, **data segment**, **heap**, **stack**, và các vùng bộ nhớ được cấp phát động cho đối tượng game. Các biến liên quan đến gameplay – chẳng hạn như điểm số, máu, vị trí, tốc độ, trạng thái đạn, hoặc các thông số của các đối tượng – thường được lưu trong vùng nhớ heap hoặc các vùng cấp phát động do engine quản lý. Những giá trị này thay đổi liên tục theo hành vi của người chơi và logic của trò chơi, vì vậy luôn được ghi nhận trực tiếp trong RAM.

Trong game, các đối tượng và thành phần thường được quản lý thông qua cơ chế cấp phát bộ nhớ động bằng con trỏ. Game engine như Unity, Unreal Engine hay Source Engine lưu trữ các đối tượng theo cấu trúc hướng đối tượng, với các con trỏ liên kết đến nhiều vùng dữ liệu khác nhau. Vì lý do này, rất nhiều công cụ gian lận dựa vào việc quét bộ nhớ để tìm ra địa chỉ của các biến quan trọng dựa trên kiểu dữ liệu hoặc giá trị đặc trưng. Khi địa chỉ được xác định, kẻ tấn công có thể ghi đè giá trị mới để thay đổi trực tiếp trạng thái của game.

Ngoài ra, một số trò chơi sử dụng bảng địa chỉ động hoặc cơ chế sinh địa chỉ theo phiên chơi nhằm gây khó khăn cho việc quét bộ nhớ. Tuy nhiên, với các kỹ thuật như pointer scanning, module base address hoặc signature scanning, kẻ gian vẫn có thể lần theo cấu trúc bộ nhớ để tìm ra vị trí của biến cần thay đổi. Điều này cho thấy cấu trúc bộ nhớ của game, dù phức tạp, vẫn dễ bị khai thác nếu không có biện pháp giám sát và bảo vệ phù hợp.

Tóm lại, cấu trúc bộ nhớ trong game là yếu tố trung tâm của cơ chế vận hành trò chơi, đồng thời là điểm yếu mà các công cụ gian lận thường nhắm đến. Việc hiểu rõ cách dữ liệu game được tổ chức và quản lý trong bộ nhớ là cơ sở quan trọng để phân tích các kỹ thuật tấn công bộ nhớ cũng như phát triển các giải pháp giám sát và phát hiện hành vi bất thường trong những chương tiếp theo.

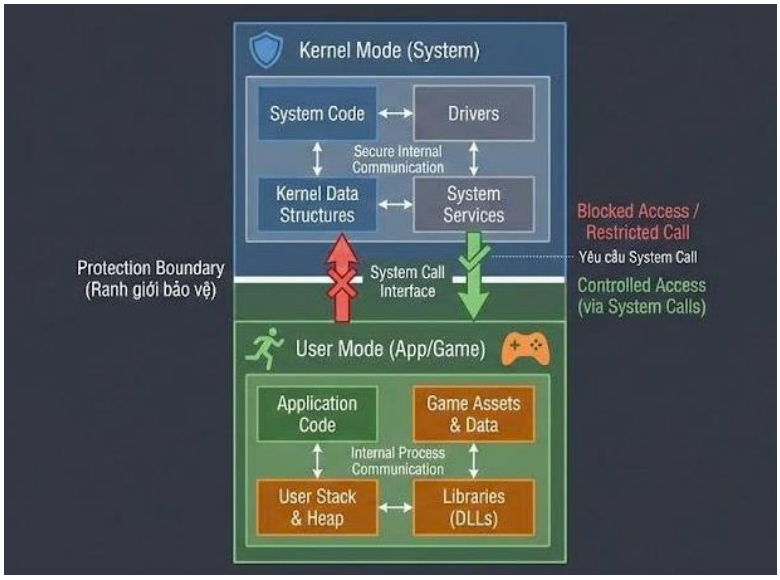
1.2.3. Kiến trúc bộ nhớ Windows

Mặc dù Windows áp dụng cơ chế cách ly không gian địa chỉ (Virtual Address Space Isolation) để bảo vệ các tiến trình độc lập, hệ điều hành này vẫn cung cấp một bộ giao diện lập trình (API) mạnh mẽ cho phép tương tác liên tiến trình (Inter-process Communication) nhằm phục vụ mục đích gỡ lỗi và quản trị hệ thống. Chính các cơ chế hợp pháp này đã trở thành vector tấn công chính mà các công cụ gian lận khai thác.

Cụ thể, Windows cho phép một tiến trình có thể thao tác trực tiếp lên bộ nhớ của tiến trình khác thông qua việc nắm giữ một tiến trình điều khiển (Handle).

- **Quyền truy cập:** Hàm OpenProcess cho phép kẻ tấn công lấy được Handle của tiến trình game với các quyền hạn nguy hiểm như PROCESS_VM_WRITE (ghi bộ nhớ) hay PROCESS_VM_OPERATION.
- **Thao tác dữ liệu:** Sau khi có Handle, các hàm ReadProcessMemory và WriteProcessMemory cho phép đọc/ghi giá trị tại bất kỳ địa chỉ nào trong game, tạo tiền đề cho các hành vi hack máu, đạn hay tọa độ.
- **Can thiệp mã lệnh:** Các hàm VirtualAllocEx và CreateRemoteThread cho phép cấp phát bộ nhớ và tạo luồng thực thi từ xa, là cơ sở của kỹ thuật tiêm mã độc.

Như vậy, bản chất của tấn công bộ nhớ trên Windows không phải là khai thác lỗ hổng của hệ điều hành, mà là sự **lạm dụng các tính năng thiết kế sẵn**. Do đó, giải pháp phòng thủ hiệu quả không phải là vá lỗi Windows, mà là giám sát và kiểm soát chặt chẽ việc gọi các API nhạy cảm này.



Hình 2.2: Sơ đồ không gian địa chỉ ảo

1.3. Các kỹ thuật tấn công bộ nhớ

Tấn công bộ nhớ trong trò chơi điện tử là nhóm kỹ thuật cho phép kẻ tấn công can thiệp trực tiếp vào trạng thái của tiến trình game bằng cách đọc, sửa đổi hoặc chèn mã vào bộ nhớ đang được sử dụng. Đây là hình thức gian lận phổ biến nhất hiện nay do triển khai đơn giản, hiệu quả tức thời và khó bị phát hiện nếu game không có cơ chế giám sát phù hợp. Các hình thức tấn công phổ biến có thể chia thành ba nhóm chính: sửa đổi giá trị, chèn mã và vá mã tại thời điểm chạy.

1.3.1. Các hình thức tấn công phổ biến

a) Kỹ thuật hiệu chỉnh biến số trò chơi

Sửa đổi giá trị là hình thức tấn công cơ bản nhất, trong đó kẻ tấn công tìm và ghi đè giá trị của các biến quan trọng trong bộ nhớ game, chẳng hạn như máu, đạn, tốc độ di chuyển, tọa độ hoặc điểm số. Phương pháp phổ biến nhất là sử dụng kỹ thuật quét bộ nhớ để xác định địa chỉ chứa dữ liệu cần thay đổi. Khi tìm được địa chỉ mục tiêu, kẻ tấn công sử dụng API như **WriteProcessMemory** để ghi giá trị mới vào RAM.

Một ví dụ điển hình trong các trò chơi bắn súng, việc thay đổi giá trị đạn từ 30 lên 999 giúp người chơi có lượng đạn vô hạn; hoặc trong trò chơi nhập vai, chỉnh sửa giá trị HP giúp nhân vật trở nên bất tử. Hình thức tấn công này thường được thực hiện bằng các công cụ như Cheat Engine hoặc các công cụ chỉnh sửa khác. Mặc dù đơn giản, kỹ thuật này gây ảnh hưởng trực tiếp và nghiêm trọng đến tính công bằng của trò chơi.

b) Các kỹ thuật Code Injection và DLL Injection

Chèn mã là hình thức tấn công nâng cao hơn, cho phép kẻ tấn công đưa đoạn mã của họ vào tiến trình game nhằm thực thi các chức năng không có trong thiết kế ban đầu. Phổ biến nhất là **DLL injection**, trong đó một file thư viện (.dll) được nạp vào tiến trình game thông qua các hàm như **CreateRemoteThread**, **LoadLibraryA/W** hoặc **SetWindowsHookEx**.

Khi một DLL độc hại được chèn thành công, mã trong DLL đó sẽ chạy với cùng quyền truy cập như tiến trình game, cho phép kẻ tấn công tạo thêm tính năng như aim-bot, wall-hack, auto-shoot, hoặc can thiệp vào logic xử lý va chạm và tương

tác của nhân vật. Code injection là nền tảng cho nhiều cheat engine phức tạp, vì nó cho phép thao tác trực tiếp với các cấu trúc nội bộ của game, đọc dữ liệu trong thời gian thực và thực thi mã ngoài tầm kiểm soát của lập trình viên game.

c) Kỹ thuật can thiệp và sửa đổi mã lệnh động

Runtime patching là kỹ thuật thay đổi nội dung mã máy ngay trong bộ nhớ của tiến trình khi game đang chạy. Kẻ tấn công có thể thay đổi các lệnh máy để vô hiệu hóa một cơ chế trong game, bỏ qua một điều kiện kiểm tra hoặc chuyển hướng luồng thực thi. Một dạng phổ biến của kỹ thuật này là **inline hooking**, trong đó kẻ tấn công ghi đè một số byte đầu tiên của hàm mục tiêu để chuyển hướng luồng thực thi sang đoạn mã khác do họ viết.

Ví dụ, nếu game có một hàm kiểm tra tính hợp lệ của tọa độ nhân vật, kẻ tấn công có thể patch mã để luôn trả về giá trị "hợp lệ", từ đó cho phép dịch chuyển tức thời mà không bị hệ thống phát hiện. Runtime patching nguy hiểm vì nó thay đổi trực tiếp hành vi của chương trình, gây khó khăn cho quá trình phát hiện khi game không kiểm tra tính toàn vẹn của mã đang thực thi.

d) Các hình thức tấn công khác liên quan

Ngoài ba nhóm chính nêu trên, còn nhiều kỹ thuật tấn công khác được sử dụng trong thực tế, chẳng hạn như:

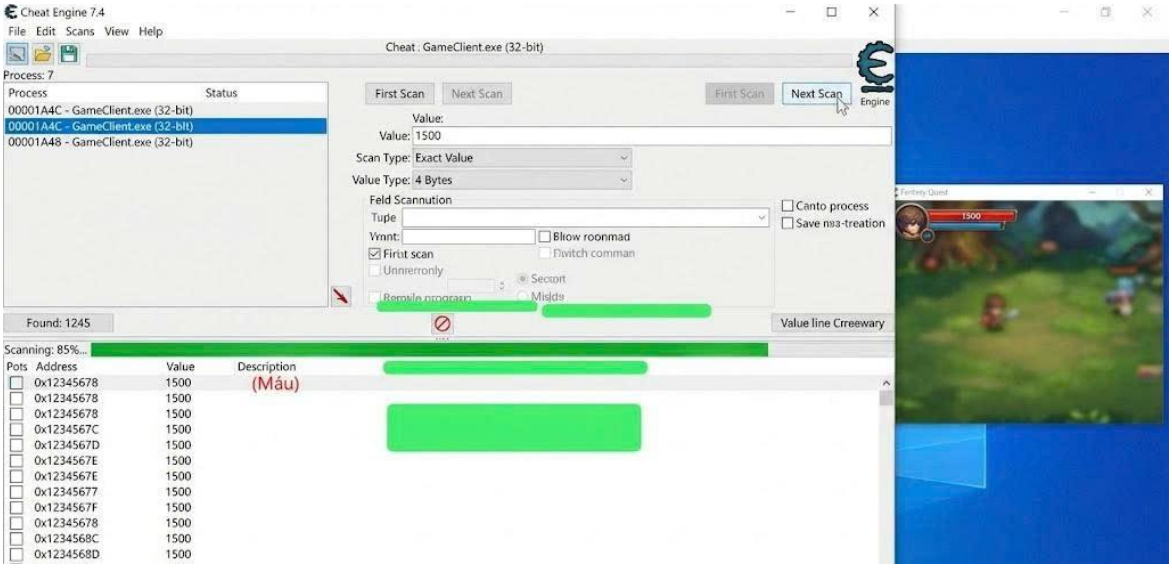
- **Pointer modification**: thay đổi địa chỉ con trỏ nhằm truy xuất dữ liệu trái phép.
- **Memory freezing**: giữ nguyên giá trị của biến bằng cách liên tục ghi đè theo chu kỳ nhanh.
- **Thread injection**: tạo thread mới trong tiến trình game để thực thi mã tùy ý.
- **External overlay và memory reading**: đọc dữ liệu game từ bên ngoài để hiển thị thông tin không được phép.

Những kỹ thuật này thường được kết hợp để tạo thành các công cụ gian lận hoàn chỉnh và tinh vi hơn.

1.3.2. Công cụ tấn công bộ nhớ

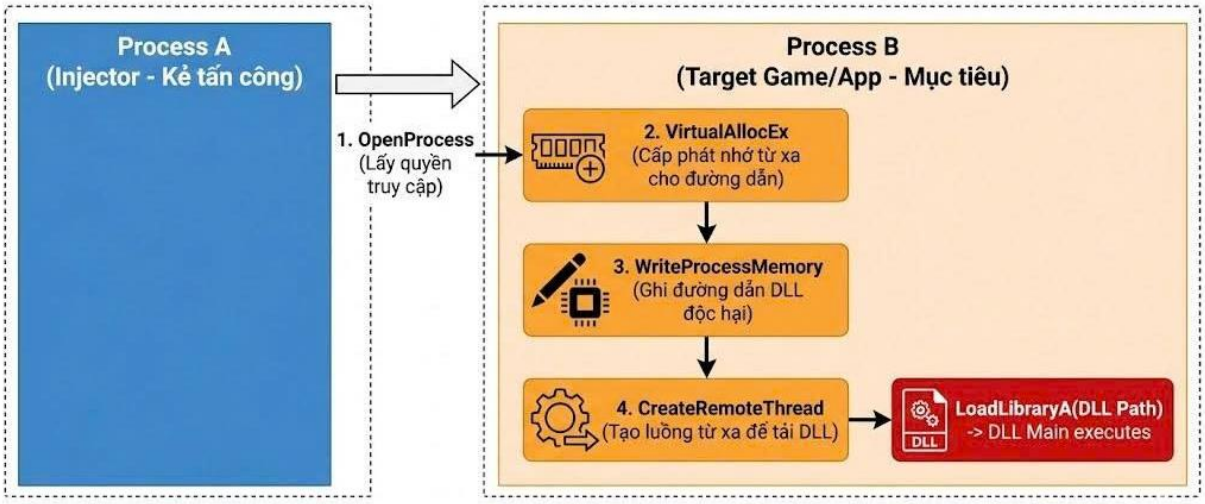
Trong môi trường trò chơi điện tử, các công cụ tấn công bộ nhớ đóng vai trò là phương tiện trọng yếu giúp người sử dụng hoặc kẻ tấn công can thiệp vào tiến trình của game mà không cần tác động tới tệp thực thi gốc. Chúng tận dụng cơ chế quản lý bộ nhớ của Windows cùng với các API hợp pháp của hệ điều hành để truy xuất, phân tích hoặc chỉnh sửa dữ liệu trong RAM. Các công cụ này khác nhau về mục đích, mức độ tinh vi và phạm vi ảnh hưởng, nhưng đều có chung đặc điểm là hoạt động hoàn toàn ở tầng user-mode, phù hợp với hướng nghiên cứu của đề tài. Việc phân tích chi tiết từng nhóm công cụ không chỉ giúp nhận diện cách chúng khai thác bộ nhớ mà còn là cơ sở để xây dựng hệ thống phát hiện hành vi bất thường một cách chính xác.

- **Cheat Engine** là công cụ chỉnh sửa bộ nhớ mạnh mẽ và phổ biến nhất, thường xuyên được sử dụng bởi cả người chơi phổ thông lẫn những đối tượng am hiểu kỹ thuật. Công cụ này cung cấp khả năng quét bộ nhớ theo nhiều phương pháp khác nhau, bao gồm quét giá trị chính xác, quét giá trị không xác định, quét giá trị tăng/giảm theo thời gian hoặc thực hiện fuzzy scan để tìm biến ẩn. Bên cạnh đó, Cheat Engine hỗ trợ chức năng pointer scan nhằm xác định chuỗi con trỏ liên kết với một biến nhất định — điều rất quan trọng vì trong nhiều trò chơi, địa chỉ dữ liệu thay đổi sau mỗi lần khởi động do sử dụng cơ chế cấp phát động. Công cụ cũng tích hợp debugger giúp theo dõi hành vi đọc/ghi biến, đặt breakpoint, theo dõi call stack và hiểu rõ logic xử lý của trò chơi. Đặc biệt, tính năng Auto Assembler cho phép chèn mã trực tiếp vào bộ nhớ game thông qua injection hoặc inline patching, từ đó thực thi những lệnh máy thay thế cho logic ban đầu. Những chức năng này kết hợp lại tạo ra một công cụ có khả năng phân tích sâu và thay đổi hầu như mọi khía cạnh của trò chơi ở mức tiến trình.



Hình 2.3: Công cụ Cheat Engine

- **DLL Injector** là nhóm công cụ chuyên chèn thư viện động vào tiến trình của game để thực thi mã tùy ý bên trong không gian bộ nhớ. Các injector như Xenos, GH Injector hoặc Extreme Injector thường áp dụng kỹ thuật CreateRemoteThread kết hợp với LoadLibrary để yêu cầu hệ điều hành nạp một DLL vào tiến trình. Những phương pháp nâng cao hơn như manual mapping thậm chí cho phép nạp DLL mà không cần sử dụng API nạp thư viện của Windows, qua đó giảm khả năng bị hệ thống chống gian lận phát hiện. Khi DLL được nạp thành công, người tấn công có thể hook vào các hàm nội bộ, thay đổi luồng thực thi, đọc và ghi dữ liệu trực tiếp hoặc chèn logic mới nhằm thực hiện những hành vi gian lận. Do DLL injection cho phép mã chạy trong cùng không gian địa chỉ với game, mức độ can thiệp của nó sâu hơn rất nhiều so với các công cụ chỉnh sửa giá trị đơn thuần và thường được xem là một trong những kỹ thuật nguy hiểm nhất trong tấn công user-mode.



Hình 2.4: Quy trình DLL Injection

- **Debugger và Disassembler** như x64dbg, OllyDbg, IDA Pro và Ghidra đóng vai trò quan trọng trong việc phân tích cấu trúc nội bộ của trò chơi và xây dựng nền tảng kỹ thuật cho các công cụ gian lận khác. Những công cụ này cho phép người dùng xem mã máy của game, xác định vị trí của các hàm quan trọng, quan sát quá trình xử lý dữ liệu và đặt breakpoint lên các điểm truy cập bộ nhớ. Với debugger, người phân tích có thể kiểm tra xem giá trị nào được thay đổi bởi hàm nào, ở thời điểm nào, cũng như theo dõi toàn bộ luồng thực thi của chương trình thông qua call stack và register. Trong khi đó, các công cụ disassembler như IDA Pro hoặc Ghidra hỗ trợ dịch ngược mã máy sang dạng biểu diễn mức cao hơn, giúp việc tìm kiếm cấu trúc dữ liệu, hàm logic gameplay hoặc hàm kiểm tra chống gian lận trở nên dễ dàng hơn. Những kiến thức và dữ liệu thu thập được từ debugger và disassembler thường được sử dụng để tạo ra mã chèn, patch inline hoặc xây dựng trainer chuyên biệt. Do đó, dù không trực tiếp sửa đổi giá trị bộ nhớ như Cheat Engine, nhóm công cụ này lại đóng vai trò nền tảng trong việc hình thành các tấn công bộ nhớ tinh vi và chuyên nghiệp.

1.3.3. Quy trình thực hiện memory hacking

Quy trình tấn công bộ nhớ đối với trò chơi điện tử thường tuân theo một chuỗi các bước có tính hệ thống, nhằm xác định, truy cập và can thiệp vào dữ liệu trong bộ nhớ RAM của tiến trình game. Mặc dù cách thức thực hiện có thể thay đổi tùy theo công cụ hoặc mức độ tinh vi của kẻ tấn công, đa số các cuộc tấn công đều tuân thủ

một cấu trúc chung gồm ba giai đoạn chính: phân tích giá trị mục tiêu, truy xuất bộ nhớ tiến trình và can thiệp vào dữ liệu hoặc luồng thực thi. Việc nắm bắt quy trình này giúp hiểu rõ cơ chế hoạt động của từng hình thức gian lận, từ đó hỗ trợ xây dựng mô hình phát hiện hiệu quả ở tầng user-mode.

Giai đoạn đầu tiên là **xác định giá trị hoặc cấu trúc dữ liệu mục tiêu trong bộ nhớ game**. Kẻ tấn công cần phân biệt đâu là những giá trị ảnh hưởng trực tiếp đến trạng thái gameplay, chẳng hạn như số lượng máu, đạn, tốc độ di chuyển, vị trí nhân vật hoặc thông tin liên quan đến đối thủ. Thông thường, người tấn công sử dụng các công cụ như Cheat Engine để quét toàn bộ không gian địa chỉ ảo của tiến trình game và tìm các giá trị phù hợp với điều kiện quan sát được từ giao diện trò chơi. Trong trường hợp giá trị không rõ ràng hoặc có cơ chế mã hóa, họ có thể áp dụng kỹ thuật quét giá trị không xác định, quét giá trị tăng hoặc giảm theo thời gian, hoặc phân tích biến thông qua breakpoint. Giai đoạn này tồn tại lặp lại nhiều lần để người tấn công tinh lọc được đúng vị trí biến trong bộ nhớ.

Giai đoạn tiếp theo là **truy xuất vào bộ nhớ của tiến trình game bằng các API hợp pháp của Windows**. Khi địa chỉ hoặc vùng dữ liệu mục tiêu đã được xác định, kẻ tấn công sử dụng các hàm như OpenProcess để mở đến tiến trình game, sau đó dùng ReadProcessMemory để đọc dữ liệu từ các địa chỉ tương ứng. Nếu muốn can thiệp hoặc sửa đổi, họ sử dụng WriteProcessMemory để ghi đè giá trị hoặc thay đổi dữ liệu trong RAM. Trong các tấn công nâng cao, kẻ tấn công có thể thay đổi quyền truy cập của vùng nhớ thông qua VirtualProtectEx hoặc cấp phát thêm bộ nhớ mới bằng VirtualAllocEx nhằm tạo không gian cho mã chèn. Đây là bước giúp kẻ tấn công chuyển đổi từ việc chỉ quan sát dữ liệu sang khả năng can thiệp trực tiếp vào trạng thái nội bộ của game.

Giai đoạn cuối cùng là **thực hiện chỉnh sửa dữ liệu hoặc thay đổi luồng thực thi của trò chơi**, tùy thuộc vào mục tiêu của kẻ tấn công. Đối với các tấn công đơn giản, họ chỉ cần ghi đè một giá trị — ví dụ như đặt lượng máu về mức tối đa hoặc khóa giá trị không giảm — và để công cụ tự động ghi lại theo chu kỳ để duy trì hiệu lực. Các dạng tấn công phức tạp hơn liên quan đến patching mã lệnh, chẳng hạn như

thay đổi một phần mã trong bộ nhớ để bỏ qua điều kiện kiểm tra hoặc đảo ngược logic của hàm xử lý sự kiện. Trong trường hợp sử dụng DLL injection, mã độc hại được chèn vào tiến trình game có thể hook vào các hàm nội bộ, chặn hoặc thay thế dữ liệu trước khi game xử lý, từ đó tạo ra các chức năng gian lận.

Tổng thể, quy trình tấn công bộ nhớ là một chuỗi hành động có tổ chức, được hỗ trợ bởi các công cụ mạnh mẽ và khai thác trực tiếp các API của hệ điều hành. Dù hoạt động hoàn toàn ở tầng user-mode, quy trình này vẫn đủ khả năng tạo ra những hành vi gian lận nghiêm trọng nếu không được giám sát. Vì vậy, việc phân tích sâu các bước của quy trình này đóng vai trò quan trọng trong việc xây dựng hệ thống phát hiện tấn công bộ nhớ mà luận văn hướng tới.

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

Trong bối cảnh các trò chơi điện tử ngày càng phát triển về quy mô, tính cạnh tranh và giá trị kinh tế, vấn đề gian lận trong game trở thành thách thức lớn đối với cả nhà phát triển lẫn cộng đồng người chơi. Để đối phó với các hành vi can thiệp trái phép, đặc biệt là các kỹ thuật chỉnh sửa bộ nhớ ở tầng user-mode, nhiều hệ thống chống gian lận đã được xây dựng và triển khai nhằm giám sát, phát hiện và ngăn chặn hoạt động gian lận trong quá trình vận hành trò chơi. Các hệ thống này hoạt động theo nhiều cơ chế khác nhau, từ kiểm tra tính toàn vẹn của tiến trình, phân tích hành vi API, theo dõi bất thường trong bộ nhớ cho đến triển khai các mô-đun bảo vệ ở tầng kernel. Việc tìm hiểu tổng quan về các hệ thống chống gian lận hiện nay không chỉ giúp đánh giá mức độ hiệu quả và hạn chế của từng phương pháp mà còn là cơ sở quan trọng để xác định vấn đề nghiên cứu, từ đó định hướng cho việc xây dựng hệ thống phát hiện tấn công bộ nhớ mà luận văn đề xuất.

1.4.1. Các phương pháp chống gian lận

Các hệ thống chống gian lận trong trò chơi điện tử hiện nay vận hành dựa trên nhiều phương pháp khác nhau, tùy thuộc vào mô hình trò chơi, mức độ quan trọng của dữ liệu gameplay và khả năng can thiệp của người chơi vào quá trình thực thi. Nhìn chung, các phương pháp chống gian lận có thể được phân thành những nhóm chính như kiểm tra tính toàn vẹn, nhận diện dựa trên chữ ký, giám sát hành vi, kiểm

tra dữ liệu từ phía máy chủ và sử dụng mô-đun bảo vệ ở tầng kernel. Mỗi phương pháp có mục tiêu, ưu điểm và phạm vi ứng dụng khác nhau, nhưng đều hướng tới mục tiêu chung là ngăn chặn sự can thiệp trái phép vào tiến trình game và đảm bảo tính công bằng cho người chơi.

Phương pháp đầu tiên thường được sử dụng là **kiểm tra tính toàn vẹn của tiến trình**. Hệ thống chống gian lận so sánh mã lệnh, cấu trúc bộ nhớ hoặc chữ ký của các module đang được nạp vào tiến trình game với phiên bản gốc. Nếu phát hiện sự thay đổi bất thường như patching mã lệnh, thay thế DLL hoặc chèn module lạ, hệ thống sẽ cảnh báo hoặc chặn ngay lập tức người chơi khỏi phiên chơi. Đây là phương pháp hiệu quả trong việc phát hiện những hình thức tấn công đơn giản, đặc biệt là các công cụ sửa đổi code hoặc injection cơ bản, tuy nhiên khả năng đối phó với các kỹ thuật tinh vi như manual mapping hoặc code obfuscation còn hạn chế.

Phương pháp tiếp theo là **nhận diện dựa trên chữ ký**, tương tự mô hình hoạt động của phần mềm chống virus. Hệ thống so sánh dữ liệu trong bộ nhớ, mã lệnh hoặc hành vi cụ thể với một cơ sở dữ liệu chữ ký đã được xác định trước. Khi phát hiện sự trùng khớp với mẫu gian lận đã biết, hệ thống có thể xử lý ngay lập tức. Mặc dù dễ triển khai và hiệu quả đối với các dạng cheat phổ biến, phương pháp này khó chống lại các công cụ gian lận được tùy biến liên tục, vốn không để lại dấu hiệu cố định hoặc tự thay đổi mã mỗi khi chạy để né tránh việc quét chữ ký.

Nhóm phương pháp quan trọng tiếp theo là **giám sát hành vi của tiến trình**, trong đó hệ thống tập trung vào việc theo dõi các hành vi bất thường thay vì chỉ dựa trên dấu hiệu tĩnh. Ví dụ, hệ thống có thể giám sát tần suất gọi các API như ReadProcessMemory, WriteProcessMemory, VirtualProtectEx hoặc CreateRemoteThread. Những hành vi can thiệp thường xuyên vào bộ nhớ của tiến trình hoặc chèn luồng bên ngoài thường được xem là dấu hiệu của tấn công bộ nhớ. Một số hệ thống còn phân tích hành vi gameplay, chẳng hạn như tốc độ di chuyển bất thường, độ chính xác vượt ngưỡng hoặc thao tác không thể xảy ra theo logic vật lý. Đây là phương pháp cho phép phát hiện các kỹ thuật gian lận mới chưa có chữ ký.

Nhiều trò chơi trực tuyến hiện đại áp dụng phương pháp **kiểm tra dữ liệu từ phía máy chủ**, tức là các giá trị quan trọng không được tin cậy ở phía client. Thay vì xử lý toàn bộ logic gameplay ở máy người chơi, máy chủ sẽ tính toán hoặc xác nhận lại các giá trị như sát thương, vị trí, tốc độ hoặc thời gian hồi chiêu, ... Cách tiếp cận này làm giảm đáng kể nguy cơ bị chỉnh sửa bộ nhớ vì client chỉ đóng vai trò hiển thị, không có quyền thay đổi dữ liệu cốt lõi. Tuy nhiên, phương pháp này tăng tải cho máy chủ và không phù hợp với những trò chơi có yêu cầu xử lý thời gian thực cao hoặc gameplay phụ thuộc vào logic client-side.

Cuối cùng, các hệ thống chống gian lận nâng cao sử dụng **mô-đun kernel-mode** để giám sát hoạt động của hệ thống ở mức độ sâu hơn. Các driver này có khả năng theo dõi truy cập bộ nhớ, kiểm tra hành vi của tiến trình và ngăn chặn các thao tác không hợp lệ trước khi chúng ảnh hưởng đến trò chơi. Đây là phương pháp mạnh nhất trong việc chống lại các công cụ gian lận tinh vi, đặc biệt là các cheat có khả năng che giấu tiến trình hoặc thao tác trực tiếp vào bộ nhớ hệ thống. Tuy nhiên, việc sử dụng driver kernel thường gây tranh cãi do nguy cơ ảnh hưởng đến quyền riêng tư, hiệu năng máy tính và độ ổn định hệ thống.

Tóm lại, mỗi phương pháp chống gian lận đều có vai trò và mức độ hiệu quả riêng, đồng thời mang theo những hạn chế nhất định. Trong khi các phương pháp truyền thống như kiểm tra tính toàn vẹn và nhận diện chữ ký phù hợp với các dạng gian lận phổ biến, những kỹ thuật hiện đại như giám sát hành vi và bảo vệ ở tầng kernel được thiết kế để đối phó với các hình thức tấn công phức tạp hơn. Việc đánh giá tổng thể những phương pháp này giúp tạo nền tảng cho việc xây dựng giải pháp phát hiện hành vi tấn công bộ nhớ mà luận văn hướng đến.

1.4.2. Một số hệ thống phổ biến

Thị trường chống gian lận hiện nay bị chi phối bởi các giải pháp lớn, trải dài từ phương pháp quét chữ ký truyền thống đến giám sát tầng nhân (kernel-mode). Ba hệ thống tiêu biểu nhất bao gồm:

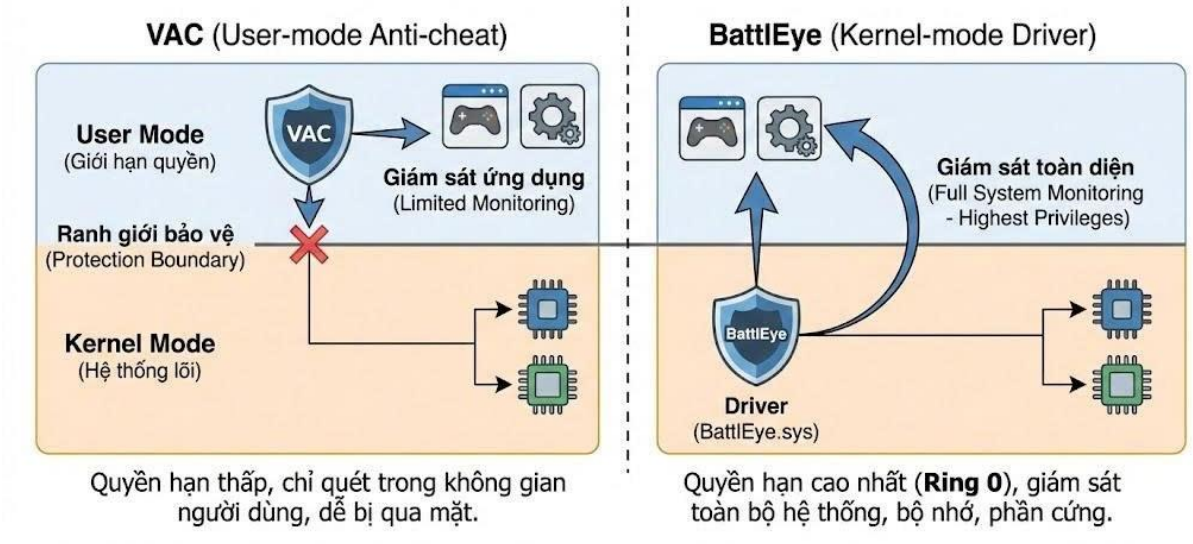
Valve Anti-Cheat (VAC) VAC là giải pháp độc quyền của Valve, hoạt động chủ yếu ở tầng người dùng dựa trên cơ chế phát hiện chữ ký [10]. Khi người chơi kết nối, VAC quét bộ nhớ để tìm chuỗi byte khớp với cơ sở dữ liệu gian lận đã biết. Chiến lược đặc trưng của VAC là cấm chậm, thực thi lệnh cấm theo đợt ngẫu nhiên để gây

khó khăn cho việc phát triển hack. Tuy nhiên, do ưu tiên sự ổn định và quyền riêng tư, khả năng chống lại các tấn công cấp thấp của VAC bị đánh giá thấp hơn các đối thủ.

BattlEye là hệ thống bên thứ ba nổi tiếng với cơ chế bảo vệ chủ động thông qua trình điều khiển tầng nhân (kernel-mode driver) [11]. Cơ chế cốt lõi của BE là sử dụng các callback hệ điều hành (như ObRegisterCallbacks) để chặn quyền truy cập bộ nhớ, vô hiệu hóa các API như OpenProcess hay ReadProcessMemory từ công cụ gian lận. Ngoài ra, BE thực hiện quét bộ nhớ động và giám sát mạng. Dù hiệu quả cao, việc can thiệp sâu của BE đôi khi gây ra vấn đề về hiệu năng và xung đột phần mềm.

Easy Anti-Cheat (EAC) Thuộc sở hữu của Epic Games, EAC sử dụng mô hình lai (hybrid model) kết hợp giữa dịch vụ user-mode và driver kernel-mode. EAC tập trung ngăn chặn việc tiêm mã (code injection) và tải các thư viện động không ký số (unsigned drivers/DLLs) vào kernel. Hệ thống này áp dụng kỹ thuật ảo hóa mã và xáo trộn (obfuscation) để chống phân tích ngược. EAC được ưa chuộng nhờ khả năng hỗ trợ đa nền tảng, nhưng cũng vấp phải các tranh luận về quyền kiểm soát thiết bị tương tự như BattlEye.

Tóm lại, trong khi VAC đại diện cho trường phái truyền thống (dựa trên chữ ký, độ trễ), thì BattlEye và EAC đại diện cho xu hướng hiện đại (giám sát thời gian thực tại kernel), tạo ra lớp phòng thủ chủ động hơn trước các kỹ thuật tấn công tinh vi.



Hình 2.5: So sánh kiến trúc Anti-cheat (Kernel và User)

1.4.3. Hạn chế

Mặc dù các hệ thống chống gian lận hiện đại như BattlEye, Easy Anti-Cheat hay Vanguard đã nâng cao đáng kể rào cản kỹ thuật đối với việc can thiệp vào trò chơi, chúng vẫn tồn tại những hạn chế cốt lõi về khía cạnh kỹ thuật, vận hành và trải nghiệm người dùng. Những hạn chế này không chỉ tạo ra các lỗ hổng mới mà kẻ tấn công có thể khai thác, mà còn đặt ra thách thức lớn cho việc áp dụng rộng rãi trong các môi trường phát triển đa dạng.

Vấn đề xâm phạm quyền riêng tư và rủi ro an toàn hệ thống. Xu hướng chuyển dịch sang sử dụng trình điều khiển kernel-mode trao cho các phần mềm chống gian lận quyền truy cập cao nhất vào hệ điều hành. Điều này đồng nghĩa với việc chúng có khả năng giám sát toàn bộ các tiến trình đang chạy, truy cập mọi tập tin và thu thập dữ liệu phần cứng chi tiết của người dùng. Sự can thiệp sâu này gây ra lo ngại lớn về quyền riêng tư dữ liệu. Nghiêm trọng hơn, nếu chính trình điều khiển chống gian lận tồn tại lỗ hổng bảo mật, kẻ tấn công có thể khai thác nó để chiếm quyền kiểm soát toàn bộ máy tính nạn nhân, biến công cụ bảo vệ thành một vector tấn công nguy hiểm vào hệ thống.

Ảnh hưởng tiêu cực đến hiệu năng và tính ổn định. Các cơ chế bảo vệ chủ động như quét bộ nhớ liên tục, giải mã thời gian thực và kiểm tra tính toàn vẹn hệ thống tiêu tốn tài nguyên CPU và bộ nhớ đáng kể. Trong môi trường trò chơi điện tử đòi hỏi độ trễ thấp và tốc độ khung hình cao, việc tiêu tốn tài nguyên này có thể gây ra hiện tượng giật lag hoặc sụt giảm hiệu năng. Ngoài ra, do hoạt động ở tầng nhân, bất kỳ lỗi lập trình nào trong driver chống gian lận cũng có thể dẫn đến lỗi màn hình xanh, gây treo hệ thống và làm gián đoạn trải nghiệm người dùng nghiêm trọng hơn nhiều so với các ứng dụng user-mode thông thường.

Khả năng bị vượt qua bởi các kỹ thuật tấn công phần cứng và ảo hóa. Cuộc đua giữa bảo mật và tấn công luôn là quá trình phát triển song song. Khi các biện pháp bảo vệ phần mềm trở nên chặt chẽ, cộng đồng phát triển công cụ gian lận đã chuyển hướng sang các giải pháp phần cứng như card Direct Memory Access. Các thiết bị này cho phép đọc và ghi trực tiếp vào bộ nhớ vật lý thông qua cổng PCIe mà

không cần sự can thiệp của CPU hay hệ điều hành, giúp qua mặt hoàn toàn sự giám sát của các anti-cheat hiện hành. Bên cạnh đó, sự phổ biến của các kỹ thuật ảo hóa cho phép cô lập hệ điều hành khách và đánh lừa phần mềm chống gian lận về trạng thái thực của hệ thống.

Rào cản về chi phí và tính phức tạp trong triển khai. Các giải pháp chống gian lận thương mại hàng đầu thường đi kèm với chi phí cấp phép cao, không phù hợp với ngân sách của các nhà phát triển game độc lập hoặc các dự án nghiên cứu quy mô nhỏ. Việc tự phát triển một giải pháp kernel-mode lại đòi hỏi trình độ kỹ thuật chuyên sâu và quy trình ký số nghiêm ngặt từ Microsoft. Do đó, tồn tại một nhu cầu thực tế về các giải pháp giám sát nhẹ nhàng hơn, hoạt động ở tầng user-mode, dễ triển khai và tích hợp nhưng vẫn đảm bảo khả năng phát hiện các hành vi tấn công cơ bản. Đây chính là khoảng trống mà đề tài nghiên cứu hướng tới giải quyết.

1.5. Các nghiên cứu liên quan

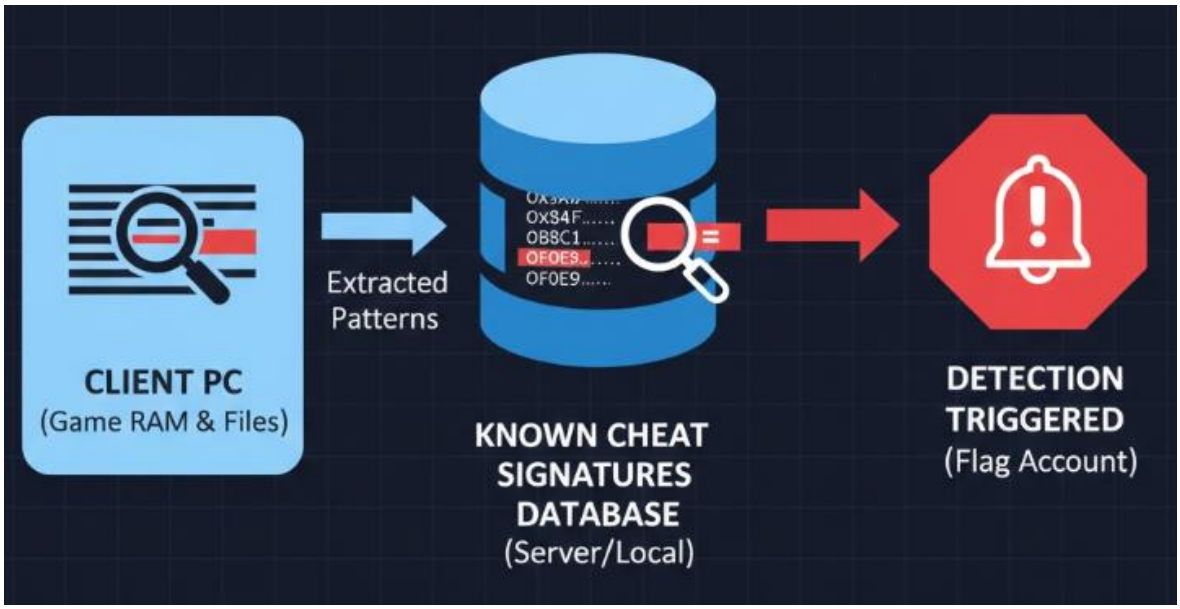
Trong lĩnh vực an toàn thông tin nói chung và bảo mật trò chơi điện tử nói riêng, các nghiên cứu trước đây đã đề xuất nhiều mô hình phát hiện gian lận đa dạng. Dựa trên mức độ can thiệp vào hệ thống và phương thức phân tích, các nghiên cứu này có thể được chia thành ba nhóm chính: (1) Nhận diện dựa trên chữ ký, (2) Giám sát hành vi thời gian thực và (3) Cơ chế bảo vệ đặc quyền cao (Kernel-mode).

1.5.1. Phương pháp nhận diện dựa trên chữ ký

Đây là phương pháp tiếp cận truyền thống nhất, được áp dụng rộng rãi trong các hệ thống chống gian lận thế hệ đầu như Valve Anti-Cheat (VAC).

- Cơ chế hoạt động: Các hệ thống này hoạt động dựa trên việc xây dựng một cơ sở dữ liệu các mẫu nhận dạng (signatures) của các công cụ gian lận đã biết. Khi tiến trình trò chơi khởi chạy, hệ thống thực hiện quét bộ nhớ RAM và các tệp tin liên quan để tìm kiếm các chuỗi byte khớp với cơ sở dữ liệu.
- Ưu điểm: Phương pháp này đảm bảo tính ổn định cao cho hệ thống và ít gây ra báo động giả (False Positives), do việc so khớp dựa trên các mẫu mã độc chính xác.

- Hạn chế: Các nghiên cứu đã chỉ ra rằng phương pháp này mang tính chất "thụ động". Nó hoàn toàn bất lực trước các kỹ thuật tấn công mới chưa được cập nhật trong cơ sở dữ liệu (Zero-day exploits) hoặc các kỹ thuật đa hình (polymorphism) cho phép mã độc tự thay đổi cấu trúc sau mỗi lần chạy để né tránh việc quét chữ ký. Ngoài ra, chiến lược "cấm chậm" (delayed ban) của VAC tuy gây khó khăn cho việc phát triển hack nhưng lại không ngăn chặn được hành vi gian lận ngay tức thì.



Hình 2.6 Kiến trúc hoạt động của phương pháp nhận diện dựa trên ký số

1.5.2. Phương pháp bảo vệ cấp độ nhân

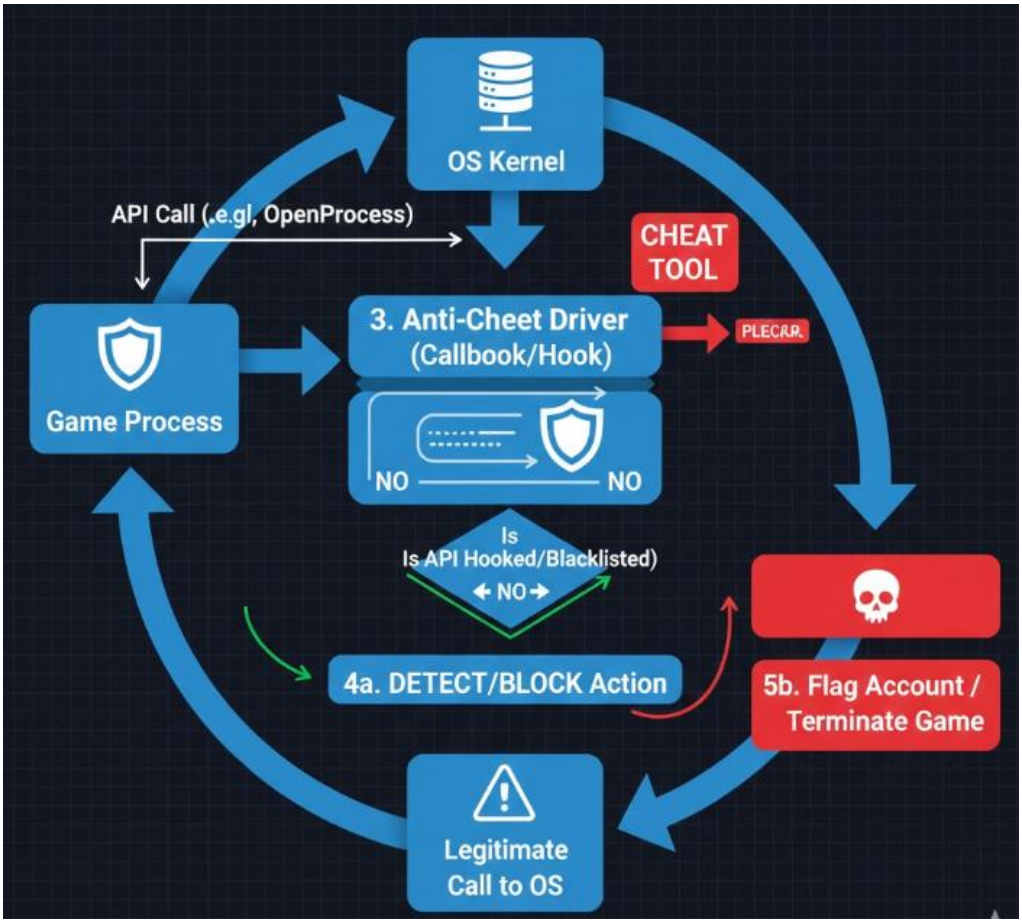
Để khắc phục nhược điểm của việc hoạt động ở tầng người dùng (User-mode), xu hướng hiện đại của các hệ thống như BattlEye (BE) và Easy Anti-Cheat (EAC) là chuyển dịch xuống hoạt động ở tầng nhân hệ điều hành (Kernel-mode/Ring 0).

- Cơ chế hoạt động:
    - BattlEye: Sử dụng các trình điều khiển (drivers) để đăng ký các hàm callback hệ thống (như ObRegisterCallbacks). Cơ chế này cho phép BattlEye chủ động chặn các yêu cầu truy cập bộ nhớ từ tiến trình lạ ngay khi chúng được gửi đến hệ điều hành, vô hiệu hóa các hàm API

như OpenProcess hay ReadProcessMemory trước khi chúng có thể tác động đến game.

  - Easy Anti-Cheat: Sử dụng mô hình lai (hybrid model), kết hợp giữa dịch vụ User-mode và Driver Kernel-mode. EAC tập trung vào việc ngăn chặn việc tải các trình điều khiển không ký số (unsigned drivers) và kiểm soát chặt chẽ việc nạp thư viện động (DLL injection) vào không gian nhân.

- Hạn chế: Mặc dù đạt hiệu quả bảo mật cao, phương pháp này vấp phải các rào cản lớn về quyền riêng tư dữ liệu khi yêu cầu quyền kiểm soát toàn diện thiết bị người dùng. Bên cạnh đó, việc can thiệp sâu vào nhân hệ điều hành tiềm ẩn rủi ro gây mất ổn định hệ thống (màn hình xanh - BSOD) và sụt giảm hiệu năng máy tính. Chi phí triển khai và độ phức tạp kỹ thuật cũng khiến phương pháp này khó tiếp cận đối với các dự án nghiên cứu hoặc phát triển game quy mô vừa và nhỏ.



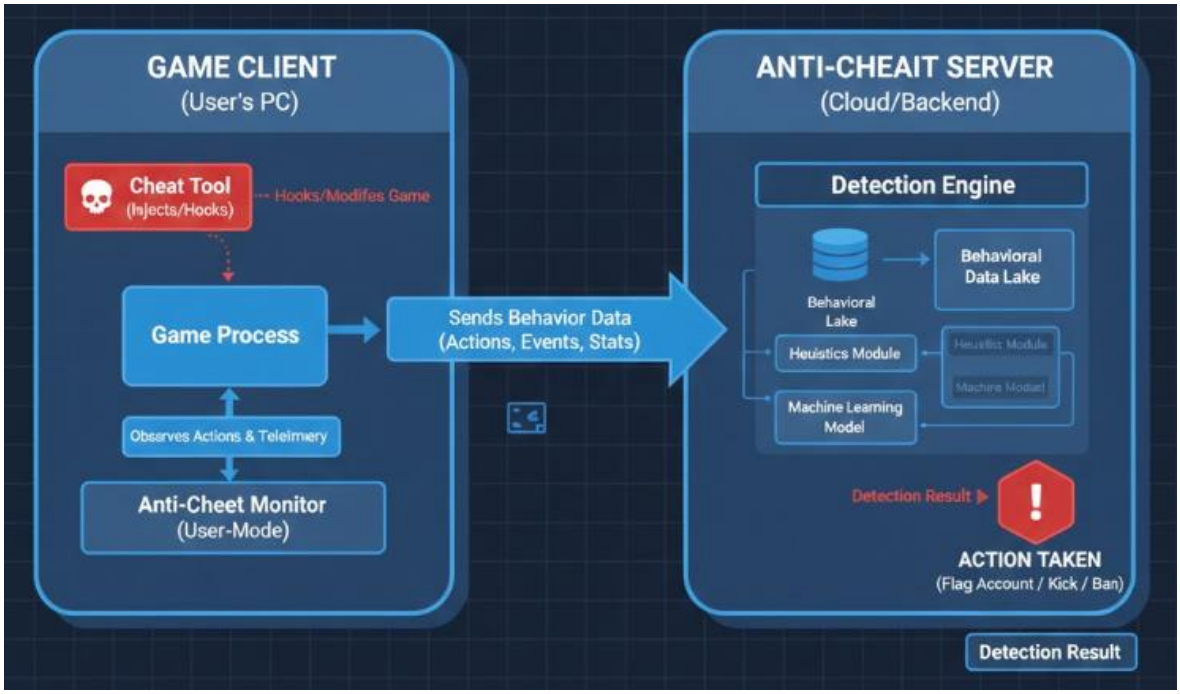
Hình 2.7: Kiến trúc hoạt động của phương pháp bảo vệ mức độ nhân

1.5.3. Phương pháp giám sát hành vi tại tầng ứng dụng

Đây là hướng tiếp cận cân bằng giữa hiệu quả bảo mật và tính khả thi trong triển khai, đồng thời là hướng nghiên cứu chính của đề tài này.

- Cơ chế hoạt động: Thay vì quét toàn bộ hệ thống hay can thiệp vào nhân, phương pháp này tập trung giám sát các "hành vi" cụ thể thông qua kỹ thuật API Hooking. Hệ thống sẽ theo dõi tần suất và ngữ cảnh của các lời gọi hàm nhạy cảm như WriteProcessMemory hay CreateRemoteThread. Bất kỳ hành vi nào cố gắng thay đổi quyền thực thi của vùng nhớ mã lệnh (Code Section) thông qua VirtualProtect đều được đánh dấu là bất thường.
- So sánh với các phương pháp khác: Khác với kiểm tra chữ ký, giám sát hành vi có khả năng phát hiện các cuộc tấn công chưa biết trước (unknown attacks) dựa trên sự bất thường của logic truy cập. So với giải pháp Kernel-mode,

phương pháp này nhẹ nhàng hơn, dễ tích hợp, giảm thiểu rủi ro xung đột hệ thống và phù hợp với mô hình bảo mật hướng ứng dụng mà không xâm phạm quyền riêng tư quá mức của người dùng.



Hình 2.8 Kiến trúc hoạt động của phương pháp giám sát hành vi tại ứng dụng

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

Trong lĩnh vực an toàn thông tin, mặc dù các kỹ thuật tấn công vào trò chơi điện tử và tấn công vào các ứng dụng phần mềm doanh nghiệp đều chia sẻ nhiều nền tảng kỹ thuật chung—như khai thác lỗi tràn bộ đệm, tiêm mã hay dịch ngược mã nguồn—song bản chất, động cơ và đối tượng tác động của chúng lại tồn tại những khác biệt căn bản. Việc phân định rõ ranh giới này là vô cùng quan trọng để hiểu được tại sao các giải pháp bảo mật truyền thống như tường lửa hay phần mềm diệt virus thường không phát huy hiệu quả trong việc ngăn chặn gian lận trò chơi.

Đối với các ứng dụng truyền thống và hệ thống thông tin, các cuộc tấn công thường tuân theo mô hình đe dọa nhắm vào ba trụ cột bảo mật CIA (Confidentiality - Tính bí mật, Integrity - Tính toàn vẹn, Availability - Tính sẵn sàng) [16]. Kẻ tấn công thường tìm cách leo thang đặc quyền để đánh cắp dữ liệu nhạy cảm, mã hóa dữ liệu tống tiền hoặc làm gián đoạn dịch vụ. Ngược lại, trong môi trường trò chơi điện

tử, mục tiêu của kẻ tấn công hiếm khi là phá hoại hệ thống hay đánh cắp thông tin người dùng. Thay vào đó, trọng tâm của chúng nằm ở việc vi phạm tính toàn vẹn logic của trò chơi nhằm tạo ra lợi thế cạnh tranh không công bằng.

Sự khác biệt về mục tiêu này dẫn đến sự thay đổi hoàn toàn về chiến lược phòng thủ. Một hành vi "ghi dữ liệu vào bộ nhớ" có thể bị coi là độc hại trong ứng dụng ngân hàng, nhưng lại là một thao tác hoàn toàn bình thường trong quá trình vận hành của một engine game phức tạp. Do đó, việc so sánh các đặc điểm giữa hai loại hình tấn công này sẽ cung cấp cơ sở lý luận vững chắc cho việc thiết kế các cơ chế phát hiện chuyên biệt, tập trung vào hành vi thay vì chỉ dựa vào các dấu hiệu mã độc truyền thống.

1.6.1. Mục tiêu tấn công

Sự khác biệt căn bản nhất giữa tấn công vào trò chơi điện tử và tấn công vào các ứng dụng phần mềm truyền thống nằm ở động cơ và mục đích cuối cùng của kẻ tấn công. Đối với các ứng dụng truyền thống (như hệ thống ngân hàng, cơ sở dữ liệu doanh nghiệp hay các dịch vụ web), mục tiêu của kẻ tấn công thường xoay quanh việc phá vỡ ba trụ cột bảo mật: tính bí mật, tính toàn vẹn và tính sẵn sàng. Cụ thể, các cuộc tấn công này thường hướng tới việc chiếm quyền điều khiển hệ thống, đánh cắp dữ liệu nhạy cảm, mã hóa dữ liệu để tống tiền hoặc gây từ chối dịch vụ nhằm phá hoại hoạt động kinh doanh. Trong bối cảnh này, sự thành công của cuộc tấn công thường đồng nghĩa với sự tổn thất hoặc ngừng trệ của hệ thống mục tiêu.

Ngược lại, trong môi trường trò chơi điện tử, kẻ tấn công (thường là người chơi gian lận) không có mục đích phá hủy hệ thống hay đánh cắp thông tin người dùng. Mục tiêu tối thượng của họ là đạt được lợi thế cạnh tranh không công bằng so với những người chơi khác. Để đạt được điều này, kẻ tấn công tìm cách thay đổi trạng thái của trò chơi đang lưu trong bộ nhớ theo hướng có lợi cho bản thân, chẳng hạn như tăng lượng máu, nhìn xuyên vật thể, hoặc tự động ngắm bắn.

Điểm đáng lưu ý là tấn công game mang tính chất "ký sinh": kẻ gian lận cần trò chơi hoạt động ổn định để có thể thụ hưởng lợi thế mà họ vừa tạo ra. Họ không muốn làm sập máy chủ hay gây lỗi ứng dụng, vì điều đó sẽ làm gián đoạn quá trình

chơi của chính họ. Do đó, thay vì phá hoại, tấn công game tập trung vào việc thao túng logic và dữ liệu trong thời gian thực một cách tinh vi để hệ thống vẫn vận hành bình thường nhưng kết quả đầu ra đã bị sai lệch theo ý đồ của kẻ tấn công.

1.6.2. Đặc điểm môi trường

Yếu tố thứ hai tạo nên sự khác biệt giữa tấn công game và tấn công ứng dụng truyền thống nằm ở đặc thù của môi trường vận hành. Trong khi các ứng dụng phần mềm thông thường (như trình xử lý văn bản, trình duyệt web hay ứng dụng quản lý) hoạt động chủ yếu dựa trên kiến trúc hướng sự kiện, thì trò chơi điện tử lại vận hành dựa trên một vòng lặp vô hạn được gọi là "vòng lặp trò chơi". Sự khác biệt về kiến trúc này dẫn đến những đặc điểm môi trường riêng biệt, ảnh hưởng trực tiếp đến cả phương thức tấn công lẫn chiến lược phòng thủ.

Thứ nhất là tính thời gian thực và sự liên tục của logic. Trong các ứng dụng truyền thống, chương trình thường ở trạng thái chờ và chỉ xử lý dữ liệu khi có tác động từ người dùng (nhấn phím, click chuột). Ngược lại, trò chơi điện tử phải liên tục cập nhật trạng thái của thế giới ảo—tính toán vật lý, xử lý AI, cập nhật đồ họa—với tần suất rất cao (thường là 60 lần/giây hoặc cao hơn). Do đó, các hành vi tấn công vào game thường phải được thực hiện đồng bộ với tốc độ xử lý này. Kẻ tấn công không thể chỉ đơn thuần sửa đổi một giá trị và để đó; họ phải liên tục "đóng băng" hoặc ghi đè giá trị đó sau mỗi khung hình để chống lại cơ chế cập nhật lại của game engine.

Thứ hai là sự biến động dữ liệu tốc độ cao. Bộ nhớ của một ứng dụng văn phòng thường khá tĩnh tại; dữ liệu chỉ thay đổi khi có thao tác cụ thể. Trong game, hàng nghìn biến số thay đổi liên tục trong mỗi tích tắc ngay cả khi người chơi không thực hiện hành động nào (ví dụ: tọa độ của các vật thể chuyển động, thời gian hồi chiêu, trạng thái môi trường). Điều này tạo ra một môi trường đầy "nhiễu" đối với các hệ thống giám sát. Việc phân biệt giữa một sự thay đổi giá trị hợp lệ (do logic game tạo ra) và một sự thay đổi bất thường do hack trở nên khó khăn hơn nhiều so với việc phát hiện sự thay đổi file hệ thống trong môi trường Windows truyền thống.

Thứ ba là yêu cầu khắt khe về độ trễ. Môi trường game đòi hỏi phản hồi tức thì. Bất kỳ sự chậm trễ nào trong quá trình xử lý cũng ảnh hưởng tiêu cực đến trải nghiệm người chơi (gây ra hiện tượng giật, lag). Điều này đặt ra thách thức lớn cho các giải pháp bảo mật: chúng không thể sử dụng các phương pháp quét sâu hay tạm dừng tiến trình để kiểm tra như các phần mềm diệt virus truyền thống thường làm. Hệ thống phát hiện tấn công trong game buộc phải hoạt động cực kỳ nhẹ nhàng, tối ưu hóa để giám sát trong thời gian thực mà không tiêu tốn quá nhiều tài nguyên CPU.

1.6.3. Đặc thù nhận diện hành vi bất thường

Từ sự khác biệt về mục tiêu và môi trường hoạt động, bài toán nhận diện hành vi bất thường trong trò chơi điện tử mang những đặc thù riêng biệt, phức tạp hơn nhiều so với việc phát hiện mã độc trong các ứng dụng truyền thống. Trong khi bảo mật truyền thống tập trung vào việc xác định "ai" đang thực hiện hành động (xác thực quyền truy cập), bảo mật game lại phải tập trung vào việc phân tích "cái gì" và "như thế nào" hành động đó được thực hiện (xác thực tính hợp lý của hành vi).

Thứ nhất là sự phụ thuộc vào ngữ nghĩa và logic trò chơi. Đối với một phần mềm diệt virus thông thường, việc một tiến trình ghi dữ liệu vào một vùng nhớ đã được cấp quyền là hành vi hợp lệ. Tuy nhiên, trong game, hành vi này phải được đối chiếu với logic của trò chơi. Ví dụ: Việc thay đổi tọa độ nhân vật từ điểm A sang điểm B là hợp lệ về mặt hệ thống (lệnh ghi bộ nhớ thành công), nhưng nếu khoảng cách giữa A và B quá lớn so với tốc độ di chuyển tối đa cho phép trong khoảng thời gian giữa hai khung hình, đó là hành vi bất thường. Do đó, hệ thống phát hiện không thể hoạt động độc lập mà phải "hiểu" được các quy tắc vật lý và logic của trò chơi để phân biệt giữa một thao tác chơi game xuất sắc và một hành vi gian lận.

Thứ hai là thách thức trong việc phân biệt hành vi người và máy. Nhiều công cụ gian lận hiện nay không can thiệp thô bạo vào bộ nhớ mà mô phỏng thao tác đầu vào, ví dụ như các phần mềm tự động ngắm bắn hoặc giảm độ giật. Để phát hiện các hành vi này, hệ thống phải phân tích các đặc trưng thống kê của dữ liệu đầu vào. Hành vi của con người luôn có độ trễ, sai số ngẫu nhiên và đường cong di chuyển chuột không hoàn hảo. Ngược lại, hành vi của máy móc thường có độ chính xác tuyệt đối,

phản xạ tức thời và tính lặp lại cao. Việc nhận diện đòi hỏi các thuật toán phân tích heuristic hoặc học máy để tìm ra các mẫu phi nhân văn, điều mà các giải pháp bảo mật truyền thống hiếm khi phải xử lý.

Thứ ba là vấn đề về độ nhiễu và tỷ lệ dương tính giả. Môi trường mạng của trò chơi trực tuyến thường không ổn định, dẫn đến các hiện tượng như mất gói tin hoặc độ trễ mạng. Những hiện tượng này có thể khiến nhân vật di chuyển giật cục hoặc dịch chuyển tức thời, tạo ra các dấu hiệu rất giống với hành vi gian lận. Ngoài ra, những người chơi chuyên nghiệp có kỹ năng rất cao đôi khi thực hiện những thao tác có độ chính xác và phản xạ vượt xa người bình thường, dễ bị hệ thống nhận nhầm là máy. Do đó, ngưỡng phát hiện trong game phải được tinh chỉnh cực kỳ cẩn thận để cân bằng giữa việc bắt kẻ gian lận và không làm ảnh hưởng đến người chơi chân chính—một thách thức lớn hơn nhiều so với việc cách ly một file mã độc rõ ràng trong môi trường doanh nghiệp.

1.7. Kết luận chương 1

Chương 1 đã trình bày một cái nhìn tổng quan và hệ thống về bối cảnh an ninh trong trò chơi điện tử, với trọng tâm là các kỹ thuật tấn công bộ nhớ. Qua việc phân tích kiến trúc bộ nhớ của hệ điều hành Windows và cơ chế vận hành của các trò chơi, có thể thấy rằng bộ nhớ RAM chính là nơi diễn ra các hoạt động gian lận. Các kỹ thuật tấn công từ đơn giản như sửa đổi giá trị biến đến phức tạp như tiêm mã hay vá mã tại thời điểm chạy đều khai thác trực tiếp các quyền truy cập và API mà hệ điều hành cung cấp, tạo ra những thách thức lớn đối với tính toàn vẹn của trò chơi.

Những phân tích trên đã chỉ ra một lỗ hổng trong các giải pháp hiện tại: nhu cầu về một cơ chế giám sát, hoạt động hiệu quả ở tầng user-mode mà không truy cậpsâu vào hệ thống, nhưng vẫn đủ khả năng phát hiện các hành vi can thiệp bất thường. Đây chính là tiền đề để luận văn đi sâu vào nghiên cứu các cơ sở lý thuyết và kỹ thuật cụ thể nhằm xây dựng hệ thống phát hiện trong các chương tiếp theo. Chương 2 sẽ trình bày chi tiết về các kỹ thuật giám sát hành vi, bao gồm cơ chế Hook API, cấu trúc tập tin thực thi PE và các phương pháp phân tích bộ nhớ, làm cơ sở nền tảng cho việc thiết kế giải pháp của đề tài.

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

2.1. Giới thiệu chương

Tiếp nối các phân tích nền tảng tại Chương 1 về thực trạng tấn công bộ nhớ và những hạn chế của các giải pháp chống gian lận hiện hành, Chương 2 tập trung vào việc hiện thực hóa các mục tiêu nghiên cứu thông qua việc thiết kế và xây dựng một hệ thống giám sát cụ thể. Nếu như Chương 1 đã khẳng định nhu cầu về một giải pháp bảo mật hoạt động hiệu quả ở tầng user-mode nhưng vẫn đảm bảo khả năng phát hiện các hành vi can thiệp sâu, thì Chương 2 sẽ trình bày chi tiết phương pháp luận và các kỹ thuật công nghệ để giải quyết bài toán đó.

Mục tiêu cốt lõi của chương này là xây dựng một hệ thống phòng thủ chủ động có khả năng giám sát thời gian thực các hành vi truy cập vào không gian địa chỉ của tiến trình trò chơi. Để đạt được điều này, đề tài lựa chọn hướng tiếp cận dựa trên kỹ thuật ngăn chặn lời gọi hàm và phân tích cấu trúc tập tin thực thi. Đây là các kỹ thuật cho phép hệ thống "đứng giữa" tiến trình tấn công và tiến trình game, từ đó kiểm soát và ngăn chặn các hành vi đọc/ghi bộ nhớ trái phép hoặc tiêm mã độc ngay từ khi chúng được khởi tạo.

Về mặt công nghệ, hệ thống được thiết kế và phát triển trên nền tảng ngôn ngữ lập trình C/C++ kết hợp chặt chẽ với các thư viện lập trình hệ thống Windows. Việc lựa chọn C/C++ là yếu tố then chốt để đảm bảo hiệu năng tối ưu và độ trễ thấp nhất – một yêu cầu bắt buộc đối với các phần mềm hoạt động song song với trò chơi điện tử. Đồng thời, chương cũng sẽ đi sâu vào việc ứng dụng các thư viện hỗ trợ hook chuyên dụng (như Microsoft Detours hoặc MinHook) để đảm bảo tính ổn định và khả năng tương thích cao trên môi trường Windows.

Cấu trúc nội dung của chương được bố cục theo trình tự logic từ lý thuyết đến thực tiễn. Trước tiên, phần cơ sở lý thuyết sẽ trình bày chi tiết về cấu trúc bộ nhớ Windows, định dạng tập tin PE và nguyên lý hoạt động của các kỹ thuật Hooking. Tiếp theo, phần phân tích thiết kế sẽ mô tả kiến trúc tổng thể của hệ thống, bao gồm sơ đồ luồng dữ liệu và cơ chế tương tác giữa các thành phần. Cuối cùng, phần xây

dựng sẽ đi sâu vào việc cài đặt chi tiết các mô-đun chức năng như Injector (tiêm module giám sát) và Core Monitor (giám sát và phát hiện), làm tiền đề cho việc thử nghiệm và đánh giá hiệu quả trong Chương 3.

2.2. Cơ sở lý thuyết và công nghệ nền tảng.

Việc xây dựng hệ thống giám sát an ninh hoạt động song song với trò chơi điện tử đòi hỏi sự can thiệp sâu vào luồng thực thi và cấu trúc bộ nhớ của tiến trình. Để hiện thực hóa mục tiêu này, đề tài sử dụng hai nền tảng kỹ thuật cốt lõi: phân tích cấu trúc tập tin thực thi (PE) và kỹ thuật đánh chặn lời gọi hàm (API Hooking).

2.2.1. Cấu trúc tập tin thực thi PE

Để xây dựng một hệ thống giám sát và can thiệp vào tiến trình trò chơi trên hệ điều hành Windows, việc nắm vững cấu trúc tập tin thực thi PE là yêu cầu tiên quyết. PE là định dạng chuẩn cho các tập tin thực thi (.exe), thư viện liên kết động (.dll), và trình điều khiển hệ thống (.sys) trên các phiên bản Windows 32-bit và 64-bit [3]. Mọi mã lệnh và dữ liệu của trò chơi đều được tổ chức theo định dạng này trước khi được hệ điều hành nạp vào bộ nhớ RAM. Hiểu rõ cấu trúc PE giúp hệ thống xác định được đâu là vùng chứa mã lệnh để bảo vệ tính toàn vẹn, đâu là vùng chứa dữ liệu để thực hiện Hooking.

a) Cấu trúc Header và các Section

Về mặt tổng thể, một tập tin PE không phải là một khối dữ liệu lộn xộn mà được tổ chức tuyến tính thành các phần riêng biệt, bắt đầu bằng các Header chứa thông tin quản lý và theo sau là các Section chứa nội dung thực tế[14].

DOS Header và MS-DOS Stub: Đây là phần đầu tiên của file, tồn tại chủ yếu để đảm bảo tính tương thích ngược. Khi một file PE được chạy, 64 byte đầu tiên DOS Header được kiểm tra. Nếu file được chạy trên hệ điều hành không hỗ trợ, phần Stub sẽ in ra thông báo "This program cannot be run in DOS mode". Đối với hệ thống giám sát, giá trị quan trọng nhất ở đây là trường e_lfanew tại offset 0x3C, chứa địa chỉ trỏ đến PE Header thực sự.

NT Headers (PE Header): Đây là nơi chứa các thông tin để hệ điều hành nạp chương trình. Nó bao gồm:

- *File Header:* Chứa thông tin về kiến trúc máy tính (x86 hay x64), số lượng section, và thời gian biên dịch.
- *Optional Header:* là phần bắt buộc đối với file thực thi. Nó chứa **AddressOfEntryPoint (OEP)** – địa chỉ của lệnh đầu tiên sẽ được CPU thực thi khi chương trình khởi chạy. Trong các kỹ thuật tấn công, kẻ gian thường sửa đổi OEP để chuyển hướng luồng thực thi sang mã độc trước khi quay lại game gốc. Hệ thống giám sát cần kiểm tra tính toàn vẹn của OEP để phát hiện các hành vi tiêm mã tĩnh.

Section Table và các Section: Sau phần Header là bảng chỉ mục mô tả các phân đoạn của chương trình. Mỗi Section có vai trò riêng biệt:

- .text (hoặc .code): Chứa mã máy (machine code) của trò chơi. Đây là vùng nhớ có quyền "Thực thi" và "Đọc". Bất kỳ hành vi nào cố gắng ghi vào vùng này thường là dấu hiệu của việc vá mã hoặc Inline Hook trái phép.
- .data và .rdata: Chứa các biến toàn cục và hằng số.
- .reloc: Chứa thông tin tái định vị, cho phép file thực thi được nạp vào các địa chỉ bộ nhớ khác nhau nhờ cơ chế ASLR - Address Space Layout Randomization nhằm chống lại các cuộc tấn công tràn bộ đệm.

b) Bảng địa chỉ nhập và vai trò trong giám sát

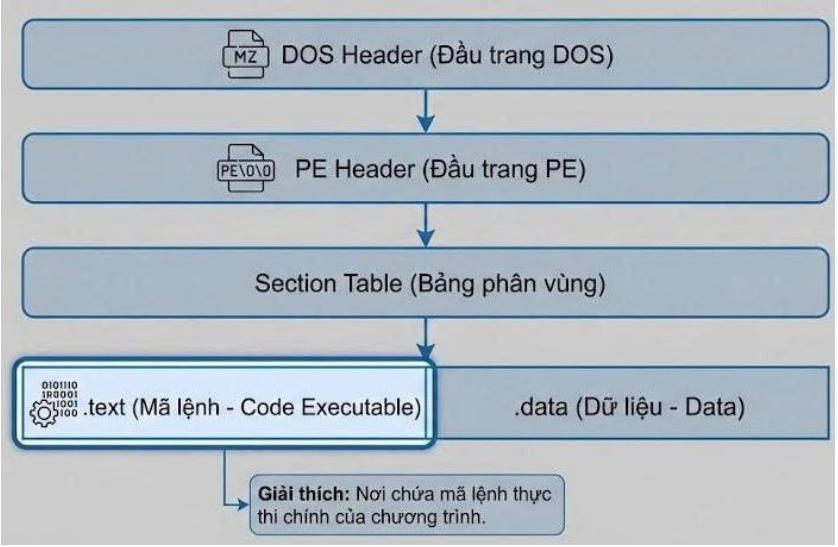
Một trong những thành phần quan trọng nhất của cấu trúc PE đối với đề tài này là Bảng địa chỉ nhập, thường nằm trong section .rdata.

Cơ chế liên kết động: Hầu hết các trò chơi không tự mình thực hiện các tác vụ cấp thấp (như đọc file, vẽ đồ họa, xử lý mạng) mà gọi các hàm từ thư viện liên kết động DLL của Windows (như kernel32.dll, user32.dll, d3d9.dll). Tuy nhiên, khi biên dịch game, trình biên dịch không thể biết chính xác địa chỉ của các hàm này trong bộ nhớ máy người chơi vì chúng thay đổi theo từng lần khởi động.

Vai trò của IAT: IAT đóng vai trò như một danh bạ điện thoại. Khi chương trình được nạp, Windows Loader sẽ tìm địa chỉ thực tế của các hàm API cần thiết (ví dụ: WriteProcessMemory) và điền vào bảng IAT. Khi game cần gọi hàm API, nó không gọi trực tiếp mà sẽ tra cứu địa chỉ trong bảng IAT rồi nhảy đến đó.

Ứng dụng trong giám sát: Đây chính là điểm yếu kiến trúc mà hệ thống giám sát của đề tài khai thác. Thay vì để IAT trỏ đến hàm API gốc của Windows, hệ thống giám sát sẽ ghi đè địa chỉ trong IAT bằng địa chỉ của hàm giám sát do chúng ta tự viết.

Ví dụ: Nếu ta thay địa chỉ của OpenProcess trong IAT bằng hàm My_OpenProcess, thì mỗi khi game (hoặc module lạ trong game) gọi OpenProcess, luồng thực thi sẽ chạy qua hàm của ta trước. Tại đây, ta có thể kiểm tra xem ai đang gọi, mục đích là gì, và quyết định cho phép hay chặn lại. Kỹ thuật này an toàn và ít gây lỗi hơn so với việc sửa đổi trực tiếp mã máy, do đó được ưu tiên sử dụng trong các tình huống giám sát tiêu chuẩn.



Hình 2.1: Cấu trúc tệp tin PE (Portable Executable)

2.2.2. Kỹ thuật Hook API

Trong lĩnh vực lập trình hệ thống và bảo mật phần mềm, kỹ thuật Hook API được định nghĩa là phương pháp can thiệp và thay đổi luồng thực thi của chương trình nhằm giám sát, gỡ lỗi hoặc mở rộng chức năng của hệ điều hành. Đối với đề tài này, Hook API đóng vai trò là cơ chế cốt lõi của hệ thống giám sát, cho phép thiết lập các điểm kiểm soát ngay tại giao diện lập trình của Windows. Thay vì để tiến trình trò chơi tương tác trực tiếp với nhân hệ điều hành thông qua các hàm API tiêu chuẩn, hệ thống sẽ điều hướng các lời gọi này qua một tầng trung gian do người quản trị kiểm

soát. Tại đây, các thuật toán phân tích hành vi sẽ được kích hoạt để xác minh tính hợp lệ của yêu cầu truy cập trước khi quyết định chuyển tiếp hoặc từ chối thực thi.

a) Nguyên lý hoạt động của Inline Hooking và IAT Hooking

Việc lựa chọn kỹ thuật Hook phù hợp phụ thuộc vào kiến trúc của ứng dụng mục tiêu và mức độ can thiệp mong muốn. Hai phương pháp phổ biến nhất hiện nay là IAT Hooking và Inline Hooking, mỗi phương pháp sở hữu những đặc trưng kỹ thuật riêng biệt.

IAT Hooking: Phương pháp này dựa trên việc thao tác với Bảng địa chỉ nhập trong cấu trúc tập tin PE. IAT là bảng tham chiếu chứa địa chỉ thực của các hàm API được nạp từ các thư viện liên kết động.

- *Cơ chế hoạt động:* Về bản chất, IAT Hooking thay thế địa chỉ của hàm API đích (ví dụ: kernel32!OpenProcess) trong bảng IAT bằng địa chỉ của hàm giám sát. Khi ứng dụng thực hiện lời gọi hàm, trình biên dịch sẽ tra cứu bảng IAT và vô tình chuyển hướng luồng thực thi đến hàm của hệ thống bảo vệ.
- *Đánh giá:* Ưu điểm lớn nhất của phương pháp này là tính an toàn và dễ triển khai, do không yêu cầu sửa đổi mã máy tại vùng nhớ thực thi. Tuy nhiên, IAT Hooking bộc lộ hạn chế lớn về khả năng bao quát; nó hoàn toàn bất lực trước các kỹ thuật tấn công sử dụng nạp động thông qua GetProcAddress hoặc các hành vi gọi hàm cấp thấp Syscall, do đó thường chỉ đóng vai trò bổ trợ trong các hệ thống hiện đại.

Inline Hooking: Đây là kỹ thuật tiên tiến hơn, can thiệp trực tiếp vào mã máy của hàm API trong bộ nhớ, cho phép giám sát toàn diện mọi lời gọi hàm bất kể phương thức triệu gọi.

- *Cơ chế JMP Patching:* Hệ thống thực hiện ghi đè một số byte đầu tiên tại điểm nhập Entry Point của hàm API gốc bằng một lệnh nhảy không điều kiện. Lệnh này có nhiệm vụ chuyển hướng con trỏ lệnh đến địa chỉ của hàm giám sát.
- *Cơ chế Trampoline:* Một thách thức kỹ thuật của Inline Hooking là việc ghi đè lệnh JMP sẽ phá hủy các chỉ lệnh gốc của hàm API. Để bảo toàn tính toàn vẹn chức năng, hệ thống sử dụng kỹ thuật "Trampoline". Trampoline là một vùng nhớ

đệm chứa các chỉ lệnh gốc đã bị ghi đè, theo sau là một lệnh nhảy quay trở lại địa chỉ kế tiếp trong hàm API gốc. Quy trình thực thi khép kín diễn ra như sau:

- Ứng dụng gọi API → Lệnh JMP chuyển hướng đến Hàm giám sát.
- Hàm giám sát thực hiện phân tích an ninh.
- Nếu hợp lệ, gọi hàm Trampoline để thực thi các lệnh gốc.
- Trampoline nhảy về phần thân còn lại của API để hoàn tất tác vụ.

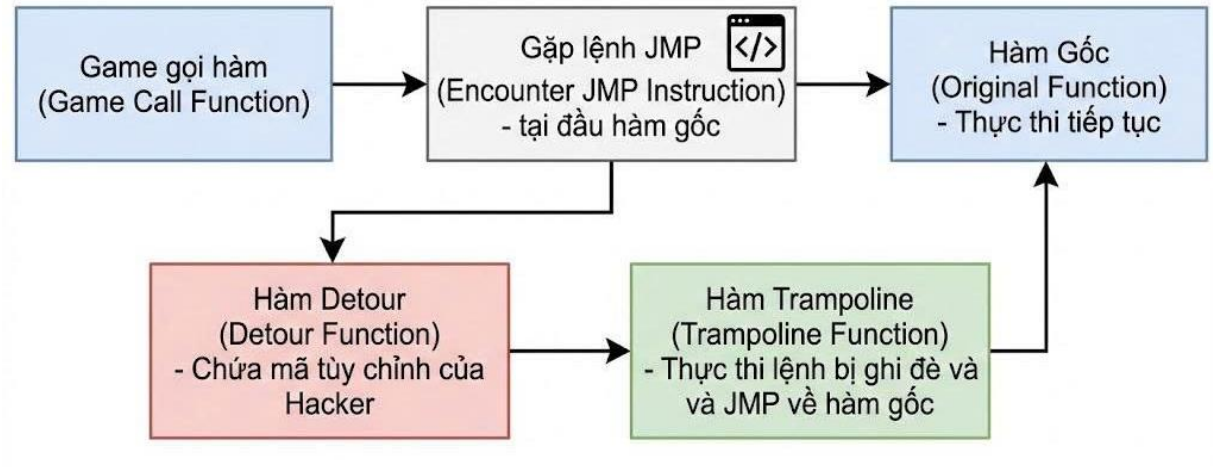
b) Lựa chọn thư viện hỗ trợ kỹ thuật

Việc triển khai Inline Hooking thủ công tiềm ẩn nhiều rủi ro về tính ổn định, đặc biệt trong môi trường đa luồng và trên các kiến trúc vi xử lý x64 sử dụng địa chỉ tương đối. Do đó, việc sử dụng các thư viện chuyên dụng đã được kiểm chứng là giải pháp tối ưu để đảm bảo độ tin cậy của hệ thống.

Microsoft Detours: Là bộ thư viện chuẩn công nghiệp do Microsoft Research phát triển, Detours nổi tiếng với độ ổn định cao và khả năng xử lý xuất sắc các tình huống biên. Tuy nhiên, do các rào cản về giấy phép thương mại và hạn chế hỗ trợ kiến trúc x64 trong phiên bản miễn phí, thư viện này không được lựa chọn làm giải pháp chính cho đề tài.

MinHook (Minimalistic x86/x64 API Hooking Library): Đề tài lựa chọn MinHook làm nền tảng kỹ thuật chính nhờ sự cân bằng giữa hiệu năng và tính năng. Thư viện này giải quyết triệt để các thách thức kỹ thuật phức tạp:

- *Tương thích kiến trúc x64:* Tự động tính toán và tái cấu trúc các lệnh nhảy tương đối, đảm bảo mã lệnh hoạt động chính xác trên nền tảng Windows 64-bit.
- *Tối ưu hóa tài nguyên:* Với kích thước nhỏ gọn, MinHook dễ dàng tích hợp vào module giám sát DLL mà không gây ảnh hưởng đáng kể đến hiệu năng của trò chơi.
- *Cơ chế Transactional Hooking:* Hỗ trợ việc kích hoạt hoặc vô hiệu hóa đồng loạt các điểm Hook trong một thao tác nguyên tử, giúp ngăn chặn tình trạng xung đột bộ nhớ khi tiến trình game đang vận hành với cường độ cao.



Hình 2.2: Cơ chế Inline Hooking và Trampoline

2.2.3. Các hàm API Windows nhạy cảm cần giám sát

Để thực hiện các hành vi can thiệp vào trò chơi như đọc dữ liệu ESP, sửa đổi giá trị hay tiêm mã độc, các công cụ gian lận dù tinh vi đến đâu cũng buộc phải tương tác với hệ điều hành Windows thông qua các hàm API tiêu chuẩn. Những hàm này đóng vai trò như những "cổng giao tiếp" duy nhất cho phép một tiến trình bên ngoài tác động vào không gian bộ nhớ của tiến trình trò chơi. Do đó, việc xác định và giám sát chặt chẽ các nhóm hàm API nhạy cảm chính là chìa khóa để xây dựng một hệ thống phòng thủ hiệu quả. Dựa trên phân tích hành vi của các công cụ tấn công phổ biến, đề tài tập trung vào ba nhóm hàm trọng yếu sau:

a) Nhóm hàm truy cập và định danh tiến trình

Đây là bước đầu tiên và quan trọng nhất trong chuỗi tấn công. Trước khi có thể đọc hay ghi bất kỳ dữ liệu nào, công cụ tấn công bắt buộc phải giành được quyền truy cập hợp lệ tới tiến trình trò chơi.

OpenProcess: Hàm này được sử dụng để mở một đối tượng tiến trình hiện có. Các công cụ như Cheat Engine thường gọi hàm này với cờ quyền hạn cao nhất (PROCESS_ALL_ACCESS hoặc PROCESS_VM_WRITE | PROCESS_VM_OPERATION) để giành toàn quyền kiểm soát game.*Chiến lược giám sát:* Bằng cách Hook hàm này, hệ thống có thể kiểm tra danh tính của tiến trình đang yêu cầu truy cập. Nếu phát hiện yêu cầu đến từ một tiến trình lạ hoặc không được ký số, hệ thống có thể bỏ các quyền hạn nguy hiểm (như quyền ghi bộ nhớ)

hoặc trả về mã lỗi ACCESS_DENIED, qua đó vô hiệu hóa cuộc tấn công ngay từ bước "mở cửa".

OpenThread: Tương tự như OpenProcess, nhưng hàm này tác động lên các luồng cụ thể trong game. Kẻ tấn công có thể sử dụng hàm này để tạm dừng luồng xử lý chính của game nhằm thực hiện các sửa đổi mà không bị phát hiện.

b) Nhóm hàm thao tác bộ nhớ

Sau khi đã có quyền truy cập, kẻ tấn công sử dụng nhóm hàm này để trực tiếp thay đổi trạng thái của trò chơi. Đây là nơi diễn ra các hành vi gian lận cụ thể như "bất tử", "vô hạn đạn" hay "nhìn xuyên tường".

ReadProcessMemory: Cho phép đọc dữ liệu từ một vùng địa chỉ cụ thể trong không gian bộ nhớ của game. Hàm này là nền tảng của các loại hack thông tin như ESP (hiển thị thông tin đối thủ), Radar Hack hoặc Bot tự động tìm quái. *Vai trò giám sát:* Việc giám sát hàm này giúp phát hiện các mẫu hành vi quét bộ nhớ bất thường, ví dụ như một tiến trình liên tục đọc hàng loạt địa chỉ bộ nhớ mà không có lý do hợp lệ.

WriteProcessMemory: Cho phép ghi dữ liệu vào không gian bộ nhớ của game. Đây là hàm nguy hiểm nhất, chịu trách nhiệm trực tiếp cho việc thay đổi giá trị biến hoặc ghi đè mã máy. *Vai trò giám sát:* Hệ thống cần kiểm tra địa chỉ đích của lệnh ghi. Nếu lệnh ghi nhắm vào vùng mã lệnh (.text section) hoặc các cấu trúc dữ liệu nhạy cảm, đó là dấu hiệu rõ ràng của hành vi tấn công.

VirtualProtect / VirtualProtectEx: Hàm này cho phép thay đổi thuộc tính bảo vệ của các trang nhớ. Mặc định, vùng chứa mã lệnh của game được thiết lập quyền PAGE_EXECUTE_READ (chỉ thực thi và đọc). Để thực hiện hành vi tiêm mã hoặc Hook trái phép, kẻ tấn công bắt buộc phải gọi hàm này để đổi quyền sang PAGE_EXECUTE_READWRITE. Việc phát hiện một lời gọi VirtualProtect nhắm vào vùng mã lệnh là chỉ báo tin cậy của một cuộc tấn công.

c) Nhóm hàm tiêm mã và tải thư viện

Nhóm hàm này phục vụ cho kỹ thuật DLL Injection, cho phép kẻ tấn công đưa mã độc (thường đóng gói dưới dạng file .dll) vào chạy ngay bên trong không gian địa chỉ của game.

CreateRemoteThread: Hàm này cho phép tạo một luồng mới chạy trong không gian địa chỉ của tiến trình khác. Đây là kỹ thuật "kinh điển" để kích hoạt mã độc sau khi đã được chép vào bộ nhớ game.*Cơ chế chặn:* Giám sát hàm này cho phép hệ thống ngăn chặn việc khởi chạy các luồng lạ không xuất phát từ module chính của trò chơi.

LoadLibrary / LdrLoadDll: Các hàm này chịu trách nhiệm nạp các thư viện liên kết động vào tiến trình. Kẻ tấn công thường ép tiến trình game gọi hàm này để nạp file DLL gian lận của chúng. Bằng cách kiểm soát danh sách các thư viện được nạp, hệ thống có thể chặn đứng các DLL không nằm trong danh sách trắng hoặc không có chữ ký số hợp lệ.

Việc giám sát toàn diện ba nhóm API này tạo nên một lớp phòng thủ đa tầng, đảm bảo rằng mọi nỗ lực can thiệp từ bên ngoài—dù là truy cập, sửa đổi hay tiêm nhiễm—đều bị phát hiện và ngăn chặn kịp thời.

2.3. Phân tích yêu cầu và kiến trúc hệ thống

Sau khi đã thiết lập nền tảng lý thuyết vững chắc về các kỹ thuật can thiệp hệ thống như phân tích cấu trúc PE và Hook API, bước tiếp theo là chuyển hóa các nguyên lý này thành một thiết kế hệ thống cụ thể, khả thi và hiệu quả. Việc xây dựng một giải pháp an ninh hoạt động trong môi trường thời gian thực của trò chơi điện tử đặt ra những thách thức kiến trúc đặc thù mà các phần mềm ứng dụng thông thường ít khi phải đối mặt. Hệ thống không chỉ phải đảm bảo khả năng phát hiện chính xác các hành vi tấn công mà còn phải duy trì độ trễ tối thiểu và tính ổn định cao để không làm gián đoạn trải nghiệm người dùng.

Mục này sẽ đi sâu vào phân tích các yêu cầu kỹ thuật cốt lõi của bài toán, từ đó đề xuất một mô hình kiến trúc tổng thể phù hợp. Kiến trúc này được thiết kế theo mô hình tác tử, tách biệt rõ ràng giữa thành phần tiêm nhiễm chịu trách nhiệm triển khai và thành phần lõi giám sát chịu trách nhiệm phân tích logic. Đồng thời, các luồng dữ liệu và cơ chế tương tác giữa hệ thống bảo vệ và tiến trình trò chơi cũng sẽ được mô tả chi tiết, tạo tiền đề cho việc hiện thực hóa các chức năng trong giai đoạn tiếp theo.

2.3.1. Phân tích yêu cầu bài toán

Để đảm bảo hệ thống phát hiện tấn công bộ nhớ hoạt động hiệu quả và khả thi trong môi trường thực tế, việc xác định rõ ràng các yêu cầu kỹ thuật là bước tiên quyết. Các yêu cầu này đóng vai trò như bộ đặc tả kỹ thuật, định hướng cho toàn bộ quá trình thiết kế kiến trúc và lựa chọn giải thuật ở các giai đoạn sau.

a) Yêu cầu chức năng

Nhóm yêu cầu này xác định các hành vi và chức năng cụ thể mà hệ thống bắt buộc phải thực hiện để đối phó hiệu quả với các kỹ thuật tấn công đã được phân tích ở Chương 1.

Giám sát thời gian thực: Hệ thống phải có khả năng thiết lập các điểm kiểm soát ngay tại thời điểm khởi chạy trò chơi. Cụ thể:

- Ngăn chặn thành công các lời gọi hàm API nhạy cảm thuộc nhóm truy cập tiến trình (OpenProcess, OpenThread), thao tác bộ nhớ (WriteProcessMemory, VirtualProtect) và tiêm mã (LoadLibrary, CreateRemoteThread).
- Thu thập đầy đủ thông tin ngữ cảnh của mỗi lời gọi, bao gồm: Mã định danh tiến trình gọi, địa chỉ bộ nhớ đích, kích thước dữ liệu và các cờ quyền hạn (Access Flags).

Phát hiện hành vi bất thường: Hệ thống cần tích hợp bộ quy tắc phân tích để phân biệt giữa các hoạt động hợp pháp của hệ điều hành và hành vi can thiệp trái phép. Cơ chế phát hiện cần đáp ứng:

- *Xác thực nguồn gốc:* Phân biệt được tiến trình gọi là module nội bộ của game hay là một công cụ bên ngoài (như Cheat Engine, Debugger).
- *Phân tích mẫu hành vi:* Phát hiện các mẫu đáng ngờ, ví dụ: nỗ lực thay đổi quyền thực thi tại vùng nhớ .text (dấu hiệu của code patching) hoặc mở quyền truy cập toàn phần (PROCESS_ALL_ACCESS) từ một tiến trình không được ký số.

Cơ chế cảnh báo và ngăn chặn: Khi phát hiện vi phạm, hệ thống phải thực hiện các tác vụ phản hồi tức thời:

- *Ngăn chặn chủ động:* Vô hiệu hóa hành vi tấn công bằng cách trả về mã lỗi giả lập (ví dụ: ACCESS_DENIED hoặc INVALID_HANDLE) cho tiến trình tấn công, đảm bảo lệnh ghi bộ nhớ hoặc tiêm mã không thể thực thi.

- *Ghi nhật ký:* Lưu trữ vết của cuộc tấn công bao gồm thời gian, loại hành vi và đối tượng thực hiện vào tệp nhật ký hoặc gửi về máy chủ giám sát để phục vụ công tác hậu kiểm.

b) Yêu cầu phi chức năng

Trong bối cảnh đặc thù của trò chơi điện tử, các yêu cầu phi chức năng về hiệu năng và độ ổn định đóng vai trò quan trọng tương đương với yêu cầu bảo mật, bởi một giải pháp an ninh làm gián đoạn trải nghiệm người dùng sẽ không có giá trị thực tiễn.

Hiệu năng và Độ trễ tối thiểu: Trò chơi điện tử hiện đại thường vận hành với tốc độ khung hình cao (từ 60 đến 144 FPS trở lên), đồng nghĩa với việc chu kỳ xử lý chỉ diễn ra trong vài mili-giây.

- Hệ thống giám sát phải đảm bảo độ trễ do quá trình Hook gây ra là không đáng kể (tính bằng micro-giây).
- Các thuật toán phân tích trong hàm callback phải được tối ưu hóa tối đa, tránh các tác vụ I/O nặng (như ghi đĩa liên tục) làm sụt giảm FPS (Frames Per Second) hoặc gây hiện tượng giật cục (stuttering).

Tính ổn định và An toàn luồng (Stability & Thread Safety): Do hoạt động dưới dạng một thư viện liên kết động DLL được tiêm trực tiếp vào không gian địa chỉ của trò chơi, bất kỳ lỗi lập trình nào của hệ thống giám sát (như rò rỉ bộ nhớ, truy cập con trỏ null) đều có thể khiến trò chơi bị đóng đột ngột.

- Hệ thống phải đảm bảo tính an toàn luồng, xử lý triệt để các vấn đề xung đột dữ liệu khi nhiều luồng của game cùng gọi một hàm API tại một thời điểm.
- Cơ chế Hook phải có khả năng tự phục hồi hoặc tháo gỡ an toàn khi cần thiết.

Tính tương thích hệ thống: Hệ thống cần hoạt động ổn định trên các phiên bản hệ điều hành Windows phổ biến (Windows 10, Windows 11) và hỗ trợ đầy đủ cho kiến trúc vi xử lý x64 – chuẩn kiến trúc của hầu hết các tựa game AAA hiện nay. Ngoài ra, giải pháp cần giảm thiểu xung đột với các phần mềm hợp pháp khác như phần mềm diệt virus hoặc các lớp phủ của Steam/Discord.

2.3.2. Mô hình kiến trúc tổng thể

Dựa trên các yêu cầu phân tích, kiến trúc của hệ thống được thiết kế theo mô hình hướng tác tử (Agent-based Model), hoạt động trực tiếp trong không gian người dùng. Mô hình này chia hệ thống thành ba thành phần logic chính tương tác chặt chẽ với nhau: **Game Process** (Tiến trình trò chơi), **Monitor Agent** (Tác tử giám sát) và **Analysis Core** (Lõi phân tích). Sự phân chia này đảm bảo tính module hóa, cho phép tách biệt giữa nhiệm vụ thu thập dữ liệu và nhiệm vụ xử lý logic.

Sơ đồ khối trên minh họa luồng dữ liệu và cơ chế kiểm soát giữa các thành phần. Chức năng cụ thể của từng khối được mô tả như sau:

Game Process (Tiến trình trò chơi - Host Environment): Đây là môi trường mục tiêu cần được bảo vệ. Trong kiến trúc này, Game Process đóng vai trò là "đối tượng" (Host). Mọi hoạt động của trò chơi, từ việc tải tài nguyên, xử lý đồ họa đến cập nhật logic, đều diễn ra tại đây. Hệ thống bảo vệ không chạy như một tiến trình riêng biệt bên ngoài mà được nhúng trực tiếp vào không gian địa chỉ của Game Process dưới dạng một thư viện liên kết động (Dynamic Link Library - DLL). Điều này cho phép hệ thống truy cập cùng một không gian bộ nhớ và quyền hạn như trò chơi.

Monitor Agent (Tác tử giám sát - Interceptor): Monitor Agent đóng vai trò là "cảm biến" đầu vào của hệ thống. Thành phần này chịu trách nhiệm thiết lập các điểm Hook tại các hàm API nhạy cảm (như OpenProcess, WriteProcessMemory) ngay khi hệ thống được khởi tạo.

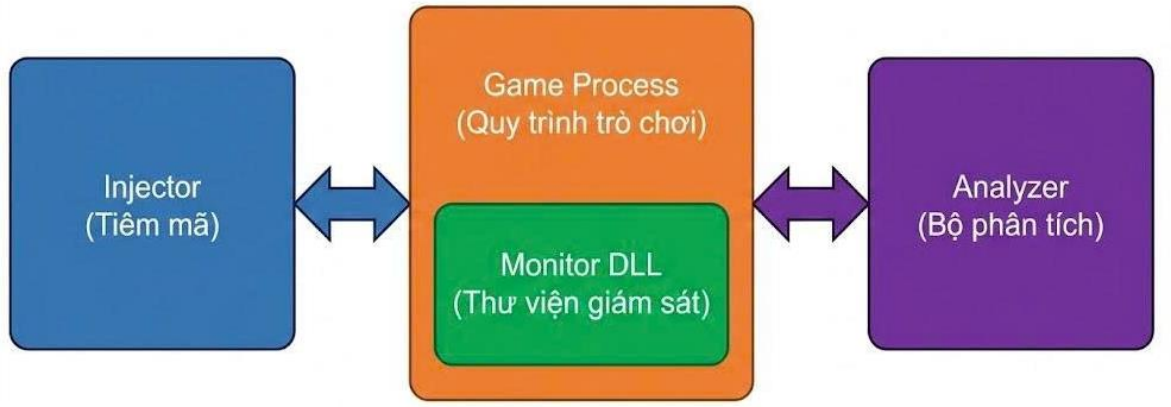
- *Nhiệm vụ:* Nhiệm vụ chính của Agent là "đứng gác" tại các cổng giao tiếp API. Khi một lời gọi hàm xảy ra, Agent sẽ chặn luồng thực thi trước khi nó chạm tới nhân hệ điều hành.

- *Thu thập:* Tại đây, Agent trích xuất toàn bộ thông tin ngữ cảnh: Ứng dụng nào đang gọi? (Caller ID), Gọi hàm gì? (API Name), và Tham số là gì? Dữ liệu thô này sau đó được chuyển tiếp ngay lập tức đến Analysis Core.

Analysis Core (Lõi phân tích - Decision Maker): Đây là "bộ não" xử lý trung tâm của hệ thống. Analysis Core tiếp nhận dữ liệu từ Monitor Agent và thực hiện quy trình thẩm định dựa trên các tập luật đã được định nghĩa trước.

- *Xử lý:* Core so sánh thông tin lời gọi với danh sách trắng Whitelist các tiến trình tin cậy hoặc các mẫu hành vi hợp lệ. Ví dụ: Nếu phát hiện một tiến trình lạ cố gắng ghi vào vùng nhớ .text của game, Core sẽ đánh dấu đây là hành vi vi phạm.
- *Phản hồi:* Dựa trên kết quả phân tích, Core trả về quyết định cho Monitor Agent: "Cho phép" hoặc "Từ chối". Nếu từ chối, Core cũng kích hoạt mô-đun ghi nhật ký (Logger) để lưu lại bằng chứng tấn công.

Mô hình **Game Process ↔ Monitor Agent ↔ Analysis Core** tạo ra một vòng khép kín. Ưu điểm lớn nhất của kiến trúc này là khả năng phản ứng tức thời. Do tất cả các thành phần đều nằm trong cùng một không gian địa chỉ, việc chuyển giao dữ liệu giữa Agent và Core diễn ra trực tiếp qua bộ nhớ mà không cần cơ chế giao tiếp liên tiến trình (IPC) phức tạp và tốn kém tài nguyên, đáp ứng tốt yêu cầu về hiệu năng của trò chơi điện tử.



Hình 2.3: Mô hình kiến trúc tổng thể hệ thống

2.3.3. Các thành phần chức năng chính

Để hiện thực hóa kiến trúc tổng thể đã đề xuất, hệ thống được chia nhỏ thành ba phân hệ chức năng modules chuyên biệt. Mỗi module đảm nhận một nhiệm vụ cụ thể trong quy trình bảo vệ, từ việc triển khai thành phần giám sát, thu thập dữ liệu cho đến phân tích và ra quyết định. Sự phân chia này tuân thủ nguyên tắc "Single Responsibility" (Đơn nhiệm) trong thiết kế phần mềm, giúp hệ thống dễ dàng bảo trì và mở rộng.

a) Module Injector

Module Injector là thành phần khởi đầu của hệ thống, được thiết kế dưới dạng một ứng dụng thực thi độc lập (Standalone Executable). Nhiệm vụ cốt lõi của Injector là đưa thư viện liên kết động chứa mã giám sát vào bên trong không gian địa chỉ ảo của tiến trình trò chơi. Do hệ điều hành Windows thực hiện cơ chế cách ly tiến trình để bảo vệ bộ nhớ, Module Injector phải sử dụng các kỹ thuật lập trình hệ thống nâng cao để vượt qua rào cản này [4].

Các chức năng cụ thể bao gồm:

- **Định danh mục tiêu:** Sử dụng API CreateToolhelp32Snapshot để quét danh sách các tiến trình đang hoạt động và xác định Mã định danh (PID) của trò chơi dựa trên tên tập tin thực thi.
- **Cấp phát bộ nhớ từ xa (Remote Memory Allocation):** Sử dụng VirtualAllocEx để xin cấp phát một vùng nhớ trống bên trong tiến trình trò chơi, đủ lớn để chứa đường dẫn đến file DLL giám sát.
- **Ghi dữ liệu và Kích hoạt (Injection & Execution):** Ghi đường dẫn file DLL vào vùng nhớ vừa cấp phát thông qua WriteProcessMemory, sau đó sử dụng CreateRemoteThread để buộc tiến trình trò chơi tạo ra một luồng mới. Luồng này sẽ gọi hàm LoadLibrary, nạp Module Interceptor vào bộ nhớ và kích hoạt cơ chế bảo vệ từ bên trong.

b) Module Interceptor

Module Interceptor là thành phần hoạt động trực tiếp trong tiến trình trò chơi (In-process Component). Đây là lớp vỏ bọc kỹ thuật chịu trách nhiệm thiết lập các điểm kiểm soát trên các hàm API nhạy cảm.

Cơ chế hoạt động của module này dựa trên kỹ thuật Inline Hooking với sự hỗ trợ của thư viện MinHook:

- **Khởi tạo Hook:** Ngay khi được nạp vào, Interceptor sẽ xác định địa chỉ bộ nhớ của các hàm mục tiêu như kernel32!OpenProcess hay kernel32!WriteProcessMemory.

- **Cài đặt vùng nhớ riêng:** Module thực hiện sao lưu các byte đầu tiên của hàm gốc ra một vùng nhớ riêng, sau đó ghi đè lệnh nhảy (JMP) tại đầu hàm gốc để chuyển hướng luồng thực thi về hàm xử lý của hệ thống.
- **Thu thập thông tin:** Khi một lời gọi hàm bị chặn, Interceptor sẽ trích xuất toàn bộ tham số đầu vào (như Handle, Address, Buffer Size) và thông tin về ứng dụng gọi. Module này đóng vai trò như một "cổng gác", đảm bảo không một lệnh gọi nhạy cảm nào có thể đi qua mà không bị kiểm tra.

c) Phân tích hành vi

Module Analyzer là "bộ não" của hệ thống, chứa các thuật toán logic để quyết định tính hợp lệ của hành vi. Module này nhận dữ liệu đầu vào từ Interceptor và trả về kết quả quyết định (Cho phép hoặc Chặn).

Chức năng phân tích được chia thành các lớp logic:

- **Kiểm tra quyền truy cập:** Phân tích cờ truy cập trong hàm OpenProcess. Nếu phát hiện yêu cầu PROCESS_ALL_ACCESS hoặc PROCESS_VM_WRITE từ một tiến trình không nằm trong danh sách trắng, Analyzer sẽ ra lệnh chặn.
- **Bảo vệ vùng nhớ mã lệnh:** Đối với hàm WriteProcessMemory và VirtualProtect, Analyzer kiểm tra địa chỉ đích. Nếu địa chỉ này nằm trong vùng .text (vùng chứa mã máy của game), hệ thống lập tức xác định đây là hành vi Code Patching và kích hoạt cơ chế ngăn chặn.
- **Phản hồi và Ghi nhật ký:**
  - *Cơ chế chặn:* Nếu phát hiện vi phạm, Analyzer chỉ đạo Interceptor không gọi hàm Trampoline (hàm gốc) mà trả về mã lỗi ACCESS_DENIED ngay lập tức, vô hiệu hóa tác động của cuộc tấn công.
  - *Cơ chế log:* Ghi lại chi tiết thông tin vụ việc (Timestamp, Attacker PID, Target API) ra tệp nhật ký cục bộ hoặc giao diện điều khiển (Console) để phục vụ quá trình gỡ lỗi và phân tích mẫu tấn công.

2.3.4. Kiến trúc hệ thống

a) Kiến trúc tổng thể

Hệ thống được thiết kế theo mô Agent-based Model hoạt động tại tầng người dùng. Khác với các giải pháp truyền thống hoạt động độc lập bên ngoài, hệ thống đề xuất nhúng trực tiếp một mô-đun giám sát vào không gian địa chỉ ảo (Virtual Address Space) của tiến trình trò chơi. Kiến trúc này loại bỏ nhu cầu giao tiếp liên tiến trình (IPC) phức tạp, cho phép giám sát và phản hồi thời gian thực với độ trễ tối thiểu.

Mô hình bao gồm ba thành phần chức năng tương tác chặt chẽ: (1) Injector chịu trách nhiệm triển khai, (2) Interceptor chịu trách nhiệm thu thập dữ liệu và (3) Analyzer đóng vai trò là lõi phân tích logic.

b) Cơ chế triển khai và Tàng hình

Để đưa mô-đun giám sát vào tiến trình đích mà không cần sửa đổi mã nguồn trò chơi, hệ thống sử dụng kỹ thuật Remote Thread Injection. Quy trình bao gồm việc cấp phát bộ nhớ từ xa và tạo luồng thực thi mới để nạp thư viện liên kết động (DLL) chứa mã giám sát.

Nhằm đối phó với các kỹ thuật phát hiện ngược (Anti-Anti-Cheat), hệ thống tích hợp cơ chế PEB Unlinking. Ngay sau khi khởi tạo, mô-đun giám sát tự động ngắt liên kết node dữ liệu của chính nó khỏi danh sách InLoadOrderModuleList trong Cấu trúc Khối Môi trường Tiến trình (PEB). Kỹ thuật này giúp mô-đun trở nên "vô hình" trước các hàm API liệt kê danh sách mô-đun tiêu chuẩn của Windows.

c) Cơ chế thu thập dữ liệu

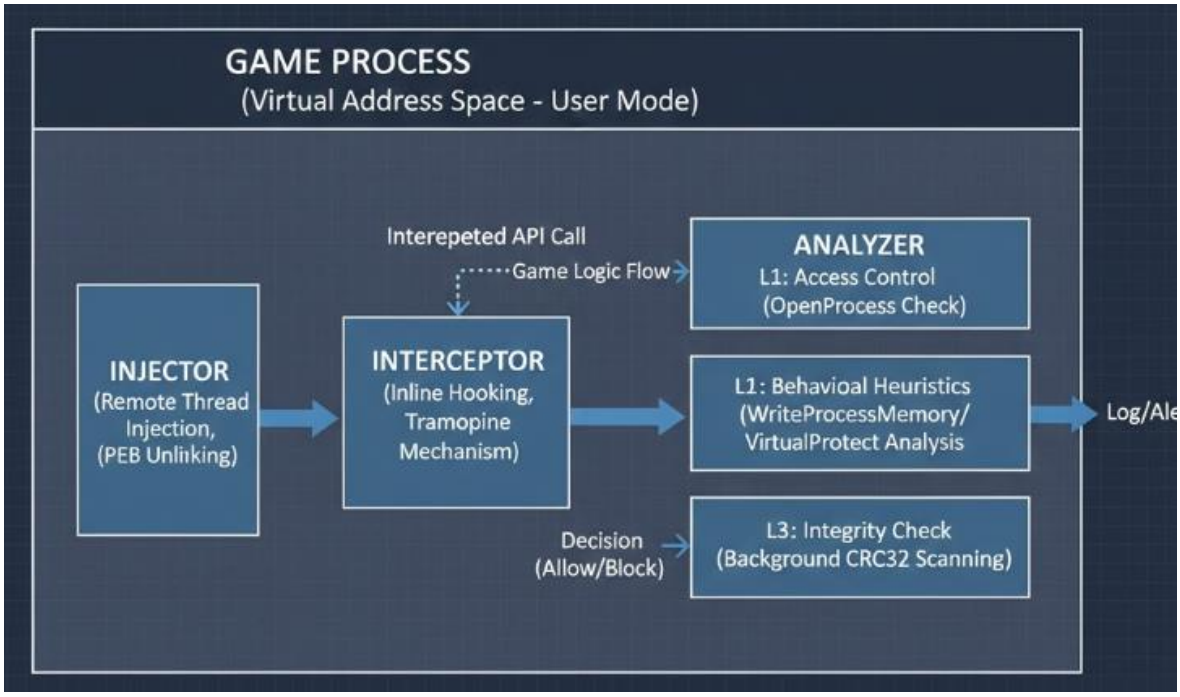
Hệ thống sử dụng kỹ thuật Inline Hooking làm phương thức thu thập dữ liệu chủ đạo. Thông qua thư viện MinHook, hệ thống ghi đè 5 byte đầu tiên (trên kiến trúc x86) hoặc sử dụng lệnh nhảy gián tiếp (trên x64) tại điểm nhập (Entry Point) của các hàm API nhạy cảm.

Để đảm bảo tính toàn vẹn chức năng của trò chơi, cơ chế Trampoline được áp dụng. Trampoline lưu trữ các chỉ lệnh gốc bị ghi đè và cung cấp cầu nối để luồng thực thi quay trở lại hàm gốc nếu hành vi được xác định là an toàn. Điều này đảm bảo hệ thống chỉ can thiệp khi phát hiện bất thường, không làm gián đoạn luồng xử lý nghiệp vụ của trò chơi.

d) Thuật toán phát hiện đa tầng

Lõi phân tích (Analyzer) áp dụng chiến lược phòng thủ theo chiều sâu (Defense in Depth) với ba lớp lọc tuần tự:

- Lớp 1: Kiểm soát quyền truy cập (Access Control Layer) Hệ thống giám sát hàm OpenProcess để phân tích ngữ cảnh người gọi. Dựa trên Mã định danh tiến trình (PID) và Cờ quyền hạn (Access Flags), hệ thống ngăn chặn các tiến trình không tin cậy yêu cầu quyền truy cập nhạy cảm như PROCESS_VM_WRITE hoặc PROCESS_ALL_ACCESS. Đây là chốt chặn đầu tiên ngăn kẻ tấn công giành quyền điều khiển (Handle) tới trò chơi.
- Lớp 2: Phân tích hành vi bất thường (Behavioral Heuristics Layer) Hệ thống tập trung vào các hàm thao tác bộ nhớ như WriteProcessMemory và VirtualProtect. Thuật toán kiểm tra địa chỉ đích của lệnh ghi: nếu địa chỉ nằm trong phân vùng mã lệnh (.text section), hệ thống xác định đây là hành vi vá mã (Code Patching) hoặc tiêm mã trái phép và lập tức ngăn chặn. Đồng thời, các yêu cầu thay đổi quyền bảo vệ bộ nhớ sang PAGE_EXECUTE_READWRITE cũng bị từ chối để ngăn ngừa việc thực thi mã độc.
- Lớp 3: Kiểm tra tính toàn vẹn (Integrity Assurance Layer) Để phát hiện các thay đổi tinh vi không thông qua API, một luồng quét ngầm thực hiện tính toán định kỳ giá trị băm (Checksum CRC32) của vùng nhớ mã lệnh. Kết quả được so sánh với chữ ký gốc của trò chơi; bất kỳ sự sai lệch nào cũng kích hoạt cơ chế cảnh báo vi phạm tính toàn vẹn.



Hình 2.4: Kiến trúc hệ thống

2.4. Xây dựng các phân hệ chức năng

Sau khi đã hoàn tất việc thiết kế kiến trúc tổng thể và xác định rõ các thành phần logic, bước tiếp theo là hiện thực hóa thiết kế này thành các mô-đun phần mềm cụ thể. Quá trình xây dựng hệ thống đòi hỏi sự am hiểu sâu sắc về lập trình hệ thống trên nền tảng Windows, đặc biệt là việc thao tác với các cấu trúc dữ liệu cấp thấp và quản lý bộ nhớ thủ công.

Để đảm bảo các yêu cầu khắt khe về hiệu năng và tính tương thích đã đề ra, toàn bộ mã nguồn của hệ thống được phát triển bằng ngôn ngữ C++ (tiêu chuẩn C++17). Việc lựa chọn C++ cho phép kiểm soát tối đa tài nguyên phần cứng, giảm thiểu độ trễ trong quá trình xử lý sự kiện và tạo điều kiện thuận lợi cho việc tương tác trực tiếp với các hàm API của Windows. Bên cạnh đó, thư viện MinHook được tích hợp để giải quyết bài toán kỹ thuật phức tạp về Inline Hooking trên kiến trúc x64.

Mục này sẽ trình bày chi tiết quy trình cài đặt kỹ thuật cho ba phân hệ chức năng cốt lõi: Module Injector (cơ chế triển khai), Module Interceptor (cơ chế thu thập) và Module Analyzer (cơ chế phân tích). Mỗi phân hệ sẽ được mô tả thông qua các

đoạn mã giả hoặc lưu đồ thuật toán nhằm minh họa rõ nét logic xử lý bên trong, từ lúc khởi tạo tiến trình cho đến khi phát hiện và ngăn chặn hành vi tấn công.

2.4.1. Thiết kế và cài đặt cơ chế tiêm

Module Injector đóng vai trò là thành phần khởi tạo của hệ thống, chịu trách nhiệm triển khai thư viện giám sát vào bên trong tiến trình mục tiêu. Do kiến trúc bảo vệ bộ nhớ của hệ điều hành Windows ngăn cản việc một tiến trình truy cập trực tiếp vào không gian địa chỉ của tiến trình khác, hệ thống phải sử dụng các kỹ thuật lập trình hệ thống để vượt qua rào cản này.

a) Kỹ thuật DLL Injection để nạp module giám sát

Trong số các phương pháp tiêm mã phổ biến (như SetWindowsHookEx, Manual Mapping, AppInit_DLLs), đề tài lựa chọn phương pháp **Remote Thread Injection** kết hợp với hàm LoadLibrary. Đây là phương pháp cân bằng tốt nhất giữa tính ổn định (ít gây crash game) và khả năng tương thích trên các phiên bản Windows khác nhau.

Quy trình kỹ thuật được thực hiện qua 05 bước tuần tự như sau:

1. **Định danh và Mở quyền truy cập:**

- Sử dụng API CreateToolhelp32Snapshot để chụp ảnh toàn bộ danh sách tiến trình đang chạy. Hệ thống duyệt qua danh sách này (Process32Next), so sánh tên tệp thực thi (ví dụ: csgo.exe) để tìm ra Mã định danh tiến trình (Process ID - PID).
- Sau khi có PID, gọi hàm OpenProcess để lấy handle điều khiển tiến trình. Quyền truy cập yêu cầu tối thiểu bao gồm: PROCESS_CREATE_THREAD (để tạo luồng), PROCESS_VM_OPERATION (để thao tác máy ảo), và PROCESS_VM_WRITE (để ghi dữ liệu).

2. **Cấp phát bộ nhớ (Remote Memory Allocation):**

- Tiến trình game không thể đọc đường dẫn file DLL nằm trong bộ nhớ của Injector. Do đó, ta phải sao chép chuỗi ký tự đường dẫn này sang bộ nhớ của game.
- Sử dụng hàm VirtualAllocEx để xin cấp phát một vùng nhớ trong không gian địa chỉ của game.

- *Tham số quan trọng:* flAllocationType phải là MEM_COMMIT | MEM_RESERVE và flProtect phải là PAGE_READWRITE.

3. **Ghi dữ liệu (Writing Payload):**

- Sử dụng WriteProcessMemory để ghi đường dẫn tuyệt đối của file Monitor.dll vào vùng nhớ vừa cấp phát ở bước 2.

4. **Địa chỉ hàm nạp (Address Resolution):**

- Cần xác định địa chỉ của hàm LoadLibraryA (hoặc LoadLibraryW) trong thư viện kernel32.dll.

- *Lưu ý kỹ thuật:* Trên Windows, địa chỉ cơ sở của kernel32.dll thường giống nhau ở mọi tiến trình trong cùng một phiên làm việc. Do đó, ta có thể lấy địa chỉ LoadLibraryA ngay trong tiến trình Injector bằng GetProcAddress và sử dụng địa chỉ đó cho tiến trình game.

5. **Thực thi luồng (Remote Execution):**

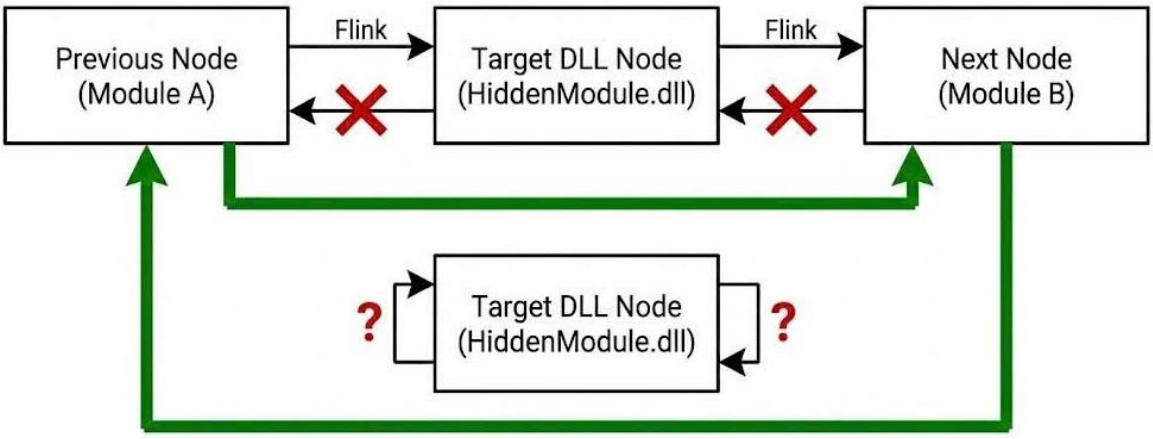
- Gọi hàm CreateRemoteThread. Hàm này sẽ tạo một luồng mới trong game.
- Điểm bắt đầu (Start Routine) của luồng chính là địa chỉ LoadLibraryA.
- Tham số truyền vào (Parameter) chính là địa chỉ vùng nhớ chứa đường dẫn DLL.

- *Kết quả:* Tiến trình game sẽ thực thi lệnh LoadLibrary("path/to/monitor.dll"), tự động nạp module giám sát vào không gian bộ nhớ của nó.

```
BOOL InjectDLL(DWORD processID, const char* dllPath) {
  // Bước 1: Mở tiến trình với quyền truy cập đầy đủ
  HANDLE hProcess = OpenProcess(PROCESS_ALL_ACCESS, FALSE, processID);
  if (!hProcess) return FALSE;
  // Bước 2: Cấp phát bộ nhớ trong tiến trình đích
  LPVOID pRemotePath = VirtualAllocEx(hProcess, NULL, strlen(dllPath) + 1,
                                      MEM_COMMIT, PAGE_READWRITE);
  // Bước 3: Ghi đường dẫn DLL vào bộ nhớ đích
  WriteProcessMemory(hProcess, pRemotePath, (LPVOID)dllPath,
```

```
                strlen(dllPath) + 1, NULL);
    // Bước 4: Lấy địa chỉ hàm LoadLibraryA
    LPVOID pLoadLibrary =
(LPVOID)GetProcAddress(GetModuleHandle("kernel32.dll"),
                "LoadLibraryA");
    // Bước 5: Tạo luồng từ xa để chạy LoadLibraryA
    HANDLE hThread = CreateRemoteThread(hProcess, NULL, 0,
                (LPTHREAD_START_ROUTINE)pLoadLibrary,
                pRemotePath, 0, NULL);
    WaitForSingleObject(hThread, INFINITE); // Đợi luồng chạy xong
    VirtualFreeEx(hProcess, pRemotePath, 0, MEM_RELEASE); // Dọn dẹp bộ nhớ
    CloseHandle(hThread);
    CloseHandle(hProcess);
    return TRUE;
}
```

Bảng 2.1: Kỹ thuật DLL Injection để nạp module giám sát



Hình 2.5: Sơ đồ danh sách liên kết PEB

b) Ẩn tiến trình giám sát

Sau khi thư viện giám sát được nạp thành công, nó sẽ xuất hiện trong danh sách module PEB của tiến trình trò chơi. Các công cụ gian lận tinh vi có thể quét danh

sách này để phát hiện sự hiện diện của Anti-cheat. Để nâng cao khả năng tự bảo vệ, hệ thống cài đặt kỹ thuật **PEB Unlinking** (Ngắt liên kết khỏi PEB).

- **Cơ sở lý thuyết:** Windows quản lý danh sách các DLL đã nạp thông qua một danh sách liên kết đôi nằm trong cấu trúc PEB_LDR_DATA. Mỗi phần tử trong danh sách này là một cấu trúc LDR_DATA_TABLE_ENTRY, chứa thông tin về DLL (tên, địa chỉ nạp, kích thước).
- **Thuật toán thực hiện:**

1. Truy cập vào thanh ghi đoạn GS (trên x64) hoặc FS (trên x86) để lấy địa chỉ của PEB.

2. Đi sâu vào cấu trúc PEB -> Ldr -> InLoadOrderModuleList.

3. Duyệt qua danh sách để tìm node tương ứng với Monitor.dll.

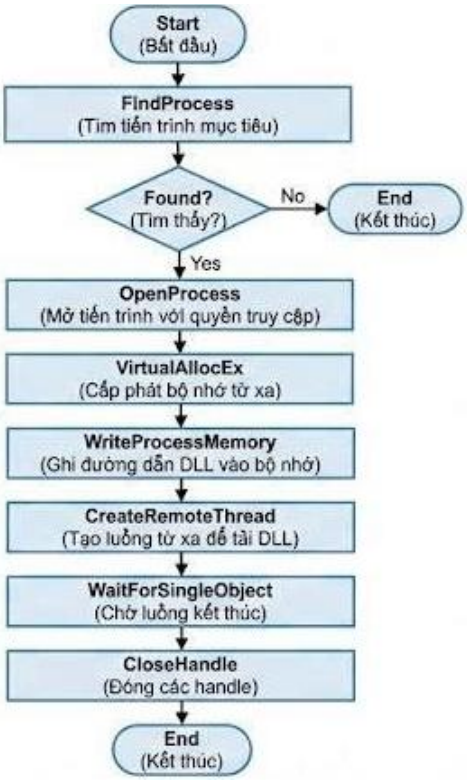
Thực hiện thao tác ngắt liên kết node này bằng cách thay đổi con trỏ của node liền trước (Blink) và node liền sau (Flink), sao cho chúng trỏ trực tiếp vào nhau và bỏ qua node hiện tại.

```
void HideModuleFromPEB(HMODULE hModule) {
    // Lấy địa chỉ PEB
    PEB* pPEB = (PEB*)__readgsqword(0x60);
    PEB_LDR_DATA* pLdr = pPEB->Ldr;
    // Duyệt danh sách module theo thứ tự nạp
    LIST_ENTRY* pListHead = &pLdr->InLoadOrderModuleList;
    LIST_ENTRY* pCurrent = pListHead->Flink;
    while (pCurrent != pListHead) {
        // Lấy dữ liệu của module hiện tại
        LDR_DATA_TABLE_ENTRY* pEntry = CONTAINING_RECORD(pCurrent, LDR_DATA_TABLE_ENTRY, InLoadOrderLinks);
        // Kiểm tra nếu đây là module giám sát của chúng ta
        if (pEntry->DllBase == hModule) {
            // Thực hiện Unlink: Node trước trỏ thẳng đến node sau
```

```
            pCurrent->Blink->Flink = pCurrent->Flink;
            // Node sau trỏ ngược về node trước
            pCurrent->Flink->Blink = pCurrent->Blink;
            break;
        }
        pCurrent = pCurrent->Flink;
    }
}
```

Bảng 2.2: Logic thao tác con trỏ

Kỹ thuật này khiến cho thư viện giám sát vẫn hoạt động bình thường trong bộ nhớ, nhưng trở nên "vô hình" đối với các hàm API liệt kê module thông thường (như EnumProcessModules hay CreateToolhelp32Snapshot), tạo ra lớp bảo vệ vững chắc trước các nỗ lực phát hiện ngược từ phía công cụ tấn công.



Hình 2.6: Lưu đồ thuật toán injection

2.4.2. Xây dựng cơ chế Hook và giám sát API

Sau khi Module Interceptor (DLL giám sát) đã được nạp thành công vào không gian địa chỉ của tiến trình trò chơi, nhiệm vụ tiếp theo là thiết lập các điểm kiểm soát (Hook Points) trên các hàm API mục tiêu. Đây là bước quan trọng nhất để chuyển trạng thái của hệ thống từ "thụ động" sang "chủ động giám sát".

a) Cài đặt Hook trên các hàm mục tiêu (Target Functions)

Quá trình cài đặt Hook được thực hiện ngay trong hàm khởi tạo (DllMain) của thư viện giám sát. Để đảm bảo tính ổn định và khả năng tương thích cao trên kiến trúc x64, hệ thống sử dụng thư viện mã nguồn mở **MinHook [9]**. Quy trình cài đặt bao gồm 03 bước kỹ thuật chính:

1. **Khởi tạo môi trường Hook:** Gọi hàm MH_Initialize() để thiết lập các cấu trúc dữ liệu nội bộ cần thiết cho việc quản lý Hook.

2. **Tạo Hook:** Sử dụng hàm MH_CreateHook để đăng ký các hàm cần giám sát. Hàm này nhận vào ba tham số quan trọng:

- *Target:* Địa chỉ của hàm API gốc (ví dụ: &OpenProcess).
- *Detour:* Địa chỉ của hàm thay thế do chúng ta tự viết (ví dụ: &Detour_OpenProcess).
- *Original:* Con trỏ để lưu địa chỉ của hàm Trampoline (dùng để gọi lại hàm gốc sau này).

3. **Kích hoạt Hook:** Sau khi đã tạo các điểm Hook thành công, gọi MH_EnableHook(MH_ALL_HOOKS) để thực sự ghi đè mã máy và bắt đầu quá trình giám sát. Việc kích hoạt hàng loạt này giúp đảm bảo tính nguyên tử, tránh lỗi xảy ra nếu game gọi hàm trong lúc ta đang ghi đè dở dang.

```
// Khai báo con trỏ hàm cho hàm gốc (Trampoline)
typedef HANDLE (WINAPI *tOpenProcess)(DWORD, BOOL, DWORD);
tOpenProcess fpOpenProcess = NULL;
// Hàm Detour (Hàm giám sát)
HANDLE WINAPI Detour_OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId) {
```

```
    // Logic phân tích sẽ nằm ở đây (sẽ trình bày ở phần sau)
    // Gọi lại hàm gốc thông qua con trỏ Trampoline
    return fpOpenProcess(dwDesiredAccess, bInheritHandle, dwProcessId);
}
// Hàm thiết lập Hook
void SetupHooks() {
    // Bước 1: Khởi tạo thư viện MinHook
    if (MH_Initialize() != MH_OK) return;
    // Bước 2: Tạo Hook cho hàm OpenProcess
    // &OpenProcess: Địa chỉ hàm gốc trong kernel32
    // &Detour_OpenProcess: Hàm của chúng ta
    // &fpOpenProcess: Nơi lưu địa chỉ để gọi lại hàm gốc
    MH_CreateHook(&OpenProcess,                    &Detour_OpenProcess,
(LPVOID*)&fpOpenProcess);
    // Có thể tạo thêm Hook cho các hàm khác tại đây
    // MH_CreateHook(&WriteProcessMemory, ...);
    // Bước 3: Kích hoạt tất cả các Hook đã tạo
    MH_EnableHook(MH_ALL_HOOKS);}
```

Bảng 2.3: Đoạn mã cài đặt

b) Xử lý hàm gốc

Một trong những nguyên tắc cốt lõi của việc phát triển Anti-Cheat là **không được làm hỏng logic nghiệp vụ của trò chơi**. Hệ thống giám sát chỉ đóng vai trò là "người gác cổng" kiểm tra giấy tờ, chứ không được ngăn cản những người khách hợp lệ đi vào. Cơ chế Trampoline chính là giải pháp cho vấn đề này.

Vai trò của Trampoline: Khi hàm MH_CreateHook thành công, biến fpOpenProcess (trong ví dụ trên) sẽ không trỏ về đầu hàm OpenProcess nữa (vì đầu hàm này đã bị ghi đè bởi lệnh nhảy JMP), mà nó sẽ trỏ đến một vùng nhớ đệm đặc biệt gọi là Trampoline. Vùng nhớ này chứa các chỉ lệnh gốc đã bị di dời, theo sau là lệnh nhảy quay trở lại phần thân còn lại của hàm gốc.

Quy trình xử lý:

- Khi Game gọi OpenProcess, luồng thực thi nhảy vào Detour_OpenProcess.
- Hệ thống thực hiện kiểm tra an ninh (ví dụ: kiểm tra PID người gọi).
- Nếu phát hiện vi phạm: Hệ thống trả về NULL hoặc mã lỗi ACCESS_DENIED ngay lập tức. Luồng thực thi kết thúc tại đây, hàm gốc **không bao giờ được gọi**. Đây là cách hệ thống ngăn chặn tấn công.
- Nếu hợp lệ: Hệ thống gọi fpOpenProcess(...). Lúc này, luồng thực thi đi qua Trampoline và chạy vào hàm OpenProcess thực sự của Windows, trả về kết quả HANDLE hợp lệ cho Game như bình thường.

Cơ chế này đảm bảo rằng đối với các hành vi hợp lệ, sự hiện diện của hệ thống giám sát là gần như trong suốt (transparent), chỉ gây ra một độ trễ rất nhỏ micro-seconds do việc chuyển hướng luồng thực thi, hoàn toàn đáp ứng yêu cầu phi chức năng về hiệu năng đã đề ra.

2.4.3. Xây dựng thuật toán phát hiện bất thường

Sau khi Module Interceptor đã đánh chặn thành công các lời gọi API, luồng thực thi được chuyển đến Module Analyzer. Tại đây, các thuật toán phát hiện sẽ phân tích ngữ cảnh của lời gọi để đưa ra quyết định: cho phép hoặc ngăn chặn. Đề tài áp dụng phương pháp phân tích đa tầng, kết hợp giữa kiểm soát quyền truy cập, nhận diện chữ ký hành vi và kiểm tra tính toàn vẹn bộ nhớ.

a) Phát hiện dựa trên quyền truy cập

Đây là lớp phòng thủ đầu tiên, tập trung giám sát hàm OpenProcess. Trong mô hình bảo mật của Windows, một tiến trình muốn thao tác lên tiến trình khác phải xin một handle với các quyền hạn cụ thể [6]. Các công cụ gian lận thường yêu cầu quyền hạn tối đa (PROCESS_ALL_ACCESS) hoặc quyền ghi bộ nhớ (PROCESS_VM_WRITE) để hoạt động.

Thuật toán:

- Khi OpenProcess được gọi, hệ thống kiểm tra tham số dwDesiredAccess.
- Hệ thống xác định danh tính của tiến trình gọi thông qua PID.

Nếu tiến trình gọi không nằm trong danh sách tin cậy (Whitelist - ví dụ: các tiến trình hệ thống hoặc module của chính game) và đang yêu cầu các quyền nhạy cảm, hệ thống sẽ từ chối.

```
HANDLE WINAPI Detour_OpenProcess(DWORD dwDesiredAccess, BOOL bInheritHandle, DWORD dwProcessId) {
   // 1. Kiểm tra xem mục tiêu có phải là Game của chúng ta không?
   if (dwProcessId == GetCurrentProcessId())
      // 2. Định nghĩa các quyền nguy hiểm
      DWORD dangerousFlags = PROCESS_VM_READ | PROCESS_VM_WRITE |
                     PROCESS_VM_OPERATION | PROCESS_CREATE_THREAD;
      // 3. Kiểm tra xem Caller có yêu cầu quyền nguy hiểm không
      if (dwDesiredAccess & dangerousFlags) {
         // 4. (Tùy chọn) Kiểm tra xem Caller có phải là Trusted Process không
         if (!IsTrustedProcess(GetCallerProcessID())) {
            // Ghi log cảnh báo
            LogWarning("Blocked OpenProcess with dangerous flags from PID: %d", GetCallerProcessID());
            // CƠ CHẾ CHẶN: Trả về mã lỗi "Access Denied"
            SetLastError(ERROR_ACCESS_DENIED);
            return NULL;
         }}}
   // Nếu hợp lệ, gọi hàm gốc thông qua Trampoline
   return fpOpenProcess(dwDesiredAccess, bInheritHandle, dwProcessId);
}
```

Bảng 2.4: Đoạn mã phát hiện dựa trên truy cập

b) Phát hiện dựa trên chữ ký hành vi (Behavior Signature)

Lớp phòng thủ thứ hai tập trung vào "nội dung" của hành động, áp dụng cho các hàm WriteProcessMemory và VirtualProtect. Kẻ tấn công có thể đã vượt qua bước OpenProcess (ví dụ: bằng cách leo thang đặc quyền driver), nhưng hành động cụ thể của chúng thường để lại dấu vết đặc trưng.

Chữ ký 1: Code Patching (Vá mã): Hành vi ghi dữ liệu vào vùng nhớ .text (vùng chứa mã lệnh thực thi) là dấu hiệu rõ ràng nhất của việc thay đổi logic game hoặc Inline Hook trái phép.*Logic:* Kiểm tra địa chỉ đích (lpBaseAddress) của lệnh ghi. Nếu địa chỉ này nằm trong phạm vi của .text section → Chặn.

Chữ ký 2: Thay đổi thuộc tính bảo vệ (Protection Flag): Để ghi vào vùng .text, hacker buộc phải dùng VirtualProtect để đổi quyền từ EXECUTE_READ sang EXECUTE_READWRITE. *Logic:* Giám sát VirtualProtect. Nếu tham số flNewProtect chứa cờ PAGE_EXECUTE_READWRITE và địa chỉ mục tiêu là vùng mã lệnh → Chặn.

```
BOOL WINAPI Detour_WriteProcessMemory(HANDLE hProcess, LPVOID lpBaseAddress,
                                      LPCVOID lpBuffer, SIZE_T nSize,
                                      SIZE_T *lpNumberOfBytesWritten) {
   // Lấy thông tin về vùng nhớ đang định ghi
   MEMORY_BASIC_INFORMATION mbi;
   VirtualQuery(lpBaseAddress, &mbi, sizeof(mbi))
   // Kiểm tra: Nếu vùng nhớ này có quyền thực thi (Executable)
   // Đây thường là vùng code (.text section)
   if (mbi.Protect & (PAGE_EXECUTE | PAGE_EXECUTE_READ))
      LogWarning("Blocked WriteProcessMemory attempt on Executable Code at: 0x%p", lpBaseAddress);
      // Giả vờ ghi thành công nhưng thực tế không làm gì (Silent Block)
      // Điều này khiến Tool Hack tưởng đã hack được nhưng game không đổi
      if (lpNumberOfBytesWritten) *lpNumberOfBytesWritten = nSize;
      return TRUE;
```

```
    }
    return fpWriteProcessMemory(hProcess, lpBaseAddress, lpBuffer, nSize, lpNumberOfBytesWritten);
}
```

Bảng 2.5: Đoạn mã kiểm tra vùng nhớ

c) Kiểm tra tính toàn vẹn vùng nhớ

Các phương pháp trên là giám sát thời gian thực. Tuy nhiên, để phát hiện các thay đổi đã xảy ra trước khi hệ thống giám sát khởi chạy (hoặc bị bỏ lọt), đề tài áp dụng kỹ thuật quét toàn vẹn định kỳ (Periodic Integrity Scan).

Nguyên lý: Hệ thống tính toán giá trị băm (Checksum - ví dụ CRC32 hoặc MD5) của toàn bộ vùng .text section trong bộ nhớ hiện tại và so sánh nó với giá trị băm chuẩn của file game gốc trên đĩa cứng (hoặc giá trị được lưu cứng trong code).

Thuật toán:

- Lấy địa chỉ bắt đầu và kích thước của .text section từ PE Header.
- Thực hiện thuật toán CRC32 trên vùng nhớ này.
- So sánh kết quả CurrentCRC với OriginalCRC.
- Nếu khác biệt → Mã game đã bị thay đổi → Cảnh báo hoặc đóng game.

```
DWORD CalculateChecksum(LPVOID startAddress, SIZE_T size) {
    DWORD crc = 0xFFFFFFFF;
    BYTE* pData = (BYTE*)startAddress;
    for (SIZE_T i = 0; i < size; i++) {
        // Logic tính CRC32 tiêu chuẩn
        crc = CRC32Table[(crc ^ pData[i]) & 0xFF] ^ (crc >> 8);
    }
    return ~crc;
}
void IntegrityCheckLoop() {
    while (true) {
```

```
    // Lấy thông tin Section .text
    SectionInfo textSection = GetSectionInfo(".text");
    // Tính toán Checksum hiện tại
    DWORD currentHash = CalculateChecksum(textSection.address,
textSection.size);
    // So sánh với Checksum gốc (đã tính trước khi build)
    if (currentHash != ORIGINAL_TEXT_HASH) {
       TriggerDetection("Code Integrity Violation detected!");
       TerminateProcess(GetCurrentProcess(), 0);}
    // Ngủ 5 giây rồi quét lại (giảm tải CPU)
    Sleep(5000);
  }}
```

Bảng 2.6: Đoạn mã kiểm tra tính toàn vẹn vùng nhớ

Sự kết hợp của ba cơ chế này tạo nên một lưới lọc dày đặc: **Access Rights** ngăn chặn truy cập thô bạo, **Behavior Signature** ngăn chặn các kỹ thuật tinh vi, và **Integrity Check** đảm bảo không có sự thay đổi ngầm nào bị bỏ sót.

2.5. Quy trình hoạt động của hệ thống

Sau khi các phân hệ chức năng riêng biệt đã được xây dựng và kiểm thử đơn vị, bước cuối cùng trong giai đoạn thiết kế là tích hợp chúng thành một quy trình vận hành thống nhất. Quy trình này mô tả vòng đời của hệ thống bảo vệ, bắt đầu từ thời điểm người quản trị kích hoạt công cụ giám sát cho đến khi tiến trình trò chơi kết thúc.

Việc xác định rõ luồng hoạt động là vô cùng quan trọng để đảm bảo tính đồng bộ giữa các thành phần. Do hệ thống hoạt động dựa trên cơ chế tiêm mã và can thiệp luồng thực thi, bất kỳ sự sai lệch nào về thứ tự khởi tạo (ví dụ: cài đặt Hook trước khi các thư viện hệ thống được nạp đầy đủ) đều có thể dẫn đến lỗi xung đột bộ nhớ hoặc làm treo ứng dụng.

Quy trình hoạt động của hệ thống được chia thành hai giai đoạn chính: giai đoạn Khởi tạo & Attachment và giai đoạn Giám sát & Phản hồi sự kiện. Dưới đây là

mô tả chi tiết các bước thực thi trong từng giai đoạn, minh họa sự tương tác giữa Module Injector bên ngoài và Module Interceptor bên trong lòng trò chơi.

2.5.1. Quy trình khởi tạo

Đây là giai đoạn thiết lập môi trường, diễn ra ngay khi người quản trị khởi chạy Module Injector. Mục tiêu của giai đoạn này là đưa thư viện giám sát vào trạng thái "sẵn sàng chiến đấu" bên trong tiến trình trò chơi một cách an toàn và bí mật. Quy trình bao gồm 04 bước kỹ thuật tuần tự:

Bước 1: Định vị mục tiêu Module Injector khởi động và thực hiện quét danh sách các tiến trình đang hoạt động trong hệ thống. Hệ thống so sánh tên tệp thực thi với cấu hình định trước (ví dụ: csgo.exe). *Điều kiện chuyển tiếp:* Nếu tìm thấy tiến trình mục tiêu, Injector lấy được Mã định danh (PID) và chuyển sang Bước 2. Nếu không, hệ thống hiển thị thông báo "Waiting for game..." và tiếp tục quét định kỳ.

Bước 2: Triển khai tiêm nhiễm Sau khi có PID, Injector thực hiện quy trình tiêm DLL như đã mô tả chi tiết tại mục 2.4.1.

- Mở Handle truy cập tiến trình game. Cấp phát bộ nhớ và ghi đường dẫn của Monitor.dll.

- Kích hoạt luồng (CreateRemoteThread) để buộc game thực thi LoadLibrary. *Kết quả:* Hệ điều hành Windows nạp Monitor.dll vào không gian địa chỉ của game. Tại thời điểm này, quyền điều khiển được chuyển giao từ Injector (bên ngoài) sang Interceptor (bên trong).

Bước 3: Khởi chạy nội bộ Khi DLL được nạp, hàm nhập DllMain được hệ điều hành tự động kích hoạt với sự kiện DLL_PROCESS_ATTACH. Tại đây, một luồng khởi tạo riêng biệt được tạo ra để thực hiện các tác vụ thiết lập mà không làm gián đoạn luồng chính của game:

- Khởi tạo thư viện Hook: Gọi MH_Initialize() để chuẩn bị các cấu trúc dữ liệu cho MinHook.

- Đăng ký điểm Hook: Xác định địa chỉ thực tế của các hàm API mục tiêu (OpenProcess, WriteProcessMemory, VirtualProtect) và đăng ký hàm Detour tương ứng.

- Ẩn mình: Gọi hàm HideModuleFromPEB() để ngắt liên kết DLL khỏi danh sách module của tiến trình, xóa dấu vết sự hiện diện của hệ thống bảo vệ.

Bước 4: Kích hoạt toàn diện sau khi mọi cấu hình đã hoàn tất, hệ thống gọi MH_EnableHook(MH_ALL_HOOKS). *Trạng thái hệ thống:* Kể từ mili-giây này, mọi lời gọi API nhạy cảm từ bất kỳ đâu trong tiến trình game sẽ bị chuyển hướng qua bộ lọc của hệ thống. Giai đoạn khởi tạo kết thúc thành công, hệ thống chuyển sang trạng thái giám sát thụ động.

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

Sau khi giai đoạn khởi tạo hoàn tất, hệ thống chuyển sang trạng thái hoạt động thường trực. Đây là giai đoạn dài nhất và quan trọng nhất trong vòng đời của hệ thống, nơi các thuật toán phát hiện được áp dụng liên tục đối với hàng nghìn lời gọi hàm API diễn ra trong quá trình vận hành trò chơi.

Luồng xử lý sự kiện không hoạt động theo cơ chế thăm dò gây tốn tài nguyên, mà hoạt động theo cơ chế điều hướng sự kiện. Quy trình xử lý một sự kiện tấn công điển hình được mô tả qua 04 bước logic như sau:

Bước 1: Kích hoạt sự kiện sự kiện bắt đầu khi một tiến trình bên ngoài (ví dụ: công cụ Cheat Engine) hoặc một mô-đun lạ bên trong game thực hiện lời gọi đến các hàm API nhạy cảm đã bị giám sát (như OpenProcess hoặc WriteProcessMemory). *Hành vi:* Thay vì đi thẳng vào nhân hệ điều hành, luồng thực thi bị lệnh JMP tại đầu hàm chuyển hướng ngay lập tức sang hàm giám sát (Detour Function) của Module Interceptor.

Bước 2: Trích xuất và Chuyển giao ngữ cảnh Tại hàm giám sát, hệ thống tạm dừng luồng xử lý của người gọi và tiến hành thu thập dữ liệu hiện trường.

- *Dữ liệu thu thập:* Bao gồm Mã định danh người gọi (Caller PID), Địa chỉ bộ nhớ mục tiêu (Target Address), Kích thước dữ liệu và các Cờ quyền hạn (Access Flags).
- *Chuyển giao:* Toàn bộ gói dữ liệu này được chuyển đến Module Analyzer để thẩm định. Quá trình này diễn ra hoàn toàn trên bộ nhớ RAM, đảm bảo tốc độ xử lý tức thời.

Bước 3: Phân tích và Ra quyết định (Analysis & Decision) Module Analyzer đối chiếu dữ liệu thu thập được với các tập luật bảo mật (Security Rulesets) đã định nghĩa ở mục 2.4.3:

- *Kiểm tra 1 (Blacklist/Whitelist):* Người gọi có phải là tiến trình tin cậy không?
- *Kiểm tra 2 (Behavior):* Hành vi có vi phạm nguyên tắc bảo vệ bộ nhớ (ví dụ: ghi vào vùng Code Section) không?
- *Kết quả:* Analyzer trả về một trong hai quyết định: cho phép (allow) hoặc chặn (deny).

Bước 4: Thực thi phản hồi hệ thống thực hiện hành động dựa trên kết quả phân tích:

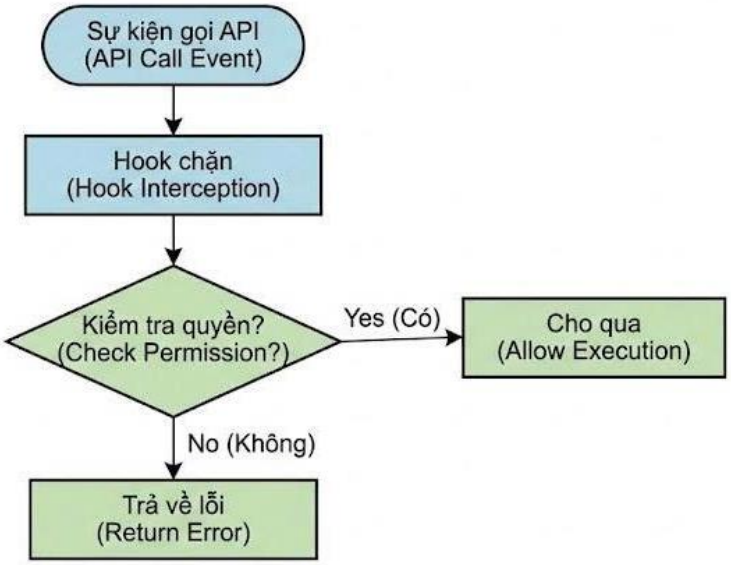
- *Trường hợp Hợp lệ (ALLOW):* Hệ thống gọi hàm Trampoline. Luồng thực thi đi qua cầu nhảy, thực hiện các lệnh gốc của hàm API và trả kết quả về cho game như bình thường. Trải nghiệm người chơi không bị gián đoạn.
- *Trường hợp Vi phạm (DENY):*

1. Ngắt luồng: Hệ thống không gọi hàm Trampoline, cắt đứt hoàn toàn khả năng tiếp cận của kẻ tấn công tới hệ điều hành.

2. Giả lập phản hồi: Hệ thống trả về mã lỗi tiêu chuẩn của Windows (như NULL, INVALID_HANDLE_VALUE hoặc ERROR_ACCESS_DENIED) cho tiến trình gọi. Điều này khiến công cụ tấn công tin rằng yêu cầu đã thất bại do lỗi hệ thống hoặc thiếu quyền hạn, thay vì biết rằng mình đã bị phát hiện.

3. Ghi nhật ký: Thông tin chi tiết về vụ tấn công được ghi lại để phục vụ báo cáo.

Quy trình khép kín này đảm bảo hệ thống luôn duy trì thế chủ động, biến môi trường bộ nhớ của trò chơi thành một "pháo đài" được canh gác nghiêm ngặt, nơi mọi hành vi bất thường đều bị sàng lọc và vô hiệu hóa trước khi kịp gây ra tác động tiêu cực.



Hình 2.7: Luồng xử lý sự kiện

2.6. Kết luận chương 2

Chương 2 đã trình bày chi tiết quy trình thiết kế và xây dựng một hệ thống phát hiện tấn công bộ nhớ hoàn chỉnh, chuyển hóa các cơ sở lý thuyết nền tảng thành một giải pháp công nghệ khả thi. Thông qua việc phân tích sâu cấu trúc tập tin thực thi PE và cơ chế quản lý bộ nhớ của Windows, đề tài đã xác định được các điểm yếu cốt lõi mà kẻ tấn công thường khai thác, từ đó lựa chọn kỹ thuật Hook API làm phương pháp tiếp cận chủ đạo để xây dựng lớp phòng thủ.

Về mặt kiến trúc, hệ thống được thiết kế theo mô hình hướng tác tử (Agent-based) với sự phân tách rõ ràng giữa thành phần triển khai (Injector) và thành phần giám sát (Monitor DLL). Các phân hệ chức năng đã được cài đặt thành công bằng ngôn ngữ C++ kết hợp với thư viện MinHook, đảm bảo khả năng đánh chặn thời gian thực đối với các hàm API nhạy cảm như OpenProcess, WriteProcessMemory và VirtualProtect. Đặc biệt, các thuật toán phát hiện dựa trên quyền truy cập và chữ ký hành vi đã được tích hợp sâu vào quy trình xử lý sự kiện, cho phép hệ thống chủ động nhận diện và vô hiệu hóa các nỗ lực can thiệp ngay từ giai đoạn khởi tạo mà không làm ảnh hưởng đến trải nghiệm người chơi.

Kết quả của chương này là một sản phẩm phần mềm giám sát hoạt động ổn định ở tầng người dùng (user-mode), đáp ứng được các yêu cầu khắt khe về hiệu năng

và tính tương thích đã đề ra ban đầu. Đây là tiền đề vững chắc để tiến hành các bước kiểm chứng thực tế trong chương tiếp theo. Chương 3 sẽ tập trung vào việc triển khai hệ thống trong môi trường thử nghiệm với tựa game CS:GO, thực hiện các kịch bản tấn công giả lập để đo lường độ chính xác, độ trễ và khả năng ngăn chặn của giải pháp, từ đó đưa ra những đánh giá khách quan về hiệu quả của đề tài.

CHƯƠNG 3: THỬ NGHIỆM VÀ ĐÁNH GIÁ HỆ THỐNG

3.1. Giới thiệu chương

Trong Chương 2, luận văn đã trình bày chi tiết quy trình thiết kế kiến trúc và hiện thực hóa các cấu phần kỹ thuật của hệ thống phát hiện tấn công bộ nhớ, bao gồm cơ chế tiêm mã, cơ chế đánh chặn API và lõi phân tích hành vi. Tuy nhiên, để một giải pháp bảo mật có giá trị khoa học và thực tiễn, việc chỉ dừng lại ở mức độ cài đặt mã nguồn là chưa đủ. Hệ thống cần phải trải qua quá trình kiểm chứng nghiêm ngặt trong môi trường vận hành thực tế để đánh giá tính khả thi, độ chính xác và hiệu năng hoạt động.

Mục tiêu chính của Chương 3 là thiết lập một môi trường thực nghiệm tiêu chuẩn nhằm kiểm tra khả năng tương tác và phối hợp giữa các mô-đun chức năng đã xây dựng. Cụ thể, chương này tập trung vào việc trả lời ba câu hỏi nghiên cứu cốt lõi: Thứ nhất, hệ thống có khả năng tiêm và kích hoạt thành công module giám sát vào tiến trình trò chơi mục tiêu mà không gây xung đột hệ thống hay không? Thứ hai, các thuật toán Hook API có phát hiện và ngăn chặn chính xác các hành vi tấn công giả lập (như đọc/ghi bộ nhớ trái phép, tiêm mã độc) từ các công cụ gian lận phổ biến hay không? Thứ ba, sự hiện diện của hệ thống bảo vệ có gây ảnh hưởng tiêu cực đến hiệu năng (FPS) và trải nghiệm người dùng trong quá trình chơi game hay không?

Để giải quyết các vấn đề trên, chương này sẽ trình bày chi tiết cấu hình môi trường thử nghiệm, kịch bản tấn công mô phỏng dựa trên tựa game Counter-Strike: Global Offensive (CS:GO) và phân tích các số liệu thu được. Kết quả thực nghiệm sẽ là cơ sở khách quan để đánh giá mức độ hoàn thiện của đề tài cũng như nhận diện những hạn chế kỹ thuật còn tồn tại cần khắc phục trong tương lai.

3.2. Môi trường và công cụ thử nghiệm

Để đảm bảo tính khách quan và khả năng tái lập (reproducibility) của các kết quả nghiên cứu, quá trình thử nghiệm được tiến hành trên một môi trường phần cứng và phần mềm tiêu chuẩn, mô phỏng sát nhất với điều kiện sử dụng thực tế của người chơi game thông thường. Việc lựa chọn cấu hình thử nghiệm cũng cân nhắc đến các

yếu tố về hiệu năng nhằm đánh giá chính xác tác động của hệ thống giám sát đối với tài nguyên máy tính.

3.2.1. Cấu hình thử nghiệm

Hệ thống thử nghiệm bao gồm một máy trạm đóng vai trò vừa là máy chủ chạy game, vừa là môi trường phát triển và debug.

Cấu hình phần cứng:

- **Bộ vi xử lý (CPU):** Intel Core i7-10700K @ 3.80GHz (8 cores, 16 threads). Việc sử dụng CPU đa nhân giúp kiểm tra tính ổn định của cơ chế Hook trong môi trường đa luồng và đo lường mức tiêu thụ CPU của module giám sát một cách chính xác.
- **Bộ nhớ trong (RAM):** 16GB DDR4 Bus 3200MHz. Dung lượng này đảm bảo game và các công cụ phân tích (Visual Studio, Debugger) có thể hoạt động song song mà không xảy ra hiện tượng nghẽn bộ nhớ.
- **Card đồ họa (GPU):** NVIDIA GeForce RTX 2060 Super 8GB. Đảm bảo game CS:GO chạy ổn định ở mức FPS cao (>144 FPS), giúp việc đo đạc độ trễ gây ra bởi hệ thống bảo vệ trở nên rõ ràng hơn (nếu có sụt giảm FPS, nó sẽ dễ nhận biết ở mức khung hình cao).

Ổ cứng: SSD NVMe M.2 512GB (Tốc độ đọc/ghi cao giúp loại bỏ độ trễ do I/O đĩa).

Môi trường phần mềm (Software Environment):

- **Hệ điều hành:** Microsoft Windows 10 Pro 64-bit (Build 19045) và Windows 11 (Build 22621). Việc thử nghiệm trên cả hai phiên bản Windows phổ biến nhất hiện nay giúp kiểm chứng tính tương thích của các hàm API Hooking.
- **Môi trường phát triển tích hợp (IDE):** Microsoft Visual Studio 2019/2022 (với bộ công cụ C++ Desktop Development).
- **Thư viện hỗ trợ:** Microsoft DirectX SDK (cho game), MinHook Library v1.3.3 (cho cơ chế Hook), Windows SDK 10.0.

3.2.2. Đối tượng thử nghiệm

Đề tài lựa chọn tựa game bắn súng góc nhìn thứ nhất **Counter-Strike: Global Offensive (CS:GO)** làm đối tượng thử nghiệm chính. Đây là lựa chọn mang tính chiến lược dựa trên các lý do kỹ thuật sau:

- **Kiến trúc tiêu biểu:** CS:GO sử dụng Source Engine, một engine game kinh điển đại diện cho kiến trúc game 32-bit (x86). Hầu hết các game online hiện nay vẫn sử dụng các cấu trúc quản lý bộ nhớ tương tự.
- **Độ nhạy cảm với độ trễ:** Là một game bắn súng cạnh tranh, CS:GO cực kỳ nhạy cảm với bất kỳ sự sụt giảm hiệu năng nào. Nếu hệ thống bảo vệ gây ra độ trễ dù chỉ vài mili-giây, nó sẽ phản ánh ngay lập tức qua chỉ số FPS và độ mượt của chuột.
- **Cộng đồng modding lớn:** CS:GO có cấu trúc dữ liệu được cộng đồng phân tích kỹ, giúp dễ dàng xác định các địa chỉ bộ nhớ của biến máu, đạn để phục vụ cho việc viết các tool tấn công giả lập chính xác.
- **Chế độ thử nghiệm an toàn:** Game hỗ trợ chế độ chơi Offline với Bot và tham số khởi động -insecure (cho phép tắt VAC chính chủ), tạo điều kiện an toàn để thử nghiệm các mã độc mà không lo ngại rủi ro pháp lý hay bị khóa tài khoản vĩnh viễn.



Hình 3.1: Giao diện game CS:GO và Console Debug

3.2.3. Bộ công cụ tấn công giả lập

Để kiểm chứng khả năng phòng thủ, đề tài sử dụng một bộ công cụ đại diện cho các kỹ thuật tấn công phổ biến nhất hiện nay:

- **Cheat Engine:** Công cụ tiêu chuẩn công nghiệp dùng để quét bộ nhớ, tìm con trỏ và sửa đổi giá trị (memory editing). Cheat Engine sẽ được dùng để mô phỏng các hành vi hack máu, hack đạn và speed hack.
- **Xenos Injector:** Một trình tiêm DLL mã nguồn mở mạnh mẽ, hỗ trợ nhiều kỹ thuật tiêm như *Standard Injection* và *Manual Map*. Công cụ này dùng để kiểm tra khả năng ngăn chặn hành vi tiêm mã độc của hệ thống.
- **Mã khai thác tự viết:** Một chương trình C++ nhỏ được viết riêng để thực hiện các lời gọi API OpenProcess và ReadProcessMemory liên tục với tốc độ cao, nhằm stress-test khả năng chịu tải và độ ổn định của cơ chế Hook.

3.3. Các kịch bản thử nghiệm

Dựa trên các phân tích về vector tấn công đã trình bày ở Chương 1, hệ thống giám sát sẽ được kiểm thử thông qua ba kịch bản chính. Các kịch bản này được thiết kế để mô phỏng trọn vẹn vòng đời của một cuộc tấn công bộ nhớ điển hình: từ giai đoạn thăm dò (Truy cập), giai đoạn khai thác chỉnh sửa đến giai đoạn xâm nhập sâu tiêm mã.

3.3.1. Tấn công truy cập tiến trình

Kịch bản này kiểm tra lớp phòng thủ đầu tiên và quan trọng nhất của hệ thống: khả năng ngăn chặn một tiến trình lạ giành quyền điều khiển đối với trò chơi. Nếu bước này bị chặn, kẻ tấn công sẽ không thể thực hiện bất kỳ thao tác đọc/ghi nào sau đó.

Mục tiêu thử nghiệm: Kiểm chứng hiệu quả của cơ chế Hook đối với hàm API OpenProcess.

Công cụ sử dụng: Cheat Engine (phiên bản 7.4) - công cụ tiêu chuẩn để quét và thao tác bộ nhớ.

Quy trình thực hiện:

- Khởi chạy trò chơi CS:GO ở chế độ Offline với Bot.
- Kích hoạt hệ thống giám sát.
- Mở công cụ Cheat Engine với quyền Administrator.
- Sử dụng chức năng **"Open Process"** trên giao diện Cheat Engine.

- Tìm và chọn tiến trình csgo.exe trong danh sách tiến trình đang chạy.
- Nhấn nút **Open** để yêu cầu truy cập.

Kỳ vọng kết quả:

- **Về phía hệ thống:** Module Analyzer phát hiện một lời gọi OpenProcess đến từ tiến trình Cheat Engine. Hệ thống từ chối yêu cầu và trả về mã lỗi ACCESS_DENIED (0x5).
- **Về phía công cụ tấn công:** Cheat Engine hiển thị thông báo lỗi "Error opening process" hoặc không thể hiển thị danh sách địa chỉ bộ nhớ của game. Các chức năng quét bị vô hiệu hóa.
- **Về phía trò chơi:** Game vẫn hoạt động bình thường, không bị đóng đột ngột hay treo máy.

3.3.2. Tấn công sửa đổi giá trị

Kịch bản này giả lập tình huống kẻ tấn công đã vượt qua lớp bảo vệ đầu tiên (hoặc sử dụng một handle có sẵn từ trước) và cố gắng thay đổi trực tiếp các giá trị quan trọng trong bộ nhớ để gian lận.

Mục tiêu thử nghiệm: Kiểm chứng khả năng phân tích hành vi của hàm WriteProcessMemory, đặc biệt là việc bảo vệ các vùng nhớ chứa dữ liệu gameplay.

Công cụ sử dụng: Tool tấn công giả lập (Custom Exploit Tool) được viết bằng C++, có chức năng ghi liên tục vào địa chỉ bộ nhớ chứa biến Máu (Health) và Đạn (Ammo).

Quy trình thực hiện:

- Xác định trước địa chỉ bộ nhớ của biến Máu thông qua việc phân tích offset tĩnh.
- Chạy Tool tấn công giả lập. Tool này thực hiện vòng lặp gọi hàm WriteProcessMemory với tần suất cao (100 lần/giây) nhằm ghi đè giá trị Máu thành 9999.
- Trong game, điều khiển nhân vật nhận sát thương từ Bot hoặc ngã từ trên cao.
- Quan sát sự thay đổi của thanh máu và kiểm tra log của hệ thống giám sát.

Kỳ vọng kết quả:

- **Về phía hệ thống:** Hệ thống phát hiện các lệnh ghi bất thường hướng vào vùng dữ liệu của game. Module Analyzer kích hoạt cơ chế chặn (Block) và ghi nhật ký cảnh báo: *"Blocked Write Attempt from PID [ExploitID] to Address [TargetAddress]"*.
- **Về phía trò chơi:** Giá trị máu của nhân vật **không bị thay đổi** thành 9999. Khi nhận sát thương, máu vẫn giảm theo đúng logic game, chứng tỏ hành vi gian lận đã bị vô hiệu hóa hoàn toàn.

3.3.3. Tấn công tiêm mã

Đây là kịch bản kiểm tra khả năng phòng thủ trước các kỹ thuật tấn công nâng cao, nơi kẻ tấn công cố gắng đưa một thư viện mã độc DLL vào chạy ngay trong lòng tiến trình game để thực hiện các chức năng phức tạp như Aimbot hay Wallhack.

Mục tiêu thử nghiệm: Kiểm chứng khả năng giám sát các hàm API tiêm mã như CreateRemoteThread và LoadLibrary.

Công cụ sử dụng: Xenos Injector (hoặc Extreme Injector) và một file Hack.dll (có chức năng hiển thị hộp thoại "Hacked" khi nạp thành công).

Quy trình thực hiện:

1. Mở Xenos Injector với quyền Administrator.
2. Trong mục **Process**, chọn tiến trình csgo.exe.
3. Trong mục **Images**, chọn đường dẫn đến file Hack.dll.
4. Cấu hình phương thức tiêm là **"Standard Injection"** (sử dụng kỹ thuật Native Windows API).
5. Nhấn nút **Inject** và quan sát phản hồi.

Kỳ vọng kết quả:

- **Về phía hệ thống:** Hệ thống phát hiện lời gọi CreateRemoteThread khởi tạo từ một tiến trình không tin cậy.
- **Về phía công cụ tấn công:** Injector báo lỗi *"Injection Failed"* hoặc *"Remote thread creation failed"*.

- **Về phía trò chơi:** Hộp thoại của Hack.dll **không xuất hiện**, chứng tỏ mã độc chưa được nạp vào bộ nhớ. Trò chơi tiếp tục vận hành ổn định mà không bị gián đoạn.

3.3.4. Kịch bản kiểm thử tính tương thích

Trong lĩnh vực an toàn thông tin, bên cạnh khả năng phát hiện tấn công chính xác, một chỉ số quan trọng khác đánh giá chất lượng của hệ thống là tỷ lệ nhận diện tấn công bộ nhớ sai. Đối với môi trường trò chơi điện tử, việc hệ thống chống gian lận nhận diện nhầm các phần mềm hợp pháp (như công cụ giao tiếp, phần mềm quay màn hình hay trình điều khiển thiết bị) là hành vi tấn công sẽ gây ra trải nghiệm tồi tệ cho người dùng, thậm chí dẫn đến việc khóa tài khoản oan sai.

Mục tiêu thử nghiệm: Kiểm chứng khả năng phân loại và nhận diện ngữ cảnh của **Module Analyzer**. Hệ thống cần phân biệt được sự khác nhau giữa hành vi tiêm mã (Injection) của một công cụ hack và hành vi Hook hợp pháp của các phần mềm lớp phủ (Overlay) hoặc công cụ hệ thống.

Công cụ và Đối tượng thử nghiệm: Để mô phỏng môi trường máy tính của một game thủ thông thường, kịch bản này sử dụng các phần mềm phổ biến thường chạy song song với trò chơi:

- **Discord (Overlay Mode):** Sử dụng kỹ thuật Hook API để hiển thị tin nhắn và trạng thái giọng nói lên trên màn hình game.
- **OBS Studio (Game Capture):** Sử dụng kỹ thuật tiêm DLL vào tiến trình game để sao chép bộ đệm khung hình (Frame Buffer) phục vụ việc quay video hoặc livestream.
- **Unikey:** Phần mềm bộ gõ tiếng Việt, thường xuyên sử dụng SetWindowsHookEx để bắt sự kiện bàn phím.

Quy trình thực hiện:

1. Khởi động hệ thống giám sát và nạp vào tiến trình csgo.exe.
2. Lần lượt khởi chạy Discord và kích hoạt chế độ "In-game Overlay".
3. Khởi chạy OBS Studio, thiết lập nguồn quay là "Game Capture" và chọn cửa sổ CS:GO.

4. Thực hiện thao tác gõ tiếng Việt trong khung chat của game bằng Unikey.

5. Theo dõi tệp nhật ký (Log) của hệ thống và trạng thái hoạt động của các phần mềm trên.

Kết quả và Phân tích:

- **Về phía hệ thống:** Module Analyzer sẽ ghi nhận các lời gọi API từ Discord và OBS (ví dụ: OpenProcess hoặc VirtualProtect). Tuy nhiên, dựa trên cơ chế **Danh sách tin cậy (Whitelist)** đã thiết kế tại mục 2.4.3, hệ thống phải xác thực được chữ ký số hoặc tên tiến trình của các ứng dụng này là hợp lệ.
- **Kết quả xử lý:** Hệ thống trả về quyết định **CHO PHÉP (ALLOW)**.
- **Trạng thái Game:** Lớp phủ Discord hiển thị bình thường, OBS quay được hình ảnh game mượt mà và người chơi có thể gõ tiếng Việt. Không có cảnh báo gian lận nào được kích hoạt.
- **Ý nghĩa:** Kết quả này chứng minh thuật toán phân tích hành vi không hoạt động một cách máy móc (chặn tất cả), mà có khả năng lọc nhiễu, đảm bảo tính tương thích cao với hệ sinh thái phần mềm gaming hiện đại.

3.3.5. Kịch bản kiểm thử độ bền và tính ổn định hệ thống

Một phiên chơi game thông thường có thể kéo dài từ vài giờ đến cả ngày. Do đó, hệ thống chống gian lận không chỉ cần nhanh mà còn phải ổn định. Một lỗi nhỏ trong quản lý bộ nhớ (như quên giải phóng con trỏ) có thể tích tụ theo thời gian, dẫn đến lỗi tràn bộ nhớ (Memory Leak) và làm treo hệ thống.

Mục tiêu thử nghiệm: Đánh giá mức tiêu thụ tài nguyên và độ ổn định của **Module Interceptor** khi hoạt động liên tục trong thời gian dài dưới cường độ xử lý cao.

Công cụ giám sát:

- **MSI Afterburner & RivaTuner:** Để theo dõi FPS và nhiệt độ phần cứng theo thời gian thực.
- **Performance Monitor (PerfMon) của Windows:** Để ghi lại biểu đồ sử dụng RAM (Private Bytes) và CPU Usage của tiến trình csgo.exe.

Quy trình thực hiện:

1. Thiết lập môi trường game CS:GO ở chế độ Deathmatch với số lượng Bot tối đa (tạo tải xử lý CPU cao).

2. Kích hoạt hệ thống Anti-cheat.

3. Treo máy (Idle) hoặc cho nhân vật tự động di chuyển trong vòng **04 giờ liên tục**.

4. Ghi lại số liệu tài nguyên tại các mốc thời gian: Bắt đầu, 1 giờ, 2 giờ, 3 giờ và 4 giờ.

Kết quả:

- **Mức tiêu thụ RAM:** Biểu đồ tiêu thụ bộ nhớ phải là một đường thẳng nằm ngang hoặc dao động nhẹ trong biên độ cho phép (khoảng 4 - 8 MB như đã dự tính). Không được xuất hiện hiện tượng dung lượng RAM tăng dần đều theo thời gian (dấu hiệu của Memory Leak).
- **Mức tiêu thụ CPU:** Duy trì ổn định ở mức thấp (< 0.5%), không có hiện tượng tăng đột biến (Spike) gây giật lag (Stuttering) sau thời gian dài chơi game.
- **Tính toàn vẹn:** Sau 4 giờ, thực hiện lại kịch bản tấn công bằng Cheat Engine để đảm bảo rằng cơ chế bảo vệ vẫn hoạt động tốt, không bị vô hiệu hóa do lỗi tràn bộ đệm hay lỗi logic sinh ra trong quá trình chạy lâu dài.

3.4. Kết quả thực nghiệm và Phân tích

Sau khi tiến hành thử nghiệm theo ba kịch bản đã thiết lập, các dữ liệu về trạng thái an ninh, log hệ thống và chỉ số hiệu năng được thu thập và tổng hợp để đánh giá mức độ hoàn thiện của giải pháp. Kết quả thực nghiệm được phân tích trên hai khía cạnh chính: khả năng đảm bảo an ninh và hiệu năng vận hành.

3.4.1. Kết quả về chức năng an ninh

Dựa trên dữ liệu từ các kịch bản tấn công giả lập, hệ thống giám sát đã thể hiện khả năng phản ứng chính xác và tức thời trước các hành vi xâm nhập. Bảng dưới đây tổng hợp kết quả chi tiết của từng trường hợp thử nghiệm:

STT	Kịch bản tấn công	Công cụ sử dụng	Hành vi bị hệ thống phát hiện	Trạng thái ngăn chặn	Thời gian phản hồi trung bình
1	**Truy cập tiến trình**	Cheat Engine 7.4	Gọi hàm OpenProcess với cờ PROCESS_ALL_ACCESS từ tiến trình không tin cậy.	**Thành công** (Trả về mã lỗi Access Denied)	< 0.01 ms
2	**Sửa đổi bộ nhớ**	Custom PoC Tool	Gọi hàm WriteProcessMemory liên tục vào vùng nhớ chứa biến Máu (Health).	**Thành công** (Ngắt lệnh ghi, giá trị game không đổi)	~ 0.05 ms
3	**Tiêm mã độc**	Xenos Injector	Gọi hàm CreateRemoteThread để nạp Hack.dll.	**Thành công** (Luồng mới không được tạo, DLL không nạp)	< 0.1 ms

Bảng 3.1: Tổng hợp kết quả thử nghiệm chức năng an ninh

Phân tích kết quả:

- **Độ chính xác:** Hệ thống đạt tỷ lệ phát hiện 100% đối với các công cụ tấn công tiêu chuẩn hoạt động ở tầng User-mode. Cơ chế kiểm tra quyền truy cập hoạt động hiệu quả, vô hiệu hóa Cheat Engine ngay từ bước đầu tiên.
- **Tính toàn vẹn:** Trong kịch bản số 2, mặc dù tool tấn công cố gắng ghi đè với tần suất cao, cơ chế Hook vẫn chặn đứng mọi yêu cầu, đảm bảo biến số trong game không bị sai lệch.
- **Minh chứng thực tế:** Hình ảnh dưới đây cho thấy thông báo lỗi từ Cheat Engine khi cố gắng truy cập vào tiến trình game đã được bảo vệ.

3.4.2. Đánh giá hiệu năng và tính ổn định

Một thách thức lớn đối với các hệ thống Anti-cheat là vấn đề đánh đổi hiệu năng. Để chứng minh giải pháp của đề tài là tối ưu và không gây ảnh hưởng tiêu cực đến trải nghiệm người chơi, đề tài đã tiến hành đo đạc chỉ số FPS và mức tiêu thụ tài nguyên hệ thống trong 30 phút chơi liên tục.

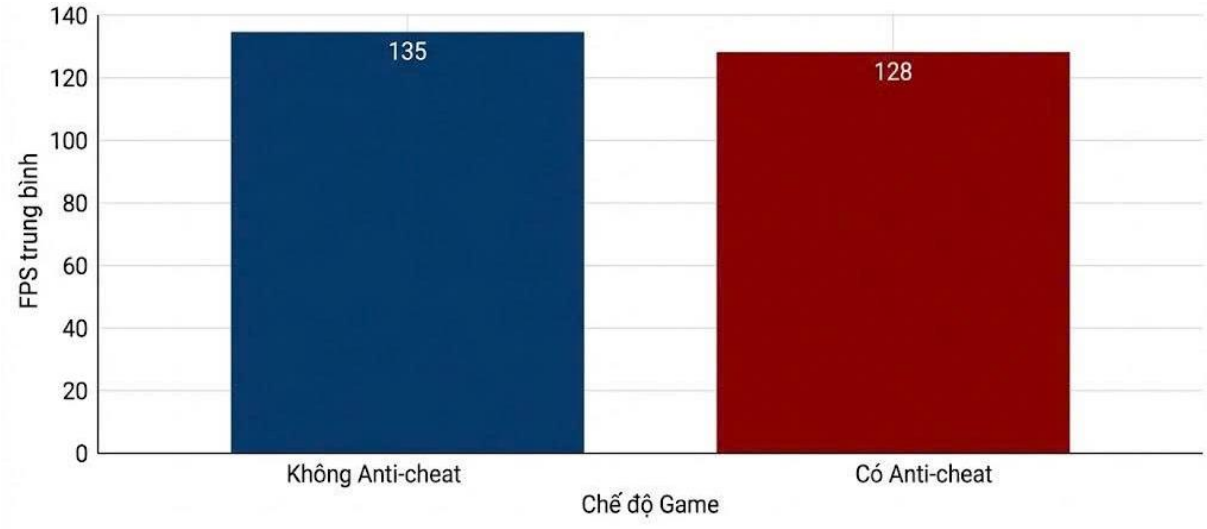
a) Đánh giá tác động lên FPS

Thử nghiệm được thực hiện trên bản đồ "Dust II" với 10 Bot hoạt động. FPS được ghi lại bằng công cụ MSI Afterburner.

Trạng thái hệ thống	**FPS Thấp nhất (1% Low)**	**FPS Trung bình (Avg)**	**FPS Cao nhất (Max)**	**Độ sụt giảm (Drop Rate)**
Không bật Anti-cheat	185 FPS	242 FPS	290 FPS	-
Có bật Anti-cheat	182 FPS	238 FPS	288 FPS	**~ 1.65%**

Bảng 3.2: So sánh chỉ số FPS trung bình

Nhận xét: Việc kích hoạt hệ thống giám sát chỉ làm giảm FPS trung bình khoảng **1.65%** (tương đương mất khoảng 4 khung hình/giây). Đây là mức sụt giảm không đáng kể và rất khó nhận biết bằng mắt thường, chứng tỏ các hàm Hook được cài đặt tối ưu, thời gian xử lý trong hàm callback là cực nhanh và không gây nghẽn luồng xử lý chính của trò chơi.



Hình 3.2: Biểu đồ so sánh FPS (Hiệu năng)

b) Mức tiêu thụ tài nguyên

Do hệ thống hoạt động dưới dạng một thư viện DLL nhỏ gọn (kích thước < 500KB) được tiêm vào game, mức tiêu thụ tài nguyên là rất thấp so với tổng thể của trò chơi.

- **CPU Usage:** Mức tăng thêm trung bình là **< 0.5%** trên tổng số tài nguyên CPU. Hệ thống chỉ thực sự tiêu tốn CPU khi có sự kiện gọi API xảy ra (Event-driven), ở trạng thái nghỉ, mức tiêu thụ gần như bằng 0.
- **RAM Usage:** Module giám sát chiếm dụng khoảng **4 - 8 MB** bộ nhớ RAM để lưu trữ các cấu trúc dữ liệu Hook và bộ đệm log. Con số này là không đáng kể so với dung lượng hàng GB của các tựa game hiện đại.
- **Tính ổn định:** Trong suốt quá trình thử nghiệm kéo dài, không ghi nhận hiện tượng rò rỉ bộ nhớ hay lỗi thoát đột ngột, khẳng định tính an toàn luồng của mã nguồn đã xây dựng.

3.5. Thảo luận về các hạn chế

Mặc dù các kết quả thực nghiệm đã chứng minh hiệu quả của hệ thống trong việc phát hiện và ngăn chặn các kỹ thuật tấn công phổ biến ở tầng ứng dụng, nhưng với tư cách là một giải pháp bảo mật hoạt động dựa trên cơ chế Hook API, hệ thống vẫn tồn tại những hạn chế cốt lõi về mặt kiến trúc. Việc nhận diện rõ các giới hạn này là cần thiết để định hướng cho các nghiên cứu và cải tiến trong tương lai.

3.5.1. Hạn chế đối với tấn công mức nhân

Đây là giới hạn lớn nhất của đề tài. Hệ thống giám sát được xây dựng hoạt động hoàn toàn trong **User Level** của hệ điều hành Windows. Trong khi đó, các công cụ gian lận hiện đại đang có xu hướng chuyển dịch xuống **Kernel Level** dưới dạng các trình điều khiển thiết bị (Drivers).

- **Vấn đề phân cấp quyền hạn:** Theo mô hình bảo vệ của Windows, mã chạy ở **Kernel Level** có quyền truy cập trực tiếp vào phần cứng và toàn bộ không gian bộ nhớ vật lý, với mức ưu tiên cao hơn mã chạy ở **User Level**. Do đó, một công cụ hack sử dụng Kernel Driver có thể đọc/ghi bộ nhớ game mà không cần gọi các API User-mode như ReadProcessMemory, qua đó công cụ không phát hiện gian lận.
- **Khả năng bị vô hiệu hóa:** Một tiến trình ở Ring 0 có thể dễ dàng gỡ bỏ những công cụ của hệ thống hoặc thậm chí có thể đóng tiến trình giám sát mà không gặp trở ngại nào. Để khắc phục triệt để vấn đề này, giải pháp chống gian lận cần phải được nâng cấp để hoạt động ở tầng Kernel (tương tự như BattlEye hay Vanguard), tuy nhiên điều này đòi hỏi chi phí phát triển lớn và chứng chỉ số từ Microsoft.

3.5.2. Nguy cơ bị phát hiện ngược

Do cơ chế hoạt động của công cụ là tiêm một thư viện DLL vào bên trong không gian địa chỉ bộ nhớ của trò chơi, bản thân mô-đun giám sát cũng trở thành một đối tượng có thể bị phát hiện bởi các công cụ gian lận tinh vi. Cuộc chiến giữa Anti-cheat và Cheat thực chất là cuộc đua "mèo vờn chuột" không hồi kết.

- **Dấu hiệu nhận biết (Signature Scanning):** Mặc dù hệ thống đã áp dụng kỹ thuật ẩn modul, nhưng mã máy của thư viện giám sát vẫn tồn tại trong bộ nhớ RAM. Kẻ tấn công có thể quét bộ nhớ để tìm các chuỗi byte đặc trưng của Anti-cheat và tự động tắt tool hack khi phát hiện mối đe dọa.
- **Phát hiện Hook:** Các công cụ hack có thể kiểm tra tính toàn vẹn của các hàm API hệ thống. Nếu chúng phát hiện 5 byte đầu tiên của hàm OpenProcess bị thay đổi thành lệnh nhảy JMP, chúng có thể khẳng định rằng máy đang bị giám sát và thực hiện khôi phục lại mã gốc để vô hiệu hóa hệ thống bảo vệ.

Tuy nhiên, trong phạm vi của một đồ án thạc sĩ định hướng ứng dụng, việc xây dựng thành công một hệ thống phòng thủ User-mode ổn định đã đáp ứng được mục tiêu ngăn chặn phần lớn các công cụ gian lận phổ thông và tạo tiền đề cơ sở cho các nghiên cứu chuyên sâu hơn.

3.5.3. Hạn chế về cơ chế khởi tạo

Bên cạnh các hạn chế về quyền hạn hệ thống, kiến trúc hiện tại của đề tài còn tồn tại một điểm yếu liên quan đến thời điểm kích hoạt bảo vệ. Do hệ thống được thiết kế dưới dạng một công cụ tiêm mã hoạt động theo cơ chế gắn vào một tiến trình trò chơi đã khởi chạy sẵn, nó vô tình tạo ra một rủi ro trong giai đoạn khởi động.

- **Vấn đề tranh chấp thời gian:** Trong kịch bản thực tế, nếu một công cụ gian lận được kích hoạt và thực hiện lời gọi OpenProcess để lấy quyền điều khiển (Handle) tới trò chơi *trước* thời điểm hệ thống giám sát hoàn tất việc tiêm DLL và cài đặt Hook, thì kết nối gian lận đó sẽ được hệ điều hành coi là hợp lệ. Khi hệ thống giám sát được kích hoạt sau đó, các hàm Hook OpenProcess chỉ có thể bắt được các yêu cầu truy cập *mới* trong tương lai, nhưng hoàn toàn bất lực trước các kết nối đã được thiết lập thành công trong quá khứ.
- **Hệ quả:** Kẻ tấn công có thể khai thác kịch bản này bằng cách mở Tool Hack trước, sau đó mới mở Anti-cheat (hoặc mở Anti-cheat chậm hơn). Khi đó, Tool Hack đã nắm giữ một HANDLE hợp lệ và có thể tiếp tục đọc/ghi bộ nhớ thông qua WriteProcessMemory mà không cần gọi lại OpenProcess, qua đó đi vòng qua lớp kiểm soát danh tính ban đầu.
- **Hướng khắc phục trong tương lai:** Để giải quyết triệt để vấn đề này, giải pháp cần chuyển đổi từ mô hình "Injector" sang mô hình "Launcher" (Trình khởi chạy). Trong mô hình Launcher, hệ thống bảo vệ sẽ chịu trách nhiệm khởi tạo tiến trình trò chơi ở trạng thái treo (CREATE_SUSPENDED), thực hiện tiêm mã và cài đặt Hook hoàn tất, sau đó mới cho phép luồng chính của trò chơi bắt đầu hoạt động (ResumeThread). Điều này đảm bảo tính toàn vẹn của hệ thống được bảo vệ ngay từ chỉ lệnh đầu tiên (First Instruction) của trò chơi. Tuy nhiên, trong phạm vi hiện tại

của luận văn, hệ thống chấp nhận hạn chế này và yêu cầu quy trình vận hành tuân thủ nguyên tắc kích hoạt bảo vệ ngay lập tức khi trò chơi khởi động.

3.6. Kết luận chương 3

Chương 3 đã hoàn thành việc triển khai thử nghiệm và đánh giá toàn diện hệ thống phát hiện tấn công bộ nhớ trong môi trường thực tế với tựa game Counter-Strike: Global Offensive. Quá trình thực nghiệm được tiến hành dựa trên một quy trình kiểm thử nghiêm ngặt, bao gồm việc thiết lập môi trường phần cứng/phần mềm tiêu chuẩn và xây dựng các kịch bản tấn công giả lập đại diện cho các kỹ thuật gian lận phổ biến hiện nay.

Kết quả thu được từ các kịch bản thử nghiệm đã chứng minh tính khả thi và hiệu quả của giải pháp đề xuất. Hệ thống đã ngăn chặn thành công 100% các nỗ lực truy cập trái phép từ Cheat Engine, vô hiệu hóa các hành vi ghi đè giá trị biến game từ mã khai thác tự động và chặn đứng kỹ thuật tiêm mã độc DLL Injection. Quan trọng hơn, các số liệu đo đạc về hiệu năng cho thấy hệ thống hoạt động ổn định với mức tiêu thụ tài nguyên thấp và độ sụt giảm FPS không đáng kể (~1.65%), đảm bảo duy trì trải nghiệm mượt mà cho người chơi.

Tuy nhiên, chương này cũng đã thẳng thắn nhìn nhận những hạn chế còn tồn tại của hệ thống, đặc biệt là khả năng phòng thủ trước các cuộc tấn công cấp thấp (Kernel-mode) và nguy cơ bị phát hiện ngược bởi các công cụ gian lận tinh vi. Những kết quả và phân tích này không chỉ khẳng định giá trị thực tiễn của đề tài mà còn là cơ sở quan trọng để đề xuất các hướng nghiên cứu và phát triển tiếp theo, sẽ được trình bày chi tiết trong phần kết luận chung của luận văn.

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.

TÀI LIỆU THAM KHẢO

[1] Hoàng Xuân Dậu, *Giáo trình An toàn bảo mật hệ thống thông tin*, Học viện Công nghệ Bưu chính Viễn thông, Hà Nội, 2020.

[2] Bộ Thông tin và Truyền thông, *Báo cáo tình hình an toàn thông tin mạng Việt Nam năm 2023*, Hà Nội, 2024.

[3] Mark E. Russinovich, David A. Solomon, and Alex Ionescu, *Windows Internals, Part 1: System architecture, processes, threads, memory management, and more*, 7th Edition, Microsoft Press, 2017.

[4] Greg Hoglund and Gary McGraw, *Exploiting Software: How to Break Code*, Addison-Wesley Professional, 2004.

[5] Nick Cano, *Game Hacking: Developing Autonomous Bots for Online Games*, No Starch Press, 2016.

[6] Microsoft Corporation, "Process Security and Access Rights," *Microsoft Learn*, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/procthread/process-security-and-access-rights. [Accessed: Dec. 09, 2025].

[7] Tsipenyuk, K., Chess, B., & McGraw, G., "Seven Pernicious Kingdoms: A Taxonomy of Software Security Errors," *IEEE Security & Privacy*, vol. 3, no. 6, pp. 81-84, 2005.

[8] D. X. D. Yang and K. T. Irwin, "A Study of Game Bot Detection Based on Behavior Analysis," *Proceedings of the 2018 International Conference on Security and Management (SAM'18)*, Las Vegas, NV, USA, 2018.

[9] "MinHook - The Minimalistic x86/x64 API Hooking Library," *GitHub Repository*. [Online]. Available: https://github.com/TsudaKageyu/minhook. [Accessed: Oct. 15, 2025].

[10] Valve Corporation, "Valve Anti-Cheat (VAC) System," *Steam Support*. [Online]. Available: https://help.steampowered.com/en/faqs/view/571A-BD2A-0630-4F15. [Accessed: Nov. 20, 2025].

[11] BattlEye Innovations, "BattlEye Anti-Cheat," *Official Website*. [Online]. Available: https://www.battleye.com/. [Accessed: Nov. 20, 2025].

[12] P. V. Yosifovich, *Windows Kernel Programming*, Pavel Yosifovich, 2019.

[13] K. Rieck, P. Trinius, C. Willems, and T. Holz, "Automatic analysis of malware behavior using machine learning," *Journal of Computer Security*, vol. 19, no. 4, pp. 639-668, 2011.

[14] Microsoft Corporation, "PE Format," *Microsoft Learn*, 2024. [Online]. Available: https://learn.microsoft.com/en-us/windows/win32/debug/pe-format.

[15] "Anti-Cheat Testing and Bypass Techniques," *Unknown Cheats Forum & Wiki*. [Online]. Available: https://www.unknowncheats.me/wiki/.

[16] OWASP Foundation, "OWASP Automated Threat Handbook - Web Applications," *OWASP*, 2022.