**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**



VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

-------------------------------------------



**BÁO CÁO BÀI TẬP LỚN**

**PROJECT 2**

**Đề tài: XÂY DỰNG CHƯƠNG TRÌNH CHẶN BẮT GÓI TIN ĐỂ SÀNG LỌC NHỮNG THÔNG TIN ĐỘC HẠI**

**Giảng viên: Đỗ Quốc Huy**

|  |  |
| --- | --- |
| Họ và tên: | Kiều Minh Hướng |
| MSSV: | 20183928 |
| Lớp: | CNTT 03 – K63 |

Hà Nội, 2020

# **LỜI CẢM ƠN**

Đối với sinh viên năm 3 khoa Công nghệ thông tin, những project thực tế đóng một vai trò cực kì quan trọng trong việc tạo cho em nền tảng tốt để có thể theo đuổi chuyên ngành mình mong ước. Trong quá trình làm project này, em đã nhận được sự chỉ bảo tận tình của thầy Đỗ Quốc Huy. Điều đó khiến em cảm thấy vô cùng vui mừng.

Sau khi thầy ra đề bài “***Xây dựng chương trình chặn bắt gói tin để sàng lọc những thông tin độc hại***”, em cũng đã cố gắng hết sức để hoàn thành project này. Tuy nhiên, do kiến thức cũng như kinh nghiệm thực tế, tư duy còn nhiều hạn chế, giới hạn trong phạm vi kiến thức của sinh viên nên không thể tránh khỏi những thiếu sót. Rất mong nhận được sự quan tâm và đóng góp ý kiến của thầy để project này được hoàn thiện hơn.

Em xin chân thành cảm ơn!

# **MỤC LỤC**

[**LỜI MỞ ĐẦU** 1](#_Toc76603361)

[**CHƯƠNG I: HỆ THỐNG THU THẬP GÓI TIN** 4](#_Toc76603362)

[***1.1. Phân tích cơ chế hoạt động của việc trao đổi gói tin giữa các socket*** 4](#_Toc76603363)

[***1.2. Thư viện Winsock*** 5](#_Toc76603364)

[***1.3. Giới thiệu về hệ thống thu thập gói tin*** 6](#_Toc76603365)

[***1.4. Cấu trúc tiêu đề gói tin*** 6](#_Toc76603366)

[***1.5. Bắt đầu chương trình sử dụng winsock*** 8](#_Toc76603367)

[***1.6. Hệ thống bắt và hiển thị gói tin*** 12](#_Toc76603368)

[**CHƯƠNG II: HỆ THỐNG CHẶN TRUY CẬP** 19](#_Toc76603369)

[***2.1. Filter Driver*** 19](#_Toc76603370)

[***2.2. Network Filter Driver*** 20](#_Toc76603371)

[***2.3. Windows Filtering Platform*** 21](#_Toc76603372)

[**KẾT LUẬN** 31](#_Toc76603373)

[**TÀI LIỆU THAM KHẢO** 32](#_Toc76603374)

# **LỜI MỞ ĐẦU**

**1. Lý do chọn đề tài**

Trong những năm gần đây, những cuộc tấn công mạng đang diễn ra liên tục và có chiều hướng ngày một nhiều hơn. Làn sóng tấn công mạng bằng mã độc (malware) gia tăng trên toàn cầu đã phơi bày tình trạng “dễ bị tổn thương” của các doanh nghiệp, khiến vấn đề này trở thành một trong những ưu tiên cần được quan tâm đúng mức. Các vấn đề về an toàn, an ninh mạng trở thành mối quan tâm chính của các nền kinh tế dựa trên công nghệ ngày nay. An toàn, an ninh mạng đã trở thành nhu cầu cốt lõi để cung cấp một xã hội bền vững và an toàn cho người dùng trực tuyến trong không gian mạng.

Trong bảng xếp hạng những mục tiêu chính của hacker, hệ điều hành Windows dường như chưa bao giờ rời khỏi top. Điều đó xảy ra do khả năng bảo mật của Windows không mạnh như những hệ điều hành nhân Linux. Hơn nữa, những kỹ thuật khai thác trên Windows là dễ dàng thực hiện hơn so với Linux.

Tuy nhiên, dù dưới bất kỳ hình thức nào, những cuộc tấn công mạng cũng phải thực hiện được khi kẻ tấn công đã kết nối tới máy tính của nạn nhân. Khi kết nối, kẻ tấn công sẽ gửi những gói tin chứa thông tin độc hại hoặc rò rỉ thông tin nhạy cảm của nạn nhân qua môi trường mạng.

Nhận thấy việc tìm hiểu sâu hơn về kiến trúc TCP/IP, hệ thống Windows cũng như từ đó nâng cao nhận thức về bảo mật hệ thống là cực kỳ cần thiết, em đã thực hiện đề tài cho project của mình là “***Xây dựng chương trình chặn bắt gói tin để sàng lọc những thông tin độc hại***”. Hi vọng project này sẽ được nhận sự đánh giá tích cực của thầy Đỗ Quốc Huy.

**2. Tổng quan về đề tài**

Microsoft Windows (hoặc đơn giản là Windows) là tên của một họ hệ điều hành dựa trên giao diện người dùng đồ hoạ được phát triển và được phân phối bởi tập đoàn Microsoft. Nó bao gồm một vài các dòng hệ điều hành, mỗi trong số đó phục vụ một phần nhất định của ngành công nghiệp máy tính. Ngày nay, Windows 10 là hệ điều hành được sử dụng phổ biến nhất trên thế giới. Và Microsoft cũng đã giới thiệu Windows 11, có thể coi là hệ điều hành của tương lai.

Tuy nhiên, vấn đề bảo mật trên Windows luôn là vấn đề nhức nhối nhất đối với các nhà phát triển hệ điều hành này. Hàng năm, có hàng chục CVE liên quan đến hệ điều hành Windows được công bố, hầu hết trong số chúng đều là các CVE có điểm CVSS cao, được đánh giá ở mức độ nghiêm trọng hoặc đặc biệt nghiêm trọng, chủ yếu liên quan đến các lỗ hổng leo thang đặc quyền, thực thi mã từ xa,…

Ngoài ra, mã độc thực thi được trên Windows cũng ngày càng đa dạng và phát triển mạnh, trong khi Windows Defender dường như không thể đáp ứng được yêu cầu trong việc ngăn chặn các loại mã độc này. Những loại mã độc đã phát triển từ rất lâu như Keylogger vẫn còn tồn tại cho đến ngày nay với vô vàn biến thể, cùng với đó là những Backdoor để kẻ tấn công điều khiển và tạo ra hệ thống botnet. Ẩn sau những công nghệ ấy là Rootkit để che đậy sự hiện diện của chúng. Và phát triển mạnh nhất trong vài năm trở lại đây là Ransomware – mã độc tống tiền đã và đang gây thiệt hại tới hàng chục tỉ đô trên toàn thế giới.

Những vấn đề đó là bằng chứng cho thấy chúng ta cần một giải pháp tốt để ngăn chặn những mối nguy hại trên môi trường mạng. Việc xây dựng hệ thống phát hiện và ngăn chặn xâm nhập đã được nghiên cứu tại rất nhiều nơi trên thế giới. Những sản phẩm nổi bật là hệ thống Phát hiện và Ngăn chặn xâm nhập – IDS IPS hay những phần mềm Antivirus.

Tuy nhiên tại Việt Nam, việc chú trọng vào ngăn chặn mối nguy hại còn chưa được quan tâm, hầu hết các doanh nghiệp không có biện pháp bảo vệ mình trên môi trường mạng, dẫn tới trở thành miếng mồi ngon cho hacker. Vì thế, em đã thực hiện đề tài này để giúp những doanh nghiệp nâng cao nhận thức và từ đó biết cách bảo vệ mình để tránh thiệt hại về tài sản và pháp luật.

**3. Mục đích nghiên cứu**

Em nghiên cứu đề tài này nhằm củng cố lại kiến thức mình đã học được trong những môn học khác, đồng thời trau dồi khả năng, tư duy lập trình cũng như tìm hiểu thêm về một số hàm can thiệp sâu vào phần cứng và hệ điều hành. Từ đó cũng đóng góp một phần sức lực của mình vào việc ngăn chặn các mối nguy cơ trên mạng.

**4. Phạm vi nghiên cứu**

Đề tài này tập trung nghiên cứu các hàm trong thư viện Winsock và thư viện liên quan đến driver trong Windows.

**5. Đóng góp của đề tài**

Với việc thực hiện đề tài này, em hi vọng rằng sẽ đóng góp được một chút kiến thức của mình vào kho tài liệu về việc nghiên cứu các giải pháp bảo mật mạng.

**6. Kết cấu bài báo cáo**

Ngoài lời cảm ơn, lời mở đầu, mục lục, kết luận và tài liệu tham khảo, phần chính báo cáo được chia làm 2 phần tương ứng với 2 module trong hệ thống:

Chương 1: Hệ thống thu thập gói tin

Chương 2: Hệ thống chặn truy cập

# **CHƯƠNG I: HỆ THỐNG THU THẬP GÓI TIN**

## ***1.1. Phân tích cơ chế hoạt động của việc trao đổi gói tin giữa các socket***

Socket là giao diện lập trình ứng dụng mạng được dùng để truyền và nhận dữ liệu trên internet. Giữa hai chương trình chạy trên mạng cần có một liên kết giao tiếp hai chiều, hay còn gọi là two-way communication để kết nối 2 process trò chuyện với nhau. Điểm cuối (endpoint) của liên kết này được gọi là socket.

Một chức năng khác của socket là giúp các tầng TCP định danh ứng dụng mà dữ liệu sẽ được gửi tới thông qua sự ràng buộc với một cổng port (thể hiện là một con số cụ thể), từ đó tiến hành kết nối giữa client và server.

Socket hoạt động trên cả giao thức TCP và UDP. Như đã nói ở trên thì có thể xem Socket = Địa chỉ IP + Số Port, port ở đây chính là port logic của máy tính(16 bits = 65535 ports).

Diagram

Description automatically generated

Ví dụ quá trình khởi tạo kết nối tới Gmail sẽ diễn ra như sau:

Client có địa chỉ IP1 đang có port 5000 rảnh rỗi và quyết định sử dụng cặp (IP, Port) = (IP1, 5000) để kết nối tới web server có địa chỉ IP2 và port 80 (để chạy giao thức HTTP => lấy về giao diện trang web)

Sau khi client được server xác thực thành công và đã có đủ thông tin cần thiết, nó sẽ mở cổng số 25 cho địa chỉ IP2. Client không hề gửi request đi mà chỉ mở port 25 cho web server.

Khi có email mới, server sẽ kiểm tra xem kết nối tới (IP1, 25) có còn sống hay không, nếu có thì nó sẽ gửi thông báo về cho client.

Có 3 loại socket: Datagram socket là dùng cho giao thức UDP; Stream socket dùng cho giao thức TCP; Websocket dùng cho truyền thông trên web.

## ***1.2. Thư viện Winsock***

Windows Sockets API (WSA), sau rút gọn thành Winsock, là một đặc tả kỹ thuật xác định cách phần mềm mạng trong Windows truy cập các dịch vụ mạng, đặc biệt là TCP / IP. Winsock định nghĩa giao diện chuẩn giữa ứng dụng (ví dụ chương trình FTP client hoặc trình duyệt web) và ngăn xếp giao thức TCP / IP bên dưới. Winsock API được đặt trong file winsock.dll (16 bit) hoặc wsock32.dll (32 bit) trong thư mục hệ thống.

Winsock là bộ API tiêu chuẩn để lập trình mạng trong windows. Tuy nhiên, việc lập trình ứng dụng với winsock có những khó khăn nhất định.

Thứ nhất, các API hệ thống thường rất phức tạp với rất nhiều tham số gây khó khăn cho việc lập trình. Để đảm bảo tính linh hoạt, mỗi API đều chứa rất nhiều tham số, sử dụng nhiều kiểu dữ liệu hỗ trợ, cũng như có rất nhiều loại “magic constant”. Lập trình với socket API cũng không ngoại lệ.

Thư hai, việc gọi đến các API của hệ thống thường chỉ phù hợp khi lập trình với một số ngôn ngữ và công nghệ nhất định. Ví dụ khi sử dụng C/C++/Delphi xây dựng ứng dụng native cho windows sẽ dễ dàng truy cập các API này hơn. Tuy nhiên, sử dụng các ngôn ngữ và công cụ bậc “không cao” như C/C++ làm tăng thời gian phát triển ứng dụng (giảm năng suất).

Các công nghệ phát triển ứng dụng hiện đại thường hạn chế việc truy xuất trực tiếp đến các API của hệ thống. Thay vào đó, các công nghệ này thường tạo ra các “vỏ bọc” (wrapper) để giúp người lập trình gọi đến các API của hệ thống một cách dễ dàng hơn.

## ***1.3. Giới thiệu về hệ thống thu thập gói tin***

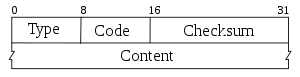
Hệ thống thu thập gói tin là một hệ thống được xây dựng dựa trên các Winsock API, với ý tưởng chính là sẽ tìm các giao diện mạng trên máy tính, chọn giao diện mạng chạy ở chế độ quảng bá để thu thập các gói tin gửi tới, rồi phân tích ra nội dung của gói tin đó.

Toàn bộ mã nguồn gồm 3 phần: 1 file header để khai bác cấu trúc của tiêu đề gói tin; 1 file header xây dựng các hàm để giải mã và in nội dung gói tin; 1 file chứa hàm main để chạy chương trình

## ***1.4. Cấu trúc tiêu đề gói tin***

Trước tiên, như chúng ta đã biết, do hệ thống network sử dụng phương thức lưu trữ dữ liệu theo dạng big-endian, đây là dạng lưu trữ byte có ý nghĩa cao hơn nằm ở trước. Tuy nhiên trong máy tính ngày nay thường sử dụng phương thức lưu trữ little-endian, vì thế thứ tự lưu trữ dữ liệu và thứ tự khai báo các biến đại diện cho các dữ liệu ấy phải ngược lại so với những gì chúng ta được học về tiêu đề của các gói tin.

Một gói tin có phần tiêu đề (header) sẽ chứa các thông tin tùy thuộc vào loại gói tin. Ví dụ như gói tin ICMP thì ICMP header sẽ chứa các thông tin như sau



Trong đó 8 bit đầu là ICMP Type, 8 bit tiếp theo là ICMP Subtype, 16 bit tiếp theo là checksum để kiểm tra lỗi gói tin. Trong 32 bit của trường nội dung thì 16 bit đầu là ID của gói tin, 16 bit sau là sequence number của gói tin.

Khi ta khai báo cấu trúc ICMP, do phải lưu trữ ngược lại nên cấu trúc ICMP header sẽ như sau:

typedef struct icmp\_hdr {

BYTE type; // ICMP Error type

BYTE code; // Type sub code

USHORT checksum;

USHORT id;

USHORT seq;

} ICMP\_HDR;

Tương tự, em sẽ mô phỏng cấu trúc IP header, TCP header và UDP header – những gói tin phổ biến và cần thiết nhất để chúng ta có thể xác định rõ ràng nội dung của gói tin.

IP header sẽ có khai báo như sau:

typedef struct ip\_hdr {

unsigned char ip\_header\_len : 4; // 4-bit header length (in 32-bit words)

unsigned char ip\_version : 4; // 4-bit IPv4 version

unsigned char ip\_tos; // IP type of service

unsigned short ip\_total\_length; // Total length

unsigned short ip\_id; // Unique identifier

unsigned char ip\_frag\_offset : 5; // Fragment offset field

unsigned char ip\_more\_fragment : 1;

unsigned char ip\_dont\_fragment : 1;

unsigned char ip\_reserved\_zero : 1;

unsigned char ip\_frag\_offset1; //fragment offset

unsigned char ip\_ttl; // Time to live

unsigned char ip\_protocol; // Protocol(TCP,UDP etc)

unsigned short ip\_checksum; // IP checksum

unsigned int ip\_srcaddr; // Source address

unsigned int ip\_destaddr; // Source address

} IPV4\_HDR;

UDP header sẽ có khai báo:

typedef struct udp\_hdr {

unsigned short source\_port; // Source port no.

unsigned short dest\_port; // Dest. port no.

unsigned short udp\_length; // Udp packet length

unsigned short udp\_checksum; // Udp checksum (optional)

} UDP\_HDR;

TCP header sẽ có khai báo:

typedef struct tcp\_header {

unsigned short source\_port; // source port

unsigned short dest\_port; // destination port

unsigned int sequence; // sequence number - 32 bits

unsigned int acknowledge; // acknowledgement number - 32 bits

unsigned char ns : 1; //Nonce Sum Flag Added in RFC 3540.

unsigned char reserved\_part1 : 3; //according to rfc

unsigned char data\_offset : 4; /\*The number of 32-bit words in the TCP header. This indicates where the data begins. The length of the TCP header is always a multiple of 32 bits.\*/

unsigned char fin : 1; //Finish Flag

unsigned char syn : 1; //Synchronise Flag

unsigned char rst : 1; //Reset Flag

unsigned char psh : 1; //Push Flag

unsigned char ack : 1; //Acknowledgement Flag

unsigned char urg : 1; //Urgent Flag

unsigned char ecn : 1; //ECN-Echo Flag

unsigned char cwr : 1; //Congestion Window Reduced Flag

unsigned short window; // window

unsigned short checksum; // checksum

unsigned short urgent\_pointer; // urgent pointer

} TCP\_HDR;

Ta có thể thấy cấu trúc của gói tin TCP là rất phức tạp, vì tính chất đáng tin cậy của nó. Những cờ điều khiển đều chỉ có 1 bit, vì thế chúng ta phải ép cho chúng chỉ chiếm 1 bit giá trị dù chúng thuộc kiểu char 8 bit.

## ***1.5. Bắt đầu chương trình sử dụng winsock***

Để bắt đầu chương trình, chúng ta sẽ sử dụng một template thường thấy của các chương trình sử dụng winsock. Đó là: Khởi tạo winsock với hàm WSAStartup(); tạo socket; gắn socket với địa chỉ IP của máy với hàm bind(); thực hiện các chức năng cần thiết và cuối cùng là đóng socket, kết thúc winsock với hàm WSACleanup()

Mở đầu chương trình, chúng ta sẽ khởi tạo winsock version 2.2 và kiểm tra lỗi khi khởi tạo với các câu lệnh:

WSADATA wsaData;

printf("\nInitialising Winsock...");

if (WSAStartup(MAKEWORD(2, 2), &wsaData)) {

printf("Cannot initiate winsock with error %d\n", WSAGetLastError());

return 1;

}

Bây giờ chúng ta sẽ thực hiện khởi tạo socket. Chúng ta cần bắt các gói tin và dựa vào header để phân biệt gói tin. Để bắt được các gói tin và đọc được các header của chúng, ta buộc phải sử dụng một loại socket đặc biệt là “raw socket”. Những socket hoạt động ở tầng giao vận là Datagram socket và Stream socket không thể giúp ta đọc được các thông tin xuất hiện ở tầng mạng, vì thế raw socket – loại socket hoạt động ở tầng mạng là điều bắt buộc phải sử dụng trong chương trình. Lưu ý, vì raw socket hoạt động ở tầng dưới nên để khởi tạo phải chạy dưới quyền admin. Khởi tạo raw socket bằng các câu lệnh:

SOCKET sniffer;

//Create a RAW Socket

sniffer = socket(AF\_INET, SOCK\_RAW, IPPROTO\_IP);

if (sniffer == INVALID\_SOCKET) {

printf("Cannot create raw socket with error %d.\n", WSAGetLastError());

return 1;

}

Giá trị SOCK\_RAW là đại diện cho raw socket. Hàm socket với các tham số như trên đại diện cho việc socket sử dụng IPv4 của giao thức IP.

Sau khi khởi tạo socket, ta cần gắn socket với địa chỉ IP của máy. Trước tiên, ta cần phải tìm ra IP của máy. Mỗi máy tính có thể có nhiều giao diện mạng: Card wifi, mạng Lan, Gateway của máy ảo Virtual box, Vmware,… Vì vậy ta sẽ lấy toàn bộ IP của giao diện mạng rồi chọn giao diện mạng cần lắng nghe các kết nối. Để làm được điều này, ta sẽ phải sử dụng hàm gethostbyname() của mạng local để lấy được danh sách các IP. Code cụ thể như sau:

int choosenInterface;

char hostname[100];

struct hostent \*local;

//Retrive the local host name

if (gethostname(hostname, sizeof(hostname)) == SOCKET\_ERROR) {

printf("Cannot get host name with error %d\n", WSAGetLastError());

return 1;

}

printf("\nHost name: %s \n", hostname);

//Get available IP of host

local = gethostbyname(hostname);

if (local == NULL) {

printf("Cannot get host by name with error %d.\n", WSAGetLastError());

return 1;

}

printf("\nAvailable Network Interfaces:\n");

for (int i = 0; local->h\_addr\_list[i] != 0; ++i) {

memcpy(&addr, local->h\_addr\_list[i], sizeof(IN\_ADDR));

printf("Interface Number %d: Address: %s\n", i, inet\_ntoa(addr));

}

printf("Enter the interface number you would like to sniff: ");

scanf("%d", &choosenInterface);

Cấu trúc hostent sử dụng để lưu trữ thông tin về máy chủ nhất định, chẳng hạn như tên máy chủ, địa chỉ IPv4, v.v… Trước tiên ta sẽ lấy tên của máy chủ bằng hàm gethostname() để gán vào biến hostname. Giá trị hostname lúc này sẽ là tên của máy tính. Sau đó ta sử dụng hàm gethostbyname() để lấy danh sách các host đang chạy trên máy tính. Tất cả các host lưu trong 1 danh sách liên kết là h\_addr\_list, sử dụng một vòng lặp để liệt kê toàn bộ host, rồi nhận đầu vào là giao diện mạng mà ta chọn. Sau khi chọn giao diện mạng, ta bắt đầu sử dụng hàm bind() để gắn giao diện vào địa chỉ IP của giao diện vừa chọn:

// Bind socket with address

memset(&dest, 0, sizeof(dest));

memcpy(&dest.sin\_addr.s\_addr, local->h\_addr\_list[choosenInterface], sizeof(dest.sin\_addr.s\_addr));

dest.sin\_family = AF\_INET;

dest.sin\_port = 0;

if (bind(sniffer, (struct sockaddr \*)&dest, sizeof(dest)) == SOCKET\_ERROR) {

printf("Cannot bind to %s with error %d.\n", inet\_ntoa(addr), WSAGetLastError());

return 1;

}

printf("Bind successful");

Đây là một template của hàm bind() cơ bản nên hoàn toàn dễ hiểu.

Sau khi thực hiện hàm bind(), việc tiếp theo ta cần thực hiện là chuyển socket sang chế độ sniffer mode (chế độ lắng nghe và bắt toàn bộ các gói tin) bằng hàm WSAIoctl():

// Setting sniffer mode

int j = 1;

if (WSAIoctl(sniffer, SIO\_RCVALL, &j, sizeof(j), 0, 0, (LPDWORD)&choosenInterface, 0, 0) == SOCKET\_ERROR) { // Enable sniffing with macro SIO\_RCVALL

printf("Cannot sniffing with error %d.\n", WSAGetLastError());

return 1;

}

Trong hàm WSAIoctl có các tham số như sau:

int WSAAPI WSAIoctl(

SOCKET s, // Socket

DWORD dwIoControlCode, // Socket mode

LPVOID lpvInBuffer, // Input buffer

DWORD cbInBuffer, // Size of input buffer

LPVOID lpvOutBuffer, // output buffer

DWORD cbOutBuffer, // size of output buffer

LPDWORD lpcbBytesReturned, // size of real output byte

LPWSAOVERLAPPED lpOverlapped, // Not use

LPWSAOVERLAPPED\_COMPLETION\_ROUTINE lpCompletionRoutine // Not use

);

Chế độ chọn ở đây là chế độ SIO\_RCVALL cho phép một socket nhận toàn bộ các gói tin IPv4 đi qua giao diện mạng. Tiếp theo là chọn chế độ riêng cho SIO\_RCVALL thông qua input buffer. Input buffer có giá trị 1, đại diện cho chế độ RCVALL\_ON receive all: on. Với việc sử dụng chế độ SIO\_RCVALL, output buffer sẽ không cần sử dụng. Giá trị đầu ra thực tế sẽ là một con trỏ đến một biến nhận kích thước, tính bằng byte, của dữ liệu được lưu trữ trong bộ đệm đầu ra, tận dụng biến choosenInterface. Sau khi cài xong chế độ, ta bắt đầu hàm Sniffing:

//Begin

printf("\nStarted Sniffing\n");

printf("Packet Capture Statistics...\n");

StartSniffing(sniffer);

// End

## ***1.6. Hệ thống bắt và hiển thị gói tin***

Hàm Sniffing sẽ bắt đầu bằng việc nhận các data từ socket sniffer phía trên:

void StartSniffing(SOCKET sniffer) {

char \*buff = (char \*)malloc(65536);

int receivedByte;

if (buff == NULL) {

printf("Cannot dynamic allocate memory\n");

return;

}

do {

receivedByte = recvfrom(sniffer, buff, 65536, 0, 0, 0); //Receive as much as buffer can

if (receivedByte > 0) {

// Classify packet received and write to log

ClassifyPacket(buff, receivedByte);

}

else {

printf("Cannot receive data with error %d.\n", WSAGetLastError());

}

} while (receivedByte > 0);

free(buff);

}

Trước tiên, ta dành ra một phần bộ nhớ để lưu trữ dữ liệu nhận được, và kiểm tra xem nó có thể khởi tạo không. Ta sẽ sử dụng hàm malloc. Tối đa sẽ nhận được 64KB dữ liệu.

Hàm recvfrom sẽ cho phép chương trình nhận nhiều data nhất mà buffer này có thể nhận và nhận đến khi không còn dữ liệu để nhận. Nếu số byte thực nhận lớn hơn 0 sẽ tiến hành phân loại loại gói tin dựa vào header. Tiếp theo ta sẽ phân tích hàm ClassifyPacket:

int tcp = 0, udp = 0, icmp = 0, others = 0, igmp = 0, total = 0;

void ClassifyPacket(char\* Buffer, int Size) {

ip\_header = (IPV4\_HDR \*)Buffer;

total++;

switch (ip\_header->ip\_protocol) { //Check the Protocol

case 1: //ICMP Protocol

icmp++;

PrintIcmpPacket(Buffer, Size);

break;

case 2: //IGMP Protocol

igmp++;

break;

case 6: //TCP Protocol

tcp++;

PrintTcpPacket(Buffer, Size);

break;

case 17: //UDP Protocol

udp++;

PrintUdpPacket(Buffer, Size);

break;

default: //Other Protocol

others++;

break;

}

printf("TCP:%d UDP:%d ICMP:%d IGMP:%d Others:%d Total:%d\r", tcp, udp, icmp, igmp, others, total);

}

Mỗi khi 1 gói tin gửi đến buffer thông qua socket sniffer sẽ sử dụng hàm ClassifyPacket(buff, size) để phân loại, dựa trên IP header mà nó nhận được. IP header thu được thông qua buffer sẽ được kiểm tra giá trị ip\_protocol để kiểm tra xem đó là loại gói tin nào. Tương ứng với các giá trị ip\_protocol sẽ là các gói tin TCP, UDP, ICMP, IGMP hoặc các loại khác. Sau khi đã phân loại được, tiến hành in ra nội dung của gói tin đó. Giới hạn trong chương trình này thì chỉ in ra gói tin TCP, UDP hoặc ICMP. Ví dụ chương trình in ra gói tin UDP:

void PrintUdpPacket(char \*Buffer, int Size) {

unsigned short ip\_header\_len;

ip\_header = (IPV4\_HDR \*)Buffer;

ip\_header\_len = ip\_header->ip\_header\_len \* 4;

udp\_header = (UDP\_HDR \*)(Buffer + ip\_header\_len);

fprintf(logSniffer, "\n\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UDP Packet\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

Printip\_header(Buffer);

fprintf(logSniffer, "\nUDP Header\n");

fprintf(logSniffer, " |-Source Port : %d\n", ntohs(udp\_header->source\_port));

fprintf(logSniffer, " |-Destination Port : %d\n", ntohs(udp\_header->dest\_port));

fprintf(logSniffer, " |-UDP Length : %d\n", ntohs(udp\_header->udp\_length));

fprintf(logSniffer, " |-UDP Checksum : %d\n", ntohs(udp\_header->udp\_checksum));

fprintf(logSniffer, "\nIP Header\n");

PrintData(Buffer, ip\_header\_len);

fprintf(logSniffer, "UDP Header\n");

PrintData(Buffer + ip\_header\_len, sizeof(UDP\_HDR));

fprintf(logSniffer, "Data Payload\n");

PrintData(Buffer + ip\_header\_len + sizeof(UDP\_HDR), (Size - sizeof(UDP\_HDR) - ip\_header->ip\_header\_len \* 4));

fprintf(logSniffer, "\n###########################################################");

}

Trong hàm PrintUdpPacket thực hiện 3 việc: 1 là in ra thông tin của IP header và UDP header, 2 là in ra nội dung của gói tin, 3 là lưu gói tin đó vào file log.

Việc in ra nội dung của IP header thực hiện thông qua 1 hàm Printip\_header:

void Printip\_header(char\* Buffer) {

unsigned short ip\_header\_len;

ip\_header = (IPV4\_HDR \*)Buffer;

ip\_header\_len = ip\_header->ip\_header\_len \* 4;

memset(&source, 0, sizeof(source));

source.sin\_addr.s\_addr = ip\_header->ip\_srcaddr;

memset(&dest, 0, sizeof(dest));

dest.sin\_addr.s\_addr = ip\_header->ip\_destaddr;

fprintf(logSniffer, "\nIP Header\n");

fprintf(logSniffer, " |-IP Version : %d\n", (unsigned int)ip\_header->ip\_version);

fprintf(logSniffer, " |-IP Header Length : %d DWORDS or %d Bytes\n", (unsigned int)ip\_header->ip\_header\_len, ((unsigned int)(ip\_header->ip\_header\_len)) \* 4);

fprintf(logSniffer, " |-Type Of Service : %d\n", (unsigned int)ip\_header->ip\_tos);

fprintf(logSniffer, " |-IP Total Length : %d Bytes(Size of Packet)\n", ntohs(ip\_header->ip\_total\_length));

fprintf(logSniffer, " |-Identification : %d\n", ntohs(ip\_header->ip\_id));

fprintf(logSniffer, " |-Reserved ZERO Field : %d\n", (unsigned int)ip\_header->ip\_reserved\_zero);

fprintf(logSniffer, " |-Dont Fragment Field : %d\n", (unsigned int)ip\_header->ip\_dont\_fragment);

fprintf(logSniffer, " |-More Fragment Field : %d\n", (unsigned int)ip\_header->ip\_more\_fragment);

fprintf(logSniffer, " |-TTL : %d\n", (unsigned int)ip\_header->ip\_ttl);

fprintf(logSniffer, " |-Protocol : %d\n", (unsigned int)ip\_header->ip\_protocol);

fprintf(logSniffer, " |-Checksum : %d\n", ntohs(ip\_header->ip\_checksum));

fprintf(logSniffer, " |-Source IP : %s\n", inet\_ntoa(source.sin\_addr));

fprintf(logSniffer, " |-Destination IP : %s\n", inet\_ntoa(dest.sin\_addr));

}

Việc in này hoàn toàn rõ ràng và không có gì khó khăn. Sau đó là in thông tin UDP header. Vấn đề xuất hiện khi cần in nội dung của gói tin. Như ta đã biết là thông tin truyền trên internet theo dạng các tín hiệu. Các tín hiệu thường gộp lại thành từng giá trị hexa. Để chuyển thành dữ liệu thân thiện với người dùng, ta cần một hàm chuyển dữ liệu hexa thành ascii. Đây là loại mã hóa có thể giải mã được, nên ta cần một hàm giải mã như sau:

// Print hex data

void PrintData(char\* data, int Size) {

char a, line[17], c;

int j;

//loop over each character and print

for (int i = 0; i < Size; i++) {

c = data[i];

//Print the hex value for every character , with a space. Important to make unsigned

fprintf(logSniffer, " %.2x", (unsigned char)c);

//Add the character to data line. Important to make unsigned

a = (c >= 32 && c <= 128) ? (unsigned char)c : '.';

line[i % 16] = a;

//if last character of a line , then print the line - 16 characters in 1 line

if ((i != 0 && (i + 1) % 16 == 0) || i == Size - 1) {

line[i % 16 + 1] = '\0';

//print a big gap of 10 characters between hex and characters

fprintf(logSniffer, " ");

//Print additional spaces for last lines which might be less than 16 characters in length

for (j = strlen(line); j < 16; j++) {

fprintf(logSniffer, " ");

}

fprintf(logSniffer, "%s \n", line);

}

}

fprintf(logSniffer, "\n");

}

Việc in các data này là bắt chước với phong cách của một phần mềm thu thập gói tin nổi tiếng Wireshark. Và sau khi đã giải mã ra, ta sẽ in cả phần hexa và phần giải mã vào file log. File log này sẽ được khởi tạo và cấp quyền ghi ở trong hàm main

logSniffer = fopen("Sniffer.txt", "w");

if (logSniffer == NULL) {

printf("Unable to create file.");

}

Những hàm như in gói tin TCP hay ICMP cũng thực hiện tương tự. Chi tiết xem ở trong code.

Như vậy khi thực hiện chương trình sniffer, kết quả sẽ như sau:

Bắt đầu chạy chương trình:

Text

Description automatically generated

Chọn giao diện số 3 là card wifi:

Text

Description automatically generated

Chương trình đang bắt gói tin, và đây là kết quả

Text

Description automatically generated

File log sẽ ghi lại:

Graphical user interface, text, application

Description automatically generated

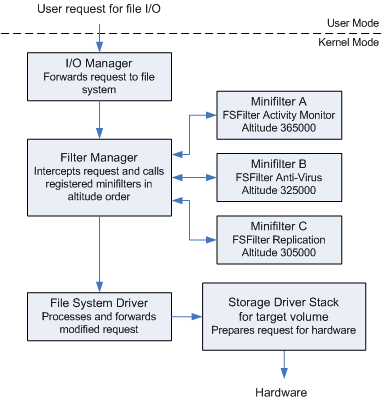
Kết thúc chương trình bắt gói tin. Tiếp theo chúng ta sẽ phân tích module chặn địa chỉ IP.

# **CHƯƠNG II: HỆ THỐNG CHẶN TRUY CẬP**

## ***2.1. Filter Driver***

Trình điều khiển bộ lọc – Filter Driver - là trình điều khiển tùy chọn thêm giá trị hoặc sửa đổi hoạt động của thiết bị. Trình điều khiển bộ lọc có thể phục vụ một hoặc nhiều thiết bị.

Mô hình cơ bản của hệ thống Windows như sau:



Khi người dùng thao tác, ứng dụng sẽ gửi các I/O Request Packet – IRP thông qua hệ thống I/O Manager để gửi trực tiếp tới File System Driver – Driver hệ thống file như NTFS, FAT để từ đó gửi các request đến Storage Driver điều khiển phần cứng cần thiết cho các thao tác. Tuy nhiên, một Filter Driver có thể can thiệp vào trong quá trình đó. Cụ thể, nó có thể can thiệp để chặn các IRP từ I/O Manager. Có thể có nhiều Driver và chúng được chia theo cấp độ ưu tiên, gọi là các Minifilter Driver. Những Driver này sẽ có quyền chặn các IRP và đọc chúng cũng như điều khiển, thay đổi các IRP này.

Có thể có những Low-level Filter Driver xuất hiện sau File System Driver và nằm phía trên Bus Driver.

Có rất nhiều loại Filter Driver, trong đó phổ biến sẽ là Keyboard Filter, Mouse Filter, File Filter, Network Filter, v.v…

## ***2.2. Network Filter Driver***

Như tên gọi của nó, Network Filter Driver – hay Network Driver Interface Specification (NDIS) filter có khả năng lọc tất cả lưu lượng mạng ở mức thấp nhất (trước khi dữ liệu được chuyển vào card mạng). Trình điều khiển giám sát tất cả các loại giao thức mạng có thể - Ethernet, IP, TCP, UDP.

Để lập trình được Network Filter Driver, ta phải cài đặt một bộ công cụ tùy chỉnh có sẵn của Windows là Windows Driver Kit (WDK).

Một Filter Driver nói chung và Network Filter nói riêng đều bắt đầu bằng một hàm DriverEntry() để khai báo khởi tạo một Driver. I/O Manager sẽ gọi DriverEntry() mỗi khi nó tải Driver.

PDEVICE\_OBJECT DeviceObject = NULL;

NTSTATUS DriverEntry(PDRIVER\_OBJECT DriverObject, PUNICODE\_STRING RegistryPath) {

NTSTATUS status;

DriverObject->DriverUnload = Unload;

status = IoCreateDevice(DriverObject, 0, NULL, FILE\_DEVICE\_UNKNOWN, 0, FALSE, &DeviceObject);

if (!NT\_SUCCESS(status)) {

return status;

}

status = InitializeWfp();

if (!NT\_SUCCESS(status)) {

IoDeleteDevice(DeviceObject);

return status;

}

}

Hàm DriverEntry này sẽ khởi tạo một con trỏ trỏ đến cấu trúc DRIVER\_OBJECT đại diện cho Driver để thao tác và đăng ký Driver này với Registry.

Mỗi Driver đều cần có một hàm DriverUnload để dỡ Driver (thuộc Windows Driver Models) khỏi hệ thống. Hàm Unload này sẽ thực hiện xóa Driver khỏi hệ thống

VOID Unload(PDRIVER\_OBJECT DriverObject) {

UnInitWfp();

IoDeleteDevice(DeviceObject);

KdPrint(("Unload\r\n"));

}

Trong hàm Unload, ta thực hiện xóa Device Object đã được khởi tạo gắn với Driver Object bằng hàm IoDeleteDevice() tương ứng với hàm IoCreateDevice(). Bắt buộc phải khởi tạo một Device Object, vì Driver Object là một đối tượng có thể hiểu là “trừu tượng” còn Device Object mới là đối tượng làm việc thực sự.

Hàm IoCreateDevice() nhận tham số là Driver Object đã khởi tạo trước đó và các chế độ làm việc chung nhất để tương thích với tất cả các loại Driver Object, đó là FILE\_DEVICE\_UNKNOWN. Như vậy coi như ta đã xong bước khởi tạo cho một Network Filter Driver.

## ***2.3. Windows Filtering Platform***

Windows Filtering Platform (WFP) là một tập hợp các dịch vụ hệ thống và API cung cấp nền tảng để tạo các ứng dụng lọc mạng. API WFP cho phép các nhà phát triển viết mã tương tác với quá trình xử lý gói tin diễn ra ở một số lớp trong ngăn xếp mạng của hệ điều hành. Dữ liệu mạng có thể được lọc và cũng có thể được sửa đổi trước khi nó đến đích.

Bằng cách cung cấp một nền tảng phát triển đơn giản hơn, WFP được thiết kế để thay thế các công nghệ lọc gói trước đây như bộ lọc Giao diện Trình điều khiển Truyền tải Transport Driver Interface (TDI), Bộ lọc Đặc tả Giao diện Trình điều khiển Mạng Network Driver Interface Specification (NDIS) và Nhà cung cấp Dịch vụ Phân lớp Winsock Layered Service Providers (LSP). Bắt đầu từ Windows Server 2008 và Windows Vista, móc tường lửa và trình điều khiển móc bộ lọc không khả dụng; các ứng dụng đang sử dụng các trình điều khiển này nên sử dụng WFP để thay thế.

Với WFP API, các nhà phát triển có thể triển khai tường lửa, hệ thống phát hiện xâm nhập, chương trình chống malware, công cụ giám sát mạng và kiểm soát của phụ huynh. WFP tích hợp và cung cấp hỗ trợ cho các tính năng tường lửa như giao tiếp được xác thực và cấu hình tường lửa động dựa trên việc sử dụng API ổ cắm (chính sách dựa trên ứng dụng) của các ứng dụng. WFP cũng cung cấp cơ sở hạ tầng để quản lý chính sách IPsec, thông báo thay đổi, chẩn đoán mạng và lọc trạng thái.

Windows Filtering Platform là một nền tảng phát triển chứ không phải là một bức tường lửa. Ứng dụng tường lửa được tích hợp trong Windows Vista, Windows Server 2008 và các hệ điều hành mới hơn Windows Firewall with Advanced Security (WFAS) được triển khai bằng WFP. Do đó, các ứng dụng được phát triển với WFP API hoặc WFAS API sử dụng logic trọng tài lọc chung được tích hợp sẵn trong WFP.

API WFP bao gồm API chế độ người dùng và API chế độ hạt nhân.

Để bắt đầu khởi tạo một WFP, ta sẽ viết hàm InitiateWfp() chạy trong DriverEntry() như sau:

NTSTATUS InitializeWfp() {

if (!NT\_SUCCESS(WfpOpenEngine())) {

goto end;

}

if (!NT\_SUCCESS(WfpRegisterCallout())) {

goto end;

}

if (!NT\_SUCCESS(WfpAddCallout())) {

goto end;

}

if (!NT\_SUCCESS(WfpAddSublayer())) {

goto end;

}

if (!NT\_SUCCESS(WfpAddFilter())) {

goto end;

}

return STATUS\_SUCCESS;

end:

UnInitWfp();

return STATUS\_UNSUCCESSFUL;

}

Để một WFP thực sự được khởi tạo, ta cần phải khởi tạo cho nó những thành phần như sau:

- Filter Engine: Đây là cơ sở hạ tầng lọc đa lớp, được lưu trữ ở cả chế độ hạt nhân và chế độ người dùng, thay thế nhiều module lọc trong hệ thống con mạng Windows XP và Windows Server 2003. Một Filter Engine có các chức năng:

+ Lọc lưu lượng mạng ở bất kỳ lớp nào trong hệ thống qua bất kỳ trường dữ liệu nào mà một miếng đệm (a shim) có thể cung cấp.

+ Triển khai bộ lọc "Callout" bằng cách gọi chú thích trong quá trình phân loại.

+ Trả về các hành động "Cho phép" hoặc "Chặn" đối với miếng đệm đã gọi nó để thực thi.

+ Cung cấp phân xử giữa các nguồn chính sách khác nhau. Ví dụ: xác định mức độ ưu tiên khi một ứng dụng được định cấu hình để bảo mật bất kỳ lưu lượng mạng nào liên quan đến nó, nhưng tường lửa cục bộ được định cấu hình để ngăn chặn lưu lượng bảo mật của ứng dụng.

- Base Filter Engine (BFE): Một dịch vụ kiểm soát hoạt động của WFP. Nó thực hiện các nhiệm vụ sau:

+ Chấp nhận bộ lọc và các cài đặt cấu hình khác cho nền tảng.

+ Báo cáo trạng thái hiện tại của hệ thống, bao gồm cả số liệu thống kê.

+ Thực thi mô hình bảo mật để chấp nhận cấu hình trong nền tảng. Ví dụ: quản trị viên cục bộ có thể thêm bộ lọc nhưng những người dùng khác chỉ có thể xem chúng.

+ Đặt lại cài đặt cấu hình cho các module khác trong hệ thống. Ví dụ: chính sách đàm phán IPsec chuyển đến module khóa IKE / AuthIP, bộ lọc chuyển đến công cụ lọc

- Shim: Các thành phần chế độ hạt nhân nằm giữa Network Stack và Filter Engine. Shims đưa ra quyết định lọc bằng cách phân loại dựa trên động cơ lọc.

- Callouts: Tập hợp các chức năng do trình điều khiển hiển thị và được sử dụng để lọc chuyên biệt. Bên cạnh các hành động cơ bản là "Cho phép" và "Chặn", chú thích có thể sửa đổi và bảo mật lưu lượng mạng đến và đi.

Như vậy, để có một WFP hoàn chỉnh, chúng ta cần thực hiện tối thiểu 2 bước sau: Khởi tạo một Filter Engine và Đăng ký ghi chú với Filter Engine.

Trong chương trình này, ta khởi tạo một Filter Engine bằng hàm:

NTSTATUS WfpOpenEngine() {

return FwpmEngineOpen(NULL, RPC\_C\_AUTHN\_WINNT, NULL, NULL, &EngineHandle);

}

Hàm FwpmEngineOpen() mở một phiên làm việc với Filter Engine, nhận vào các tham số quan trọng là RPC\_C\_AUTHN\_WINNT chỉ định dịch vụ xác thực để sử dụng ở đây là sử dụng Microsoft NT LAN Manager (NTLM) SSP (Windows Challenge/Response (NTLM) là giao thức xác thực được sử dụng trên các mạng bao gồm các hệ thống chạy hệ điều hành Windows và trên các hệ thống độc lập), và một tay cầm điều khiển Engine là EngineHandle được khai báo là NULL từ trước.

Để thêm bộ lọc tham chiếu đến chú thích, ta cần gọi các hàm theo thứ tự sau: Gọi FwpsCalloutRegister (được ghi trong Windows Driver Kit), để đăng ký chú thích với công cụ lọc; gọi FwpmCalloutAdd để thêm chú thích vào hệ thống và gọi FwpmFilterAdd để thêm bộ lọc tham chiếu đến chú thích vào hệ thống.

Sau khi thực hiện mở một phiên làm việc với Filter Engine, ta sẽ thực hiện đăng ký một chú thích bằng hàm:

// Register a Callout with Filter engine

NTSTATUS WfpRegisterCallout() {

FWPS\_CALLOUT Callout = { 0 };

Callout.calloutKey = WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID;

Callout.flags = 0;

Callout.classifyFn = FilterCallback;

Callout.notifyFn = NotifyCallback;

Callout.flowDeleteFn = FlowDeleteCallback;

FwpsCalloutRegister(DeviceObject, &Callout, &RegCalloutId);

}

Cấu trúc FWPS\_CALLOUT xác định dữ liệu cần thiết để trình điều khiển chú thích đăng ký chú thích với công cụ lọc. Trong đó có 2 giá trị quan trọng là calloutKey và classifyFn. Giá trị calloutKey là GUID do trình điều khiển chú thích xác định sẽ xác định duy nhất chú thích. GUID này khởi tạo bằng hàm:

DEFINE\_GUID(WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID, 0xd969fc67, 0x6fb2, 0x4504, 0x91, 0xce, 0xa9, 0x7c, 0x3c, 0x32, 0xad, 0x36);

Những giá trị phía sau là khởi tạo giá trị riêng cho GUID này, chỉ cần đảm bảo nó khác với tất cả GUID hiện tại đã tồn tại trong hệ thống.

Giá trị classifyFn là biểu thị cho chức năng chú thích của trình điều khiển chú thích. Công cụ bộ lọc gọi hàm này bất cứ khi nào có dữ liệu mạng được chú thích xử lý. Vì vậy, mỗi khi có một trao đổi dữ liệu diễn ra trên mạng, hàm FilterCallBack() sẽ được gọi một lần. Giá trị notifyFn là hàm trả về mã thông báo cho Callout Driver biết về các sự kiện được liên kết với chú thích, hiện tại trong giới hạn project này, ta để nó luôn luôn đúng:

NTSTATUS NotifyCallback(FWPS\_CALLOUT\_NOTIFY\_TYPE type, const GUID\* filterkey, const FWPS\_FILTER\* filter) {

return STATUS\_SUCCESS;

}

Giá trị flowDeleteFn: Một con trỏ đến chức năng chú thích flowDeleteFn của trình điều khiển chú thích. Công cụ bộ lọc gọi hàm này bất cứ khi nào một luồng dữ liệu đang được chú thích xử lý bị kết thúc. Nếu trình điều khiển chú thích không liên kết ngữ cảnh với luồng dữ liệu mà chú thích xử lý, thì thành viên này phải được đặt thành NULL. Vì Callout Driver không liên kết ngữ cảnh với luồng dữ liệu nên ta để hàm này là NULL.

Hàm classifyFn là FitlerCallback sẽ là hàm quan trọng nhất trong Driver này. Cụ thể, hàm sẽ nhận toàn bộ giá trị IP kết nối tới thiết bị, rồi so sánh với giá trị IP mà mình muốn chặn để chặn nó:

VOID FilterCallback(const FWPS\_INCOMING\_VALUES0\* Values, const FWPS\_INCOMING\_METADATA\_VALUES0\* MetaData, PVOID layerdata, const void\* context, const FWPS\_FILTER\* filter, UINT64 flowcontext, FWPS\_CLASSIFY\_OUT\* classifyout) {

ULONG LocalIP, RemoteIP;

ULONG targetIP = 0x2d3729df;

if (!(classifyout->rights & FWPS\_RIGHT\_ACTION\_WRITE)) {

goto end;

}

LocalIP = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_LOCAL\_ADDRESS].value.uint32;

RemoteIP = Values->incomingValue[FWPS\_FIELD\_ALE\_AUTH\_CONNECT\_V4\_IP\_REMOTE\_ADDRESS].value.uint32;

KdPrint(("PID is %ld --- Path is %S --- LocalIP is %u.%u.%u.%u --- RemoteIP is %u.%u.%u.%u \r\n",

(ULONG)(MetaData->processId),

(PWCHAR)(MetaData->processPath->data),

(LocalIP >> 24) & 0xFF, (LocalIP >> 16) & 0xFF, (LocalIP >> 8) & 0xFF, (LocalIP) & 0xFF,

(RemoteIP >> 24) & 0xFF, (RemoteIP >> 16) & 0xFF, (RemoteIP >> 8) & 0xFF, (RemoteIP) & 0xFF));

if (targetIP == RemoteIP) {

KdPrint(("Block \r\n"));

classifyout->actionType = FWP\_ACTION\_BLOCK;

classifyout->rights &= ~FWPS\_RIGHT\_ACTION\_WRITE;

return;

}

else {

classifyout->actionType = FWP\_ACTION\_PERMIT;

}

end:

classifyout->actionType = FWP\_ACTION\_PERMIT;

if (filter->flags & FWPS\_FILTER\_FLAG\_CLEAR\_ACTION\_RIGHT) {

classifyout->rights &= ~FWPS\_RIGHT\_ACTION\_WRITE;

}

}

Đầu tiên, ta cần cài đặt quyền cho hàm này, bằng cách kiểm tra quyền của nó thông qua giá trị classifyout->rights. Cấu trúc FWPS\_CLASSIFY\_OUT xác định dữ liệu được trả về cho trình gọi của hàm chú thích ClassFn của chú thích. Nếu quyền của chú thích khác với FWPS\_RIGHT\_ACTION\_WRITE (Nếu cờ này được đặt, trình điều khiển chú thích có thể ghi vào thành viên actionType của cấu trúc này. Nếu cờ này không được đặt, trình điều khiển chú thích chỉ có thể ghi vào thành viên actionType của cấu trúc này nếu nó đang phủ quyết một hành động FWP\_ACTION\_PERMIT đã được trả về trước đó bởi một bộ lọc có trọng số cao hơn trong bộ lọc), thì sẽ cài đặt cho chú thích có quyền FWP\_ACTION\_PERMIT (Cho phép dữ liệu được truyền hoặc nhận). Cấu trúc FWPS\_FILTER xác định một bộ lọc thời gian chạy trong công cụ bộ lọc. Nếu bộ lọc thời gian đã xóa hết quyền của chú thích, lúc này ta phải cấp quyền cho trình điều khiển chú thích có thể ghi vào thành viên actionType của cấu trúc này. (Tất cả thực hiện ở end: ).

Nếu chú thích đã có quyền, ta sẽ nhận giá trị LocalIP là IP của máy tính và RemoteIP là IP của những kết nối đến máy tính. Cấu trúc FWPS\_INCOMING\_VALUES xác định các giá trị dữ liệu được bộ lọc chuyển đến hàm phân loại của chú thích. Khi đó, LocalIP và RemoteIP đã được xác định, tuy nhiên chúng sẽ ở dưới dạng hexa, nên để đọc được, ta sẽ sử dụng các phép toán dịch bit, and bit.

Cuối cùng, so sánh RemoteIP với targetIP, nếu targetIP trùng RemoteIP thì sẽ chặn quyền truy cập tới IP đó, bằng cách thực hiện FWP\_ACTION\_BLOCK – chặn truy cập và xóa quyền FWPS\_RIGHT\_ACTION\_WRITE của Callout Driver. Lúc này, IP đó sẽ bị chặn không tới được máy tính, và Callout Driver cũng không cho phép dữ liệu được truyền hoặc nhận. Nếu không trùng IP, Callout Driver sẽ cho phép dữ liệu được truyền và nhận.

Sau khi đăng ký một Callout Driver, việc tiếp theo là gắn chú thích vào trong Filter Engine, sử dụng hàm FwpmCalloutAdd():

// Add new callout object annotations

NTSTATUS WfpAddCallout() {

FWPM\_CALLOUT callout = { 0 };

callout.flags = 0;

callout.displayData.name = L"EstablishCalloutName";

callout.displayData.description = L"EstablishCalloutName";

callout.calloutKey = WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID;

callout.applicableLayer = FWPM\_LAYER\_ALE\_AUTH\_CONNECT\_V4;

return FwpmCalloutAdd(EngineHandle, &callout, NULL, &AddCalloutId);

}

Những giá trị của cấu trúc FWP\_CALLOUT là cơ bản và rất dễ hiểu, điều quan trọng cần chú ý là calloutKey phải trùng GUID với calloutKey trong Callout đã đăng ký với Filter Engine, khi đó Filter Engine mới nhận Callout object.

Bình thường đến bước này ta chỉ cần sử dụng hàm FwpmFilterAdd() thêm bộ lọc vào lớp con được chỉ định tại mọi lớp lọc trong hệ thống. Tuy nhiên hiện tại không có một lớp con được chỉ định nào, vì vậy ta phải tạo ra một lớp con cho hệ thống:

// Add new sublayer

NTSTATUS WfpAddSublayer() {

FWPM\_SUBLAYER sublayer = { 0 };

sublayer.displayData.name = L"EstablishedSublayerName";

sublayer.displayData.description = L"EstablishedSublayerName";

sublayer.subLayerKey = WFP\_SAMPLE\_SUB\_LAYER\_GUID;

sublayer.weight = 65560;

FwpmSubLayerAdd(EngineHandle, &sublayer, NULL);

}

Ta cũng phải tạo ra một GUID riêng cho sublayer này:

DEFINE\_GUID(WFP\_SAMPLE\_SUB\_LAYER\_GUID, 0xed6a516a, 0x36d1, 0x4881, 0xbc, 0xf0, 0xac, 0xeb, 0x4c, 0x4, 0xc2, 0x1c);

Sau khi tạo ra một sublayer, ta mới có thể gắn Filter vào các lớp con của hệ thống:

// Add new filter

NTSTATUS WfpAddFilter() {

FWPM\_FILTER filter = { 0 };

FWPM\_FILTER\_CONDITION condition[1] = { 0 };

FWP\_V4\_ADDR\_AND\_MASK AddrandMask = { 0 };

filter.displayData.name = L"filterCalloutName";

filter.displayData.description = L"filterCalloutName";

filter.layerKey = FWPM\_LAYER\_ALE\_AUTH\_CONNECT\_V4;

filter.subLayerKey = WFP\_SAMPLE\_SUB\_LAYER\_GUID;

filter.weight.type = FWP\_EMPTY;

filter.numFilterConditions = 1;

filter.filterCondition = condition;

filter.action.type = FWP\_ACTION\_CALLOUT\_TERMINATING;

filter.action.calloutKey = WFP\_SAMPLE\_ESTABLISHED\_CALLOUT\_V4\_GUID;

condition[0].fieldKey = FWPM\_CONDITION\_IP\_REMOTE\_ADDRESS;

condition[0].matchType = FWP\_MATCH\_EQUAL;

condition[0].conditionValue.type = FWP\_V4\_ADDR\_MASK;

condition[0].conditionValue.v4AddrMask = &AddrandMask;

return FwpmFilterAdd(EngineHandle, &filter, NULL, &filterid);

}

Biên dịch chương trình, ta nhận được một file .sys là file để load driver vào hệ thống. Dùng phần mềm OSR Library để load file vào hệ thống và cho Start Service, ta được kết quả là những IP khác với targetIP đều truy cập được, tuy nhiên web có targetIP thì bị chặn, cụ thể đó là sublimetext.com.

Thực hiện kiểm tra các giá trị log trên DebugView, ta sẽ thu được kết quả là những IP mà ta kết nối đến cùng với thông tin là trang có targetIP đã bị block.

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

# **KẾT LUẬN**

Ngày nay hệ điều hành Windows là hệ điều hành cực kỳ phổ biến, tuy nhiên chúng có rất nhiều lỗ hổng bảo mật gây thiệt hại nặng nề về tính riêng tư và toàn vẹn dữ liệu. Những giải pháp giống như project em thực hiện là cực kỳ cần thiết để bảo vệ sự an toàn cho hệ điều hành Windows.

Hệ thống phòng chống xâm nhập liên tục giám sát lưu lượng mạng, đặc biệt là ở các gói tin riêng lẻ, để tìm kiếm bất kỳ cuộc tấn công nguy hiểm nào có thể xảy ra. Nó thu thập thông tin về các gói tin này và báo cáo cho quản trị viên hệ thống, nhưng nó cũng thực hiện những động thái phòng ngừa của riêng mình. Nếu phát hiện phần mềm độc hại tiềm ẩn hoặc loại tấn công khác, IPS sẽ chặn các gói đó truy cập vào mạng. IPS được triển khai dựa trên tiền đề của Network Filter Driver.

Project này vẫn còn thô và có thể phát triển lên rất nhiều trong tương lai. Vì vậy em hi vọng mình có thể dựa trên project này để xây dựng những ứng dụng có tính thực tiễn cao, áp dụng rộng rãi và đem lại sự an tâm cho những người sử dụng.

# **TÀI LIỆU THAM KHẢO**

1. <https://docs.microsoft.com/en-us/>

2. <https://www.osr.com/osr-learning-library/>

3. Microsoft Cooporation, Windows Internal Part I, Part II, 2020, <https://github.com/zodiacon/WindowsInternals>

4. Tham khảo fix bug trên stackoverflow