HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT 2

Nguyễn Đức Tâm 20210767

Topic: Building a tool to dump information from compromised network nodes

Supervisor: MSc.Bùi Trọng Tùng

School: Information and Communications Technology

HANOI, 2024

TABLE OF CONTENTS

List of Figures

List of Tables

1 Introduction

In today's digital world, cyberattacks targeting businesses, organizations, as well as individuals are tending to become more and more common, complex and sophisticated. From this reality, securing network system has become more urgent and necessary than ever.

Additionally, the reason I chose this topic comes from the relevance to the work I am currently doing as well as my future career orientation which is to become an incident response specialist and forensic investigator. At the place where I am currently working, the main job is to learn about new tool that can be useful in incident response cases. The reason why I need to learn about tool is because in such cases, the number of compromised machines can be very large and the information which need to be gathered is scattered everywhere in the machine. This leading to the investigator may spend hours just to collecting information. Therefore, the need of a tool that can automatically collect needed information from many compromised machines to the investigator can be very effective and productive.

Based on the realistic and personal reason above, my topic in this project is about "Building a tool to dump information from compromised network nodes". In this topic, I will focus on building a tool to extract the needed information from the nodes which is marked compromised. This tool can be used in the forensic investigations in cyberattacks. By using this tool we can gather as much information from the compromised network nodes as possible, after that the analyzing process will be taken place to produce the IOCs (Indicator of Compromised) for preventing future attacks.

To achieve the goal of this topic about dumping information from compromised network nodes, I will develop a system based on the server-client model. The server will manage multi-client connection, send a signal to specific client to inform it to start extract information. The server also manages the received information from the client by using the filesystem of the operation system. For the client, i will develop it so that as soon as it receives the signal from server, it starts extracting process to get the required information. After that, the client will send all the information to the server. Additionally, the client also manages the extracted information by using the filesystem of the operating system as the server.

The structure of this report will be as following. For the first part, I will introduce to you my topic as well as the reason I chose it. For the second part, it will be the analysis of the topic problem. In this part, i will mention to the goals that I want to achieve in this topic as well as the tasks that i was assigned during the development by my supervisor. I also briefly outline the subproblems and the method to solve them. In the third part, I will dive into details of each method that i used to solve each subproblem. The forth part will be about the deployment and testing the tool. The last one will summary all the work that I have done in this project and what I have learn through this. I also mention to the future work of this topic in this part as well.

2 Problem Analysis

2.1 Statement of The Problem

Currently, with the world in general and Vietnam in particular, cyberattacks are tending to increase rapidly. Not only that, the attackers's methods and tricks are becoming increasingly complex and difficult to dectect.

According to a newspaper of "Lao Dong" page. The number of cyber attacks on organizations in Vietnam increased by 9.5% compared to 2022, an average of 1,160 cases per month. In 2023, NCS (National Cyber Security - a corporation operates in the field of technology, providing solutions and services for cybersecurity, information security, system integration, as well as IT and telecommunication package solutions) recorded 13,900 network attacks into organizations in Vietnam. The targets that suffered the most attacks were government agencies, banking systems and other critical systems. According to NCS statistics, there were up to 342 educational websites with the domain name ".edu.vn" and 212 websites were attacked many times without a thorough solution.

There are many existing solutions that help the investigator to faster in the process of acquisition and analysis cyberattacks. The first one that I want to mention is the Endpoint Detection and Respond, abbreviated as EDR. The Endpoint Detection and Respond also referred to as endpoint detection and threat response (EDTR), is an endpoint security solution that continuously monitors end-user devices to detect and respond to cyber threats like ransomware and malware. Endpoint Detection and Respond is defined as a solution that records and stores endpoint-system-level behaviors, uses various data analytics techniques to detect suspicious system behavior, provides contextual information, blocks malicious activity, and provides remediation suggestions to restore affected systems. The Endpoint Detection and Respond records the activities and events taking place on endpoints and all workloads, providing security teams with the visibility they need to uncover incidents that would otherwise remain invisible. The use of Endpoint Detection and Response tools is expected to grow a lot in the next few years. According to Stratistics MRC's report on EDR from 2017 to 2026, sales of EDR products, both on-premises and in the cloud, will likely reach $7.27 billion by 2026, with an annual growth rate of almost 26%. There are a couple of reasons for this increase. One reason is the rising number of devices connected to networks. Another reason is that cyberattacks are becoming more advanced and often target these devices because they are easier to attack.

The second solution that I want to mention is the Security Information and Event Management, abbreviated as SIEM. Security Information and Event Management is a security solution designed to help organizations identify and address potential security threats and vulnerabilities before they can impact business operations. Security Information and Event Management systems enable security teams to detect unusual user behavior and automate many of the manual processes involved in threat detection and incident response. The origi-

nal Security Information and Event Management systems started as log management tools, integrating security information management (SIM) and security event management (SEM) functions. These platforms allowed real-time monitoring and analysis of security events, while also helping in the tracking and logging of security information for compliance and auditing needs. Gartner coined the term SIEM in 2005 to describe this fusion of SIM and SEM technologies. Over time, SIEM software has advanced to include user and entity behavior analytics (UEBA) and other sophisticated security analytics, leveraging AI and machine learning to detect unusual behaviors and signs of advanced threats. Today, Security Information and Event Management system is essential in modern security operation centers (SOCs) for monitoring security and managing compliance. The Security Information and Event Management system is defined with many functionalities such as gathering event data from diverse sources like users, endpoints, applications, and networks, including cloud and on-premises environments. It integrates data from security tools such as firewalls and antivirus software, analyzing it in real-time.

The third one that I want to mention is the Volatility Framework. In 2007, the first version of The Volatility Framework was released publicly at Black Hat DC. The software was based on years of published academic research into advanced memory analysis and forensics. Volatility introduced people to the power of analyzing the runtime state of a system using the data found in volatile storage (RAM). It also provided a cross-platform, modular, and extensible platform to encourage further work into this exciting area of research. Another major goal was to encourage collaboration, innovation, and accessibility to knowledge that had been common within the offensive software communities. Volatility is now the world's most widely used memory forensics platform, which is supported by one of the largest and most active communities in the forensics industry. Volatility also provides a unique platform that enables research to be immediately transitioned into the hands of digital investigators, and Volatility has been used on some of the most critical investigations of the past decade. It has become an indispensable digital investigation tool forensic investigators around the world.

2.2 Goal and Task

The goal of this project is to develop a automation tool for extracting needed information from the network nodes that is marked compromised to help the investigators addressing challenges of attackers in cyberattacks. The primary functionalities in this project will be as follow. First, the server need to be able to manage multiple client connection at the same time. Secondly, the server need to be able to make a schedule to clients which means sending a signal to the client after some period of time. Thirdly, the server need to be able to manage the receiving data which is sent from the client effectively and systematically so that the investigator can easily to track and use. Fourthly, the client need to be able to make connection with the server. Fifthly, the client need to be able to receiving incoming signal which is sent from the server, it also need to send the extracted data to the server. Next, the client need to be

able to store extracted data in an effectively and systematically way. Finally, the client need to be able to perform extracting the RAM, information of current running process, registry of the system and the information of the network connection.

During the process of implementing this project, my supervisor, Master Bui Trong Tung, assigned me tasks that helped me complete the proposed project goals. The first task is to learn about the topic which help me thoroughly understand the topic, know what to do next, what is the purpose of this topic. The second task is to implement a server in Linux operating system that can handle connection request from multiple clients, sent a signal to clients to inform them to start extracting information and store receiving information which is sent from the client. The third task is to develop a client running in Windows operating system. The client can receive the signal sent from the server and start extracting information. The fourth task is to develop the client so that it will extract RAM, registry, information of current running process in the system and the information of the network connection of the compromised network node. The fifth task is to develop the API for the client so that other software rather than only my server can use it functionalities. The last task is to write a report that detail the progress of implementation this topic.

2.3 Subproblem and Solution Orientation

The problem of extracting effectively information like RAM, process, network,... in the compromised network nodes is the big problem that if we want to solve it, we need to break it down into easier subproblems.

The first subproblem we need to solve is that, how can a server control or in other way manage multiple thread that each thread represents a connection with a specific client. Since my supervisor was assigned to me to build a server running in the Linux operating system, the first ideal comes into my mind was using Thread API of the Linux operating system. Threads are powerful tools for concurrent and parallel execution in a Linux environment. In computer science, a thread is a sequence of programmed instructions managed independently by a scheduler, usually part of the operating system. Typically, a thread operates within a process. Threads can run concurrently using multithreading capabilities, allowing them to share resources like memory. This is different from processes which do not share these resources. Threads within a process share its executable code and the values of dynamically allocated variables and non-thread-local global variables at any point in time. In Linux, threads are implemented as a lightweight alternative to processes, and they are known as "lightweight processes" or "LWP". They use the same memory as the main process and are managed by the kernel. This helps them talk to each other easily and quickly. Threads are good for jobs that need to run at the same time, like making graphical programs, running web servers, and handling databases.

After we have the ideal to solve the first problem, the next thing need to be solved is how to establish the connection between server and client. So to solve this problem, the most

famous and popular way is, using Network Socket. So what is Socket? Socket is considered a communication port between applications on the network. Socket allows communication between two different processes on the same machine or different machines. To be more precise, it is a way to communicate with computers using standard Unix file descriptors. In Unix, every I/O action is performed by writing or reading a file descriptor. The file descriptor is just an integer associated with an open file, and it could be a network connection, text file, terminal, or something else. To a programmer, a socket acts like a low-level file descriptor. This is because commands like read() and write() work with sockets the same way they do with files and paths. Sockets were first introduced in 2.1BSD and later refined into their current form with 4.2BSD. The Socket feature is available with most current UNIX system releases. Unix Socket is used in client-server application framework. Most application-level protocols such as FTP, SMTP and POP3 use Socket to establish a connection between client and server and then to exchange data.

Now we have the connection between the server and the client, how can we ensure that the connection is reliable so that the data sending between server and client does not lost or out of order. Even if those cases occur how can we to resend it. Those problems are the next subproblem we need to solve which we can call sending and receiving message control problem. Based on the above description of the problem, one solution which are very suitable to this situation is Transport Control Protocol, abbreviated as TCP. Transmission Control Protocol is a communications standard that enables application programs and computing devices to exchange messages over a network. It is designed to send packets across the internet and ensure the successful delivery of data and messages over networks. TCP is one of the basic standards that define the rules of the internet and is included within the standards defined by the Internet Engineering Task Force (IETF). It is one of the most commonly used protocols within digital network communications and ensures end-to-end data delivery. TCP organizes data for transmission between a server and a client, guaranteeing the integrity of the data. Before transmitting data, TCP establishes a connection between the source and destination, keeping this connection active until the communication is complete. It then breaks large amounts of data into smaller packets, maintaining data integrity throughout the process. As a result, high-level protocols that need to transmit data all use TCP Protocol. Examples include peer-to-peer sharing methods like File Transfer Protocol (FTP), Secure Shell (SSH), and Telnet. It is also used to send and receive email through Internet Message Access Protocol (IMAP), Post Office Protocol (POP), and Simple Mail Transfer Protocol (SMTP), and for web access through the Hypertext Transfer Protocol (HTTP). In addition to the above protocols, there is also a type of socket that uses TCP for transmission, called Stream Socket. This is the one that we choose to use. With Stream Socket, delivering in network is guaranteed. If you send three items "A, B, C" in order, they will arrive in the same order - "A, B, C". These Socket use TCP to transmit data. If delivery is not possible, the sender will receive an error indication. Data records do not have any boundaries. All of these are implemented under Windows APIs and Linux syscall.

So at this moment, the client and server can send data to each other in a reliable way. Then, what do they do as they receive data which is sent from the other. Or more specific, how do they store them so that they can easily to maintain, update as the new one comes. This is the fourth subprolem that I need to solve. Since in every operating system, there always exists a component responsible for storing and managing file, data, information,... which is called file system. I will leverage this functionality of the operating system to solve the subproblem. File System is a software or system used to manage, organize and store data on storage devices such as hard disks, USB drives or memory cards. It provides users with an interface to access and manage files and folders in a storage system. File System also ensures data integrity and security by controlling access and providing data backup and restore features. A file system contains all features such as file name, path, size, access rights and time of file creation, modification and access. File System is an important component in a computer system, especially when it relates to applications that manage large data and require high reliability. Without a file system, the operating system would see only large chunks of data without any way to distinguish one file from the next. File system can be thought of as a type of index for all the data contained in a storage device. In addition to Solid-state drive - SSDs and Hard disk drive - HDDs, file systems are used for optical disks, flash drives and magnetic tape. File systems specify conventions for naming files, including the maximum number of characters in a name, which characters can be used and how long the file name extension can be. Directoriy hierarchy is the most popular way that most file systems use to organize files. The file location will be described by its path within the directory structure. Directories are organized like a tree, with the root directory at the top. Each file is placed in a directory or subdirectory within this structure. Another concept in the file system is partition which need to be created before creating files and directories on storage. A partition is a section of storage that the operating system manages separately, we can pretend them like different drives. Each partition can have its own file system. Partitions help keep files and operations separate, improving performance, security, and maintenance. File systems manage the physical and logical tasks of handling files.File systems also store small amounts of other data to maintain the system and help access files. A file system divides a partition into blocks. Most blocks store file content data, while some blocks store metadata and management data. The size of these blocks varies depending on the file system.

After we have handled the connection and storage, we start considering to how to extract the requirement information. The first information we need to achieve is the Random Access Memory, abbreviated as RAM. Random access memory is the short-term memory in a computer, it temporarily holds the data that the processor needs for immediate tasks. Unlike hard disks which are slower, RAM allows for much faster data retrieval, significantly boosting system performance. All devices in the computer world, from desktops and laptops to tablets, smartphones, and IoT devices like smart TVs, utilize RAM. While these devices also have long-term storage options, the data required for current operations resides in RAM. Notably, RAM is volatile memory, meaning it loses its data when the device powers off. This

characteristic makes RAM ideal for managing active processes, applications, and programs, such as web browsing. To accomplish a specific task, the operating systems load data from the hard disk into RAM to process it. When it's finished working with that data, the computer converts it back into long-term storage. For example, As you open a Microsoft Word program, the computer loads the file executable and its requirement libraries into its RAM. Or if you open a document which already have stored in the disk of your computer, the operating system will locates the file in the long-term storage and moves it into the RAM. It does so because of the fast performance of the RAM. In contrast, when you save a document, its content will be moved from RAM back to the long-term storage, and closing an application will clear the content of it in RAM to free up space for new application. These are some fundamental definitions relate to RAM, so how to extract it. There are various tools available for memory acquisition, ranging from open-source to commercial solutions. But in this project I choose to use WinPMEM, an open-source memory acquisition tool that can capture physical memory on both 32-bit and 64-bit Windows operating systems. WinPMEM is a free, open-source tool that is available for download from the GitHub repository. This tool is written in C++ and supports various operating systems, including Windows, Linux, and FreeBSD. It provides the user with the ability to acquire the memory contents of a target system by creating a raw memory dump. One of the key benefits of WinPMEM is its ability to capture memory from systems with different hardware configurations. The tool can automatically detect the system's hardware configuration and adjust its memory acquisition techniques accordingly. This makes it an effective tool for acquiring memory from different systems, regardless of their hardware specifications.

The next important thing need to extract from the compromised network nodes is the current running process information. In the field of computing, a process refers to an instance of a computer program being executed by one or more threads. There are various models of processes, ranging from lightweight to more complex structures, such as entire virtual machines. Basically, most processes originate from an operating system process, which encompasses the program's code, allocated system resources, access permissions, and data structures necessary for initiating, managing, and coordinating execution activities. Depending on the operating system, a process might consist of multiple threads running instructions simultaneously. In contrast, a computer program is a passive set of instructions usually stored in a file on a disk. When these instructions are loaded into memory for execution, they become a process. Multiple processes can be linked to the same program. For example, opening multiple instances of the same application results in the creation of several processes. There are many ways to retrieve the current working process information, but in this project I will leverage the built-in utility named "tasklist". The tasklist command-line utility offers a straightforward way to enumerate all active processes on a Windows machine, displaying details such as process IDs, memory usage, and image name. With the verbose mode, it can even extract more information like session name, session number, username,...

There is one place that the attackers often access to once they gain the access to the system

to perform the persistence. The name of that place is registry. This is next kind of information that we need to collect on the compromised network nodes. So what is registry. The Windows Registry is a hierarchical database that stores low-level settings for the Microsoft Windows operating system and for applications that choose to use the registry. The kernel, device drivers, services, Security Accounts Manager, and user interfaces can all use the registry. In other words, the registry or Windows Registry contains information, settings, options, and other values for programs and hardware installed on all versions of Microsoft Windows operating systems. For example, when a program is installed, a new subkey containing settings such as a program's location, its version, and how to start the program, are all added to the Windows Registry. The registry contains two basic elements which are keys and values. Registry keys are container objects similar to folders. Registry values are non-container objects similar to files. Keys may contain values and subkeys. Keys are referenced with a syntax similar to Windows path names, using backslashes to indicate levels of hierarchy. The hierarchy of registry keys can only be accessed from a known root key handle that is mapped to the content of a registry key preloaded by the kernel from a stored "hive", or to the content of a subkey within another root key, or mapped to a registered service or DLL that provides access to its contained subkeys and values. There are five common root keys that I will focus to extract in this project which are HKEY_LOCAL_MACHINE or HKLM, HKEY_CURRENT_CONFIG or HKCC, HKEY_CLASSES_ROOT or HKCR, HKEY_CURRENT_USER or HKCU and HKEY_USERS or HKU. The method that I use to extract those things is by leveraging 2 core Windows APIs which are RegEnumKey and RegEnumValue. These APIs allow for detailed enumeration of registry keys and values, enabling comprehensive data collection from the Windows Registry. RegEnumKey enumerate subkeys within a specified registry key, while RegEnumValue retrieves the values associated with a specified key. Together, they provide a complete view of the registry's structure and contents, essential for identifying malicious modifications or unauthorized changes.

The final information which is important for a digital forensics investigator or cybersecurity analyst is the network connection. How to gain these connection information are the final subproblem that we need to solve. To extract network connection information from compromised network nodes, I use both built-in netstat commands and PowerShell cmdlets Get-NetAdapterStatistics to meet the requirement. The netstat command is a powerful built-in tool that provides detailed information about active network connections, including local and remote addresses, port numbers, and connection states. Running netstat -ano can further include process IDs associated with each connection. For more detailed, I combine it with the PowerShell's Get-NetAdapterStatistics cmdlet. This cmdlet extracts the network adapter statistics, including bytes sent and received, packet counts, and error rates. Combining result of netstat with the detailed statistics from Get-NetAdapterStatistics provides an effective way for detecting and analyzing compromised network, thereby supporting incident response and forensic investigations.

3 Implementation

3.1 Multi-thread Control in Server

So as discussed above, I will use multiple thread, each for one specific client connection so that the server can handle many requests from different places at the same time. There is a library in C for Linux operating system which provides a set of functions for creating and manipulating threads, named pthreads. I will leverage this thread library. To create a new thread, I am going to use the `pthread_create()` function. The `pthread_create()` function will starts a new thread in the calling process. This function takes four arguments.

```
int pthread_create(pthread_t* thread_id,
                   pthread_attr_t* attr,
                   void* (*function)(void*),
                   void* param);
```

The parameters of the `pthread_create()` function are described as follows:

- `pthread_t* thread_id`: A pointer to a `pthread_t` variable, which will be used to store the thread ID of the newly created thread.
- `pthread_attr_t* attr`: A pointer to a `pthread_attr_t` variable, which can be used to set attributes for the new thread (such as the thread's scheduling policy or stack size). This argument is optional and can be set to NULL if the default attributes should be used.
- `void (*function)(void*)`: A pointer to the function that the new thread should execute. This function should have a `void*` argument and return `void*`.
- `void* param`: Last parameter is `void*` which is parameter to be sent to thread function from parent thread or process.

In my server I will apply the syscall as follow:

```
int ret = pthread_create(&x, NULL, &clientThread, cinf);
```

As you can see, I create a new thread that will run the `clientThread` function with the parameter `cinf`. The `cinf` parameter is a `clientInfo` structure, a structure that I defined so that the server can manage the clients.

```
struct clientInfo
{
   int sockfd;
   struct sockaddr_in cli_in;
   unsigned int interval;
};
```

Each element in the `clientInfo` struct will have the following meaning:

- `sockfd`: An integer that represents the sock file descriptor. This element is required so that the server can know where to send and receive data from client.
- `cli_in`: An element that stores the information about the client such as client's IP, client's port,...
- `interval`: An element that let the server know after how long it will need to send again a `dump signal`

For the `clientThread` function, here is how it work. First, it will get the socket file descriptor `sockfd` from the struct `clientInfo` argument of the thread so that it knows where to send and receive data from. Then, the client thread will retrieve the current working directory to store the future sent client data. After that, server starts an infinite `while loop`. Each time, server sends a signal which is just a message with content `dump` to client after a period of time which is defined in `interval` element of `clientInfo` struct. Then, server waits to receive the extracted data sent from client. The flowchart of `clientThread` function will be as below:

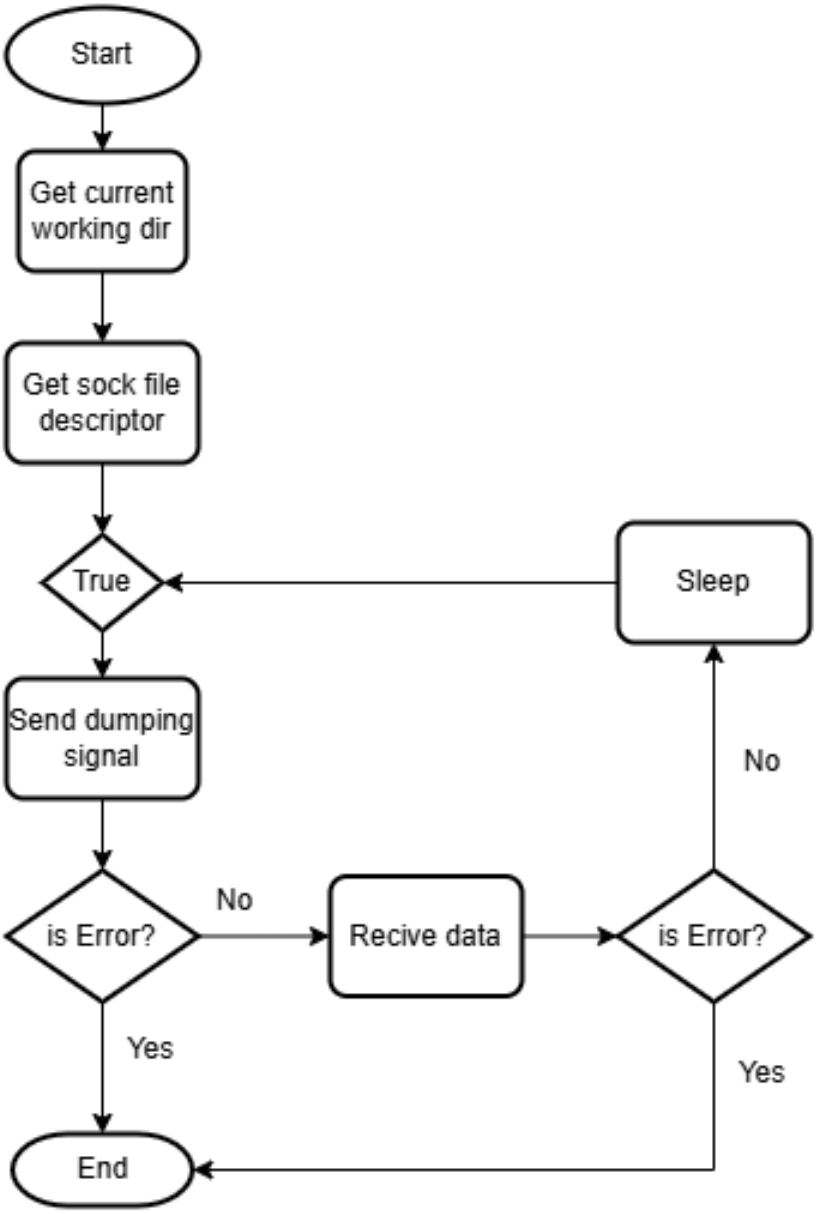

Fig. 1. Client Thread Flowchart

3.2 Establishing Connection

3.2.1 Server Side

To establish connection, the server side, first, creates a listening stream socket which uses Transport Control Protocol (TCP) and IPv4 Internet Protocol (AF_INET). This socket will responsible for receive the connection request from client.

```
int listeningSockfd = socket(AF_INET, SOCK_STREAM, 0);
```

After that, server sets a specific port to the socket and binds all available network interfaces on the host to the socket

```
sockaddr_in serv_addr;
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(port_num);
serv_addr.sin_addr.s_addr = INADDR_ANY;
bind(listeningSockfd,
     (sockaddr*) &serv_addr,
     sizeof(serv_addr))
```

Finally, server begins an infinite `while loop`. Each time, server waits for new connection, then storing the new client information and creating a new thread to serve that client.

```
while(true)
{
     int clientSocket = accept(listeningSockfd,
                               (struct sockaddr*)&cliAddr,
                               &addr_size);
     struct clientInfo* cinf = new clientInfo;
     cinf->sockfd = clientSocket;
     cinf->interval = interval;
     cinf->cli_in = cliAddr;
     pthread_create(&x, NULL, &clientThread, cinf);
}
```

The flowchart that describes the connection establishing process of server will be as follow:

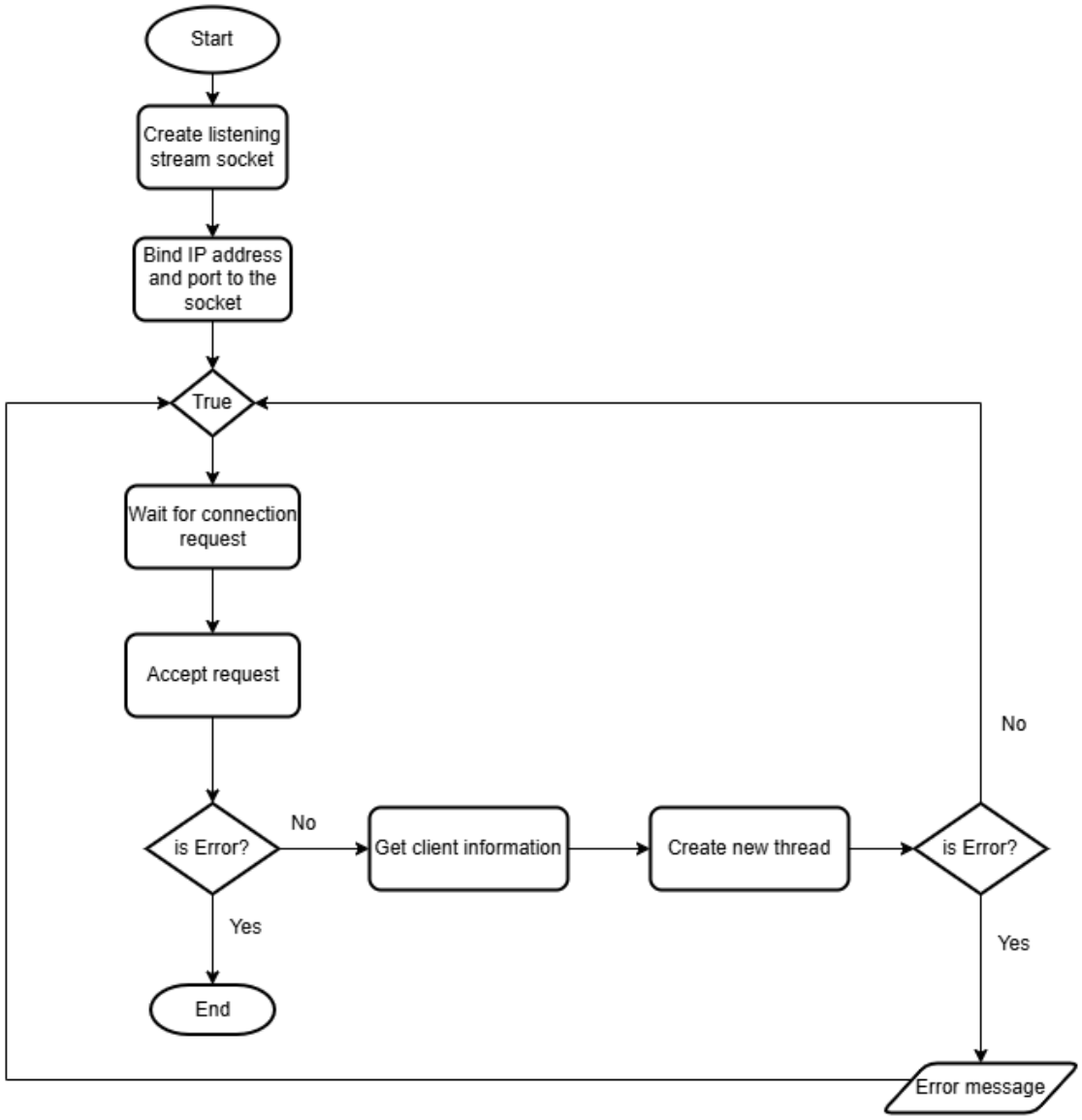

Fig. 2. Server Establishing Connection Flowchart

3.2.2 Client Side

For the client, to create a connection with the server, I have created a function named `constructConnection` to fulfill the requirement. This function take two string arguments which is the port and the IP address of the server. The function returns a `SOCKET` which is defined in Windows operating system so that it can be used to send and receive data from server. When the function is called, it, first, tries to resolve the server address and port into a form that can be used by network functions. Next, `constructConnection` function create a socket based on the information received after resolving such as address family (IPv4 or IPv6) and socket type. Finally, the function sends a connection request to the

specified server through the created socket. If the server accepts the request, the connection is established. The flowchart of the `constructConnection` function will be as flow:

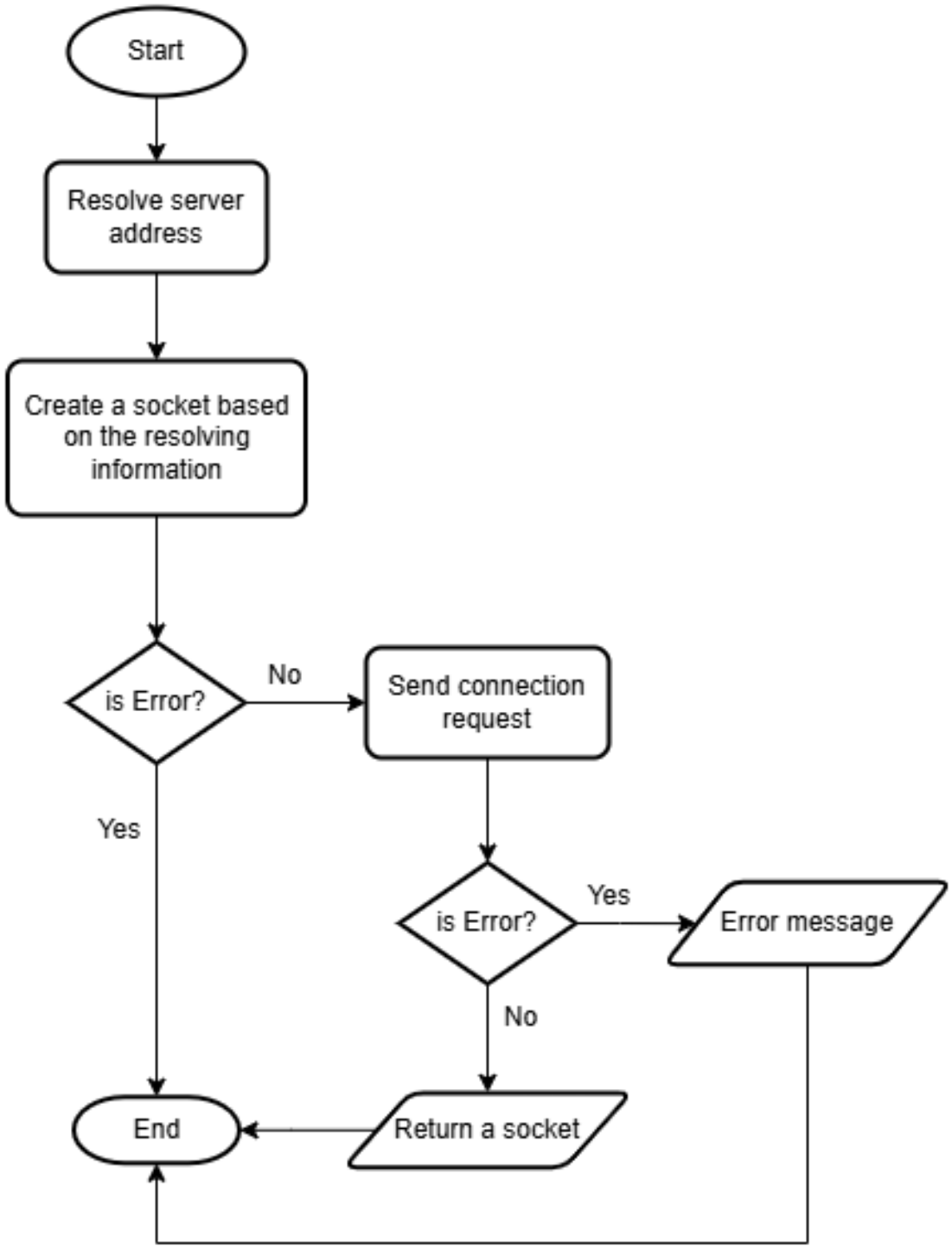

Fig. 3. Constructing Connection Flowchart

3.3 Sending and Receiving Message Control

To solve the sending and receiving message problem of both client and server, I have defined a `box` structure so that for each message sending from both sides can have more information.

```
struct box
{
    short fileType;
    unsigned long long fileSize;
    char data[SIZE];
};
```

Each element in the `box` struct will have the following meaning:

- `fileType`: This element will let the receiving side know whether the data its receiving is a normal file, a directory or a nature data.
- `fileSize`: This element will let the receiving side know the size of the file will be sent so that it knows how much data need to received.
- `data`: If the 2 elements above just are metadata, this element contains the actual content.

Now, to send or receive the data from the other side through socket, I will use some Windows APIs (in Windows) or syscall (in Linux), but normally using it will lead to some unexpected results. These consequences are because of the mechanism of the function which is if we want to send, saying, 1000 bytes then It may only send 500 first bytes and return the number of sent bytes. Therefore, I have defined two functions based on the Windows APIs (or syscall in Linux), one for sending and the other for receiving.

The sending function names `sendAll`. This function will have the input arguments almost the same as the Windows API (or syscall in Linux) which is the address of the source data to send, the number of bytes that will be sent and the socket file descriptor to know the sending destination. It will return 0 on success or 1 in other cases. This sending function will ensure that all the required data will be sent. The flowchart of the function will be as follow:

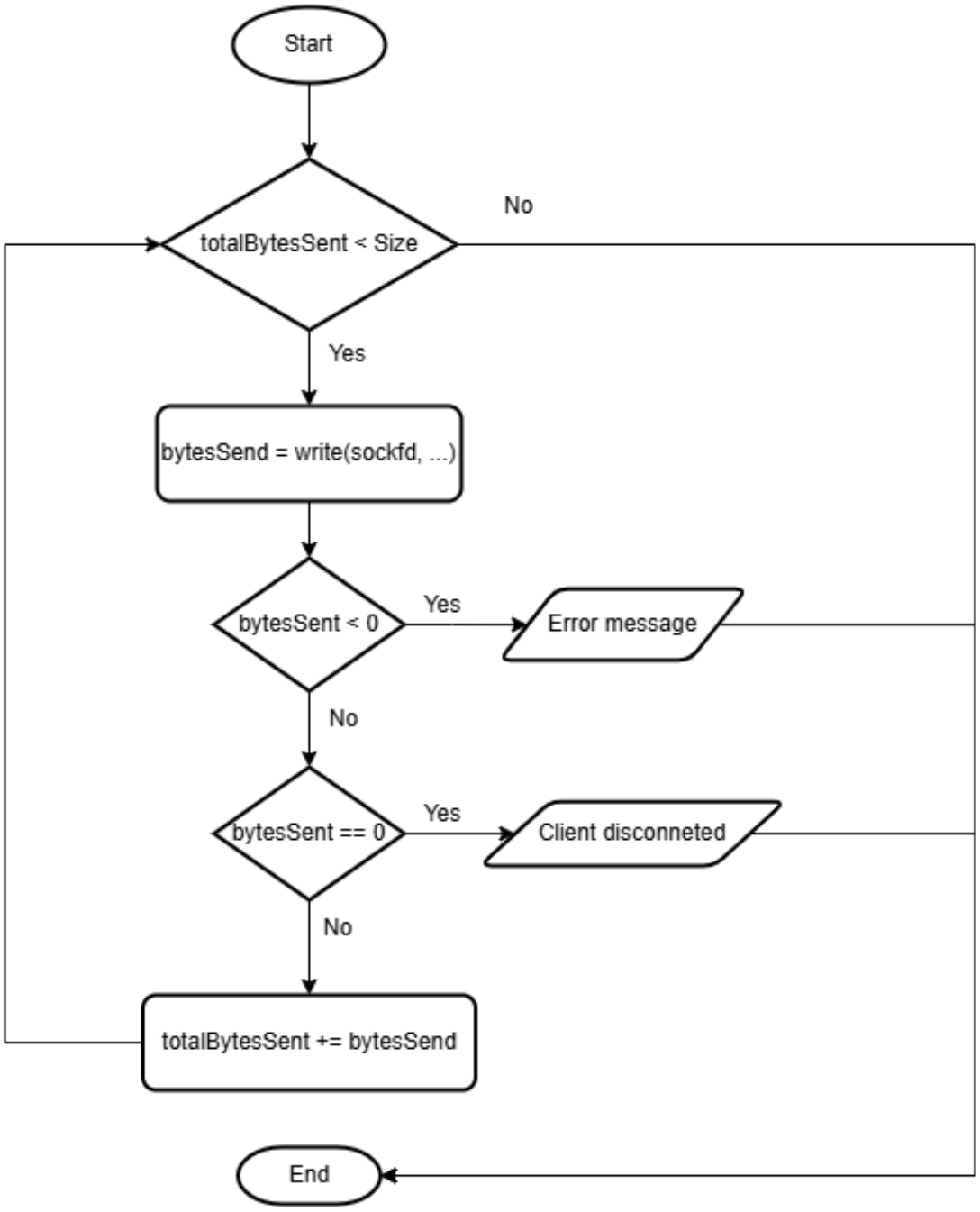

Fig. 4. Sending Flowchart

For receiving, I have defined a function named `receiveAll`. This function also have the input arguments almost the same as the Windows API (or syscall in Linux) which is the address of the destination place to store data, the number of bytes will need to receive and the socket file descriptor to know the receiving destination. It will return 0 on success or 1 in other cases. This receiving function will ensure that all the required data will be received. The flowchart of the function will be as follow:

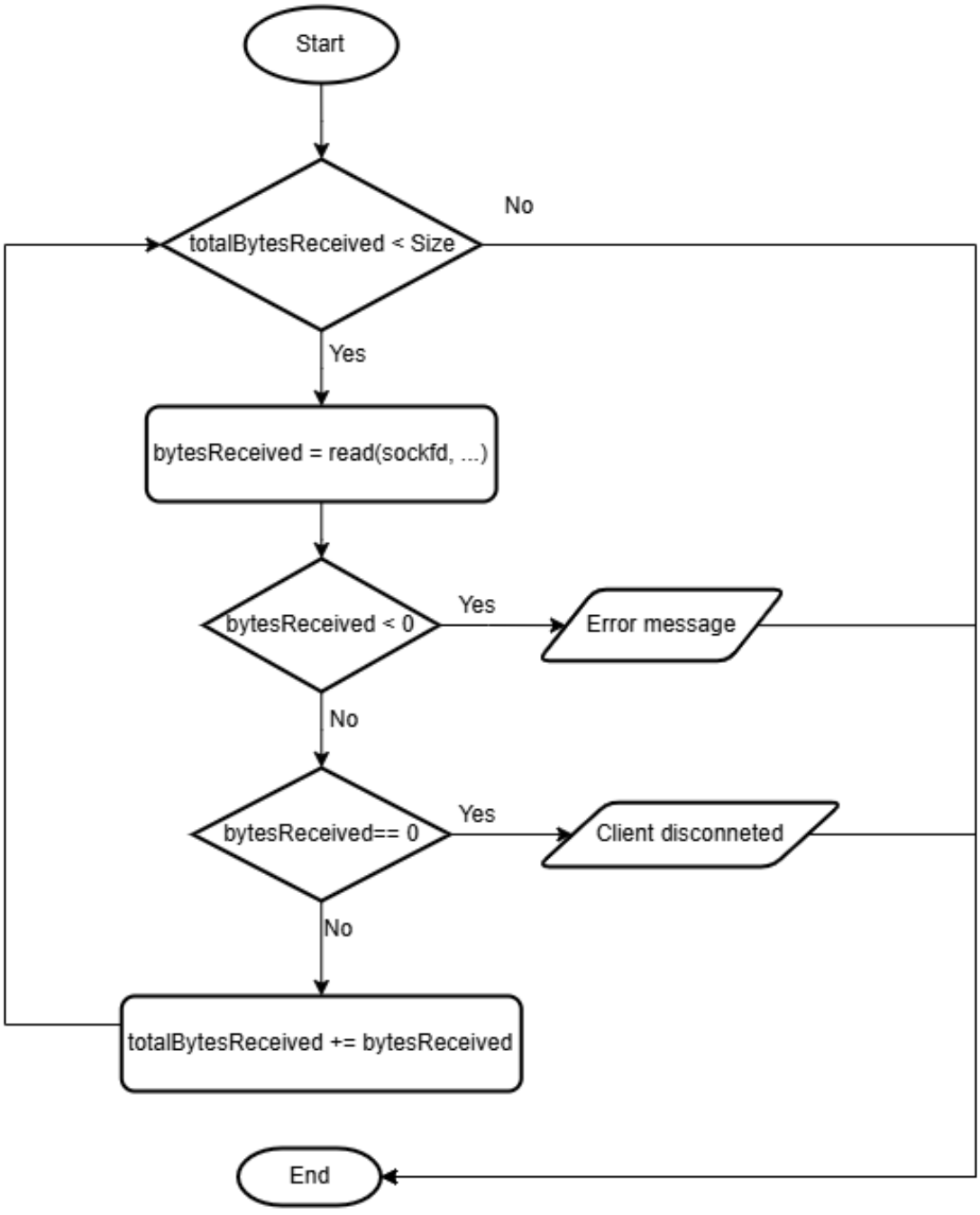

Fig. 5. Receiving Flowchart

3.3.1 Server Side

As I have already mention in the previous section that the main job of the server is to send dumping signal to clients and receiving the data sent back from clients. Since the client will send the extracted information in term of directory and file, therefore I need to handle this problem by creating two new functions. One is responsible for handling file sent from clients and the other is responsible for handling directory sent from clients.

The `writeFile` function is responsible for receiving a file from a socket connection and saving it to the specified directory. It starts by constructing the full path for the new file and attempts to open it for writing. If the file cannot be created, it prints an error message, closes the socket, and returns an error code. The function then enters a loop where it repeatedly receives chunks of data from the socket until the entire file is received. For each chunk, it writes the data to the file and updates the remaining bytes to be received. After successfully receiving and writing all the data, the function closes the file and returns a success code. If any errors occur during the data reception, the function returns an error code immediately. The flowchart of the function will be as follow:

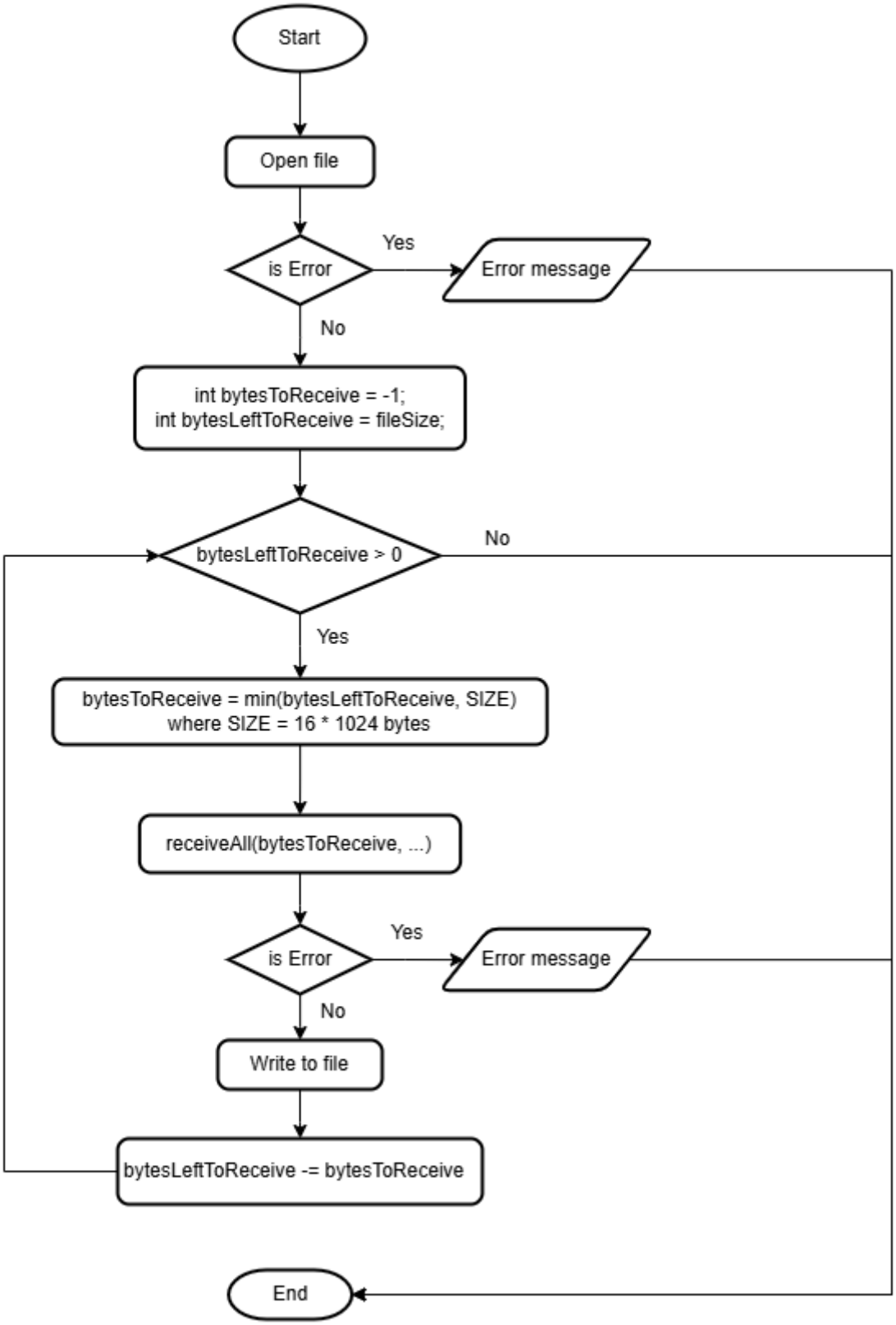

Fig. 6. File Receiving Handling Flowchart

In case the client send a directory, I have defined a function named `writeDir`. The function is responsible for handling directory from client. It starts by initializing buffers and receiving a message from the socket. If the message indicates success, it proceeds to receive additional data into a `box` structure. Depending on the type of the received data (file or directory), it either calls the `writeFile` function to handle file reception or creates a new directory. If a directory is to be created, it constructs the new directory name based on certain conditions and attempts to create it, handling errors appropriately. After creating the directory, it enters a loop to handle nested files and subdirectories by recursively calling itself until all entries are processed. If any errors occur during data reception or directory creation, the function returns an error code. The flowchart of the function will be as follow:

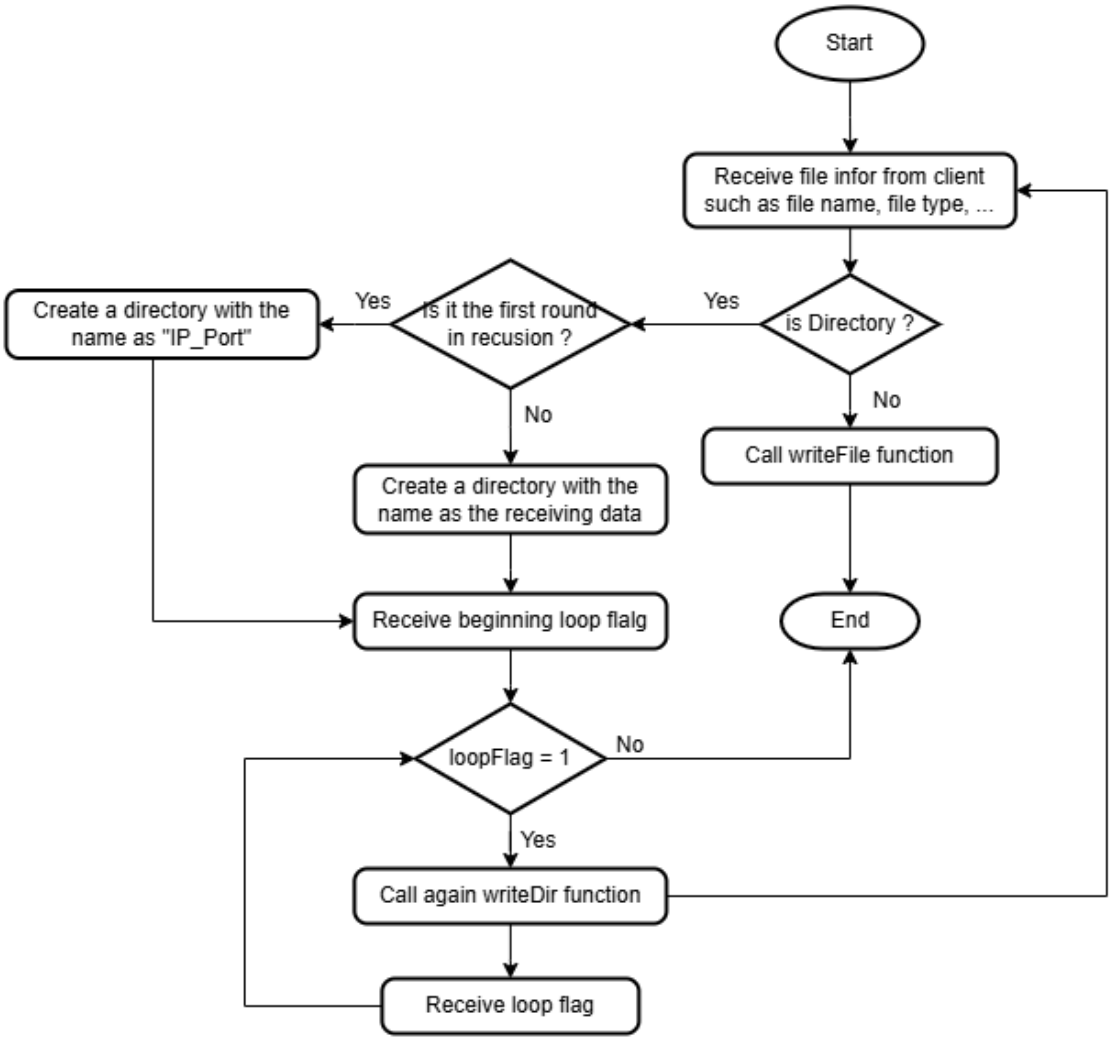

Fig. 7. Directory Receiving Handling Flowchart

3.3.2 Client Side

If the the main task of the server side is to send signal and receive data from clients, the main task of the client side will be to receive signal and send back extracted data. Since the extracted data will be stored in multiple files and directories, therefore, existing functions that can handle it is important.

To send a file from client to server, I have defined a function name `sendFile` to handle this situation. The `sendFile` function is responsible for sending a specified file over a socket connection. The function begins by attempting to open the specified file in read-binary mode. If the file cannot be opened, an error message is displayed, the socket is closed, and the function terminates. It then initializes a buffer for reading data from the file. The function enters a loop where it reads chunks of the file into the buffer, and then sends these chunks over the socket using the `sendAll` function. This process continues until the entire file is read and sent. After successfully sending the file, the file is closed, and the function exits. Any errors encountered during the sending process are handled by returning early and performing necessary cleanup. The flowchart of the function will be as follow:

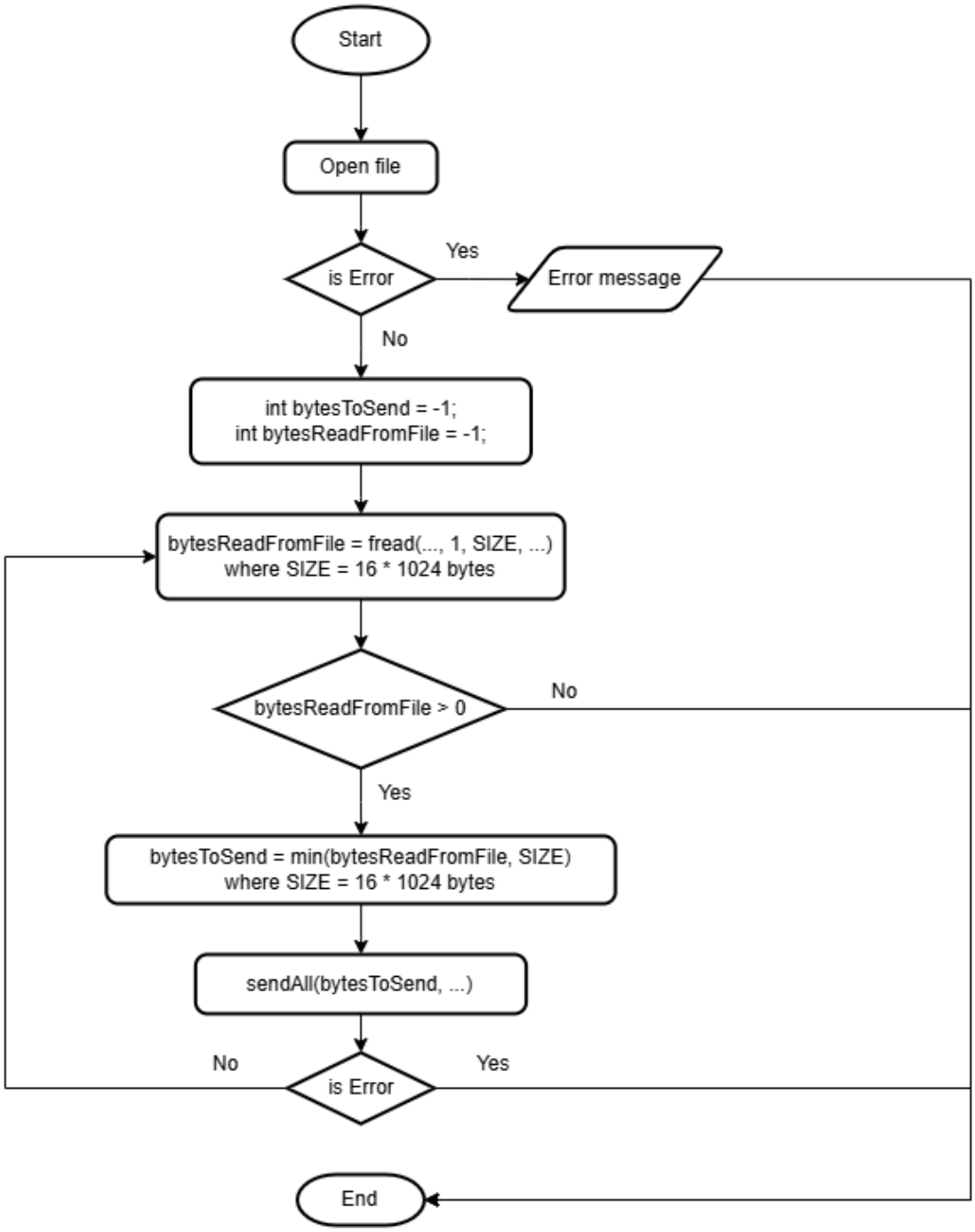

Fig. 8. File Sending Handling Flowchart

Since each time the client extracts data, it creates many files. Therefore placing files into a directory and sending it are a good ideal. The `sendDir` function is responsible for sending the contents of a directory over a socket connection. It starts by initializing a message buffer and a `box` structure for holding file or directory metadata. The function retrieves and verifies the file attributes of the specified path. If the path is invalid, an error message is sent over the socket, and the function terminates. For regular files, it obtains the file size, fills the `box` structure, and sends this metadata to the receiver. The actual file contents are then sent using the `sendFile` function. For directories, it constructs the full path for each file or subdirectory, and recursively calls `sendDir` to handle nested contents. It manages directory navigation and ensures the sender returns to the original directory after processing. Throughout, the function handles errors by closing the socket and performing necessary cleanup, ensuring the integrity of the file transfer process. The flowchart of the function will be as follow:

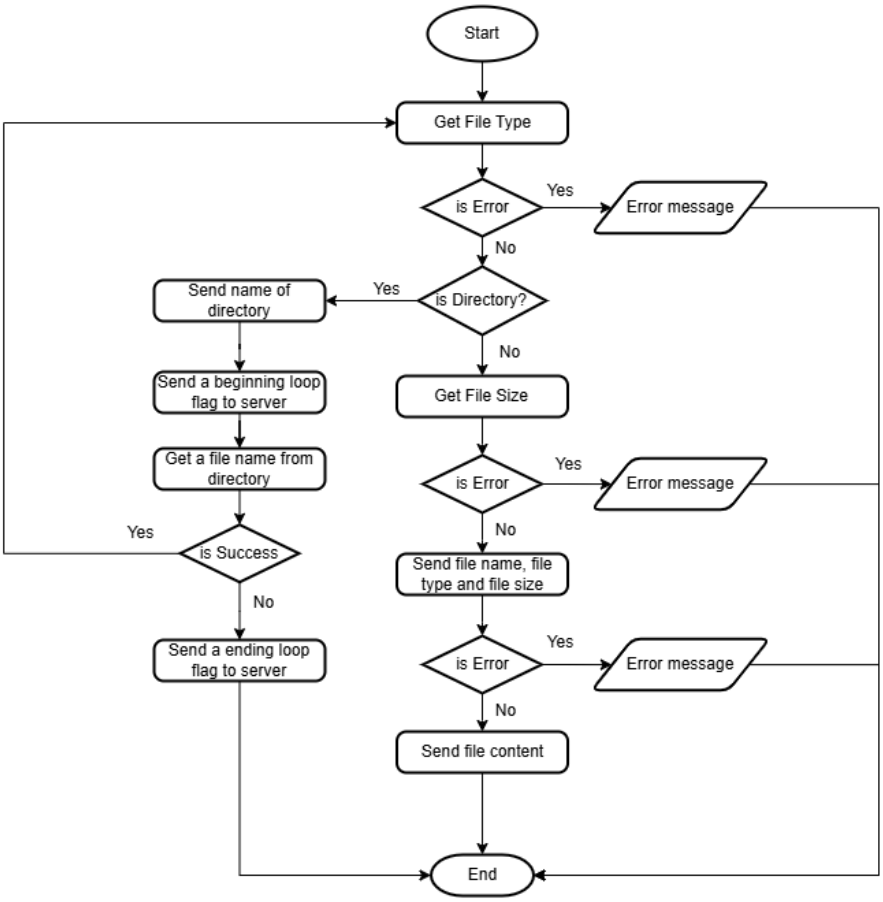

Fig. 9. Directory Sending Handling Flowchart

3.4 Storing Dumping Data

3.4.1 Server Side

When the server receives data extracted from clients, a storing mechanism that can help managing the data effectively and systematically is important. To do so, I will store the receiving data sent from clients as follow. For each new client connects to, the server will first check if the directory has already existed. Otherwise, server creates a new unique directory for storing all the data sent from the client. The name of the folder will be set according to the pattern `IP_Port`. Example a client with IP of 192.168.111.1 and connecting from port 5787, then server will create a new directory named `192.168.111.1_5787` in the current working directory for storing all data sent from that client. From now on, each time the client sends back data, server store it in this folder.

3.4.2 Client Side

With the client, as soon as it starts running, it check whether a directory with the user-specified name already exists. Otherwise, client creates a new directory named according to the user's wishes in the current working directory. This directory will be used to store all the future directories and files that the client extracted. Each time receiving the dumping signal from server, it tries retrieving the current timestamp by calling a function named `get_current_timestamp`. This function starts by retrieving the current system time and converting it into a formatted timestamp string. Initially, it fetches the current time and converts it to a time representation. Then, it formats the time into a string formatted as `_YYYYMMDD_HHMMSS`(Year-Month-Day-Hour-Minute-Second). Finally, it returns this formatted timestamp string, which is used for generating unique timestamps for file naming.

```
TString get_current_timestamp()
{
    // Get current time
    std::time_t now = std::time(nullptr);
    std::tm tm;
    localtime_s(&tm, &now);

    // Create a timestamp string
    Tostringstream timestamp;
    timestamp << std::put_time(&tm,
                               TEXT("_%Y%m%d_%H%M%S"));

    return timestamp.str();
}
```

After successfully retrieving current timestamp, the client goes into the previous newly created directory and creates a new one named according to the following pattern `log_curent_timestamp`. This directory will be used to store all the dumping data related to this signal. All the file extracted in this turn also have the name ending with `_YYYYMMDD_HHMMSS`.

3.5 Extracting RAM

To extract RAM of the client, I have utilizes an open source tool which is `WinPmem`. To integrate the tool with my project, I have defined a function named `dump_ram` which leverage its interface. The function is designed to dump the physical RAM into a raw image file. It begins by constructing an output filename based on the provided directory path (dir) and timestamp (timestamp). This filename is formatted to include the directory path, prefix `"ramDump"`, the timestamp, and the `".raw"` extension. The function then initializes variables and objects necessary for interfacing with the `WinPmem` tool, specifically creating an instance of `WinPmem`. Next, it proceeds to install the driver, set the acquisition mode to physical memory `PMEM_MODE_PHYSICAL`, and initiates the writing of the raw memory image. Finally, it uninstalls the driver, deallocates memory, and returns an integer status code indicating the success or failure of the operation. The flowchart of the function will be as follow:

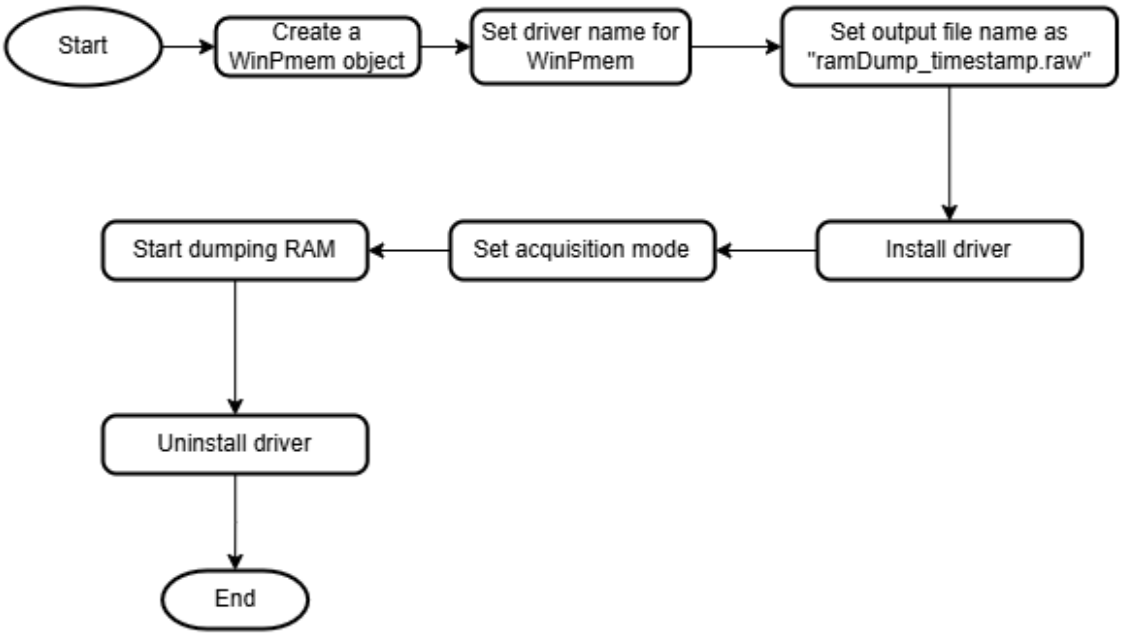

Fig. 10. Extracting RAM Flowchart

3.6 Extracting Registries

Registry are considered the database of the Windows operating system, it stores many important information such as licence, key, application configuration... Therefore, retrieving those are crucial in forensic investigation. In this project, to receiving these registries, first, I intended to used an utility available in Windows to extract it, but it only return in text format. Therefore, I have create a function named `dump_registries` to perform the task. It a simple function that extracts registry keys `HKEY_CLASSES_ROOT`, `HKEY_CURRENT_USER`, `HKEY_LOCAL_MACHINE`, `HKEY_USERS` and `HKEY_CURRENT_CONFIG` one by one. The flowchart of the process will be as follow:

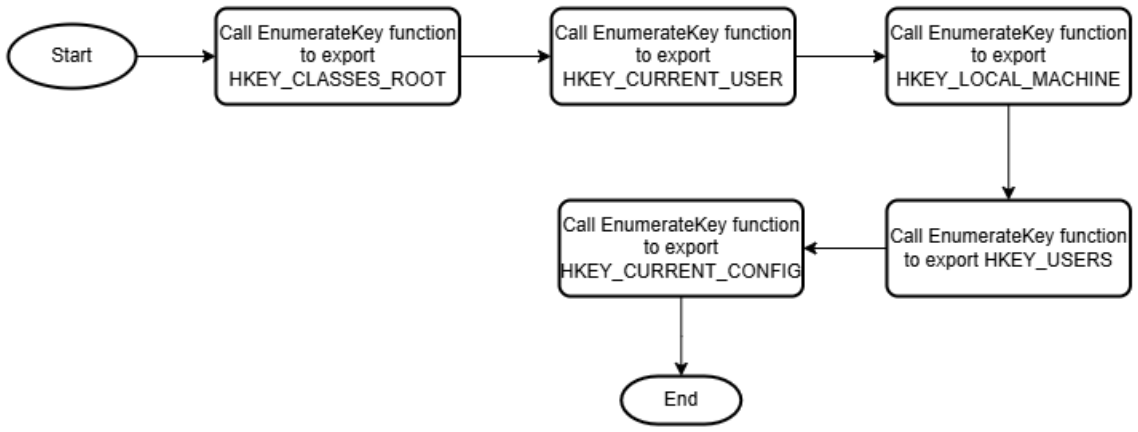

Fig. 11. Extracting Registries Flowchart

The key different between this function and the utility available on the Windows is that it can parse the result into a json file. To perform this task, I have defined a function named `EnumerateKey`. The `EnumerateKey` function recursively enumerates and outputs the structure of a Windows registry key and its subkeys and values to a specified file. It begins by opening the specified registry key and, if successful, writes the key name to the output file. It then enumerates all subkeys and recursively calls itself to process each subkey, formatting the output as JSON. After handling subkeys, the function enumerates all values within the current key, including their names, types, and data. Different registry data types such as strings, binary data, and DWORD values are processed and formatted appropriately. Once all subkeys and values are processed, the function closes the registry key and finishes writing to the output file. The flowchart of the parsing process will be as follow:

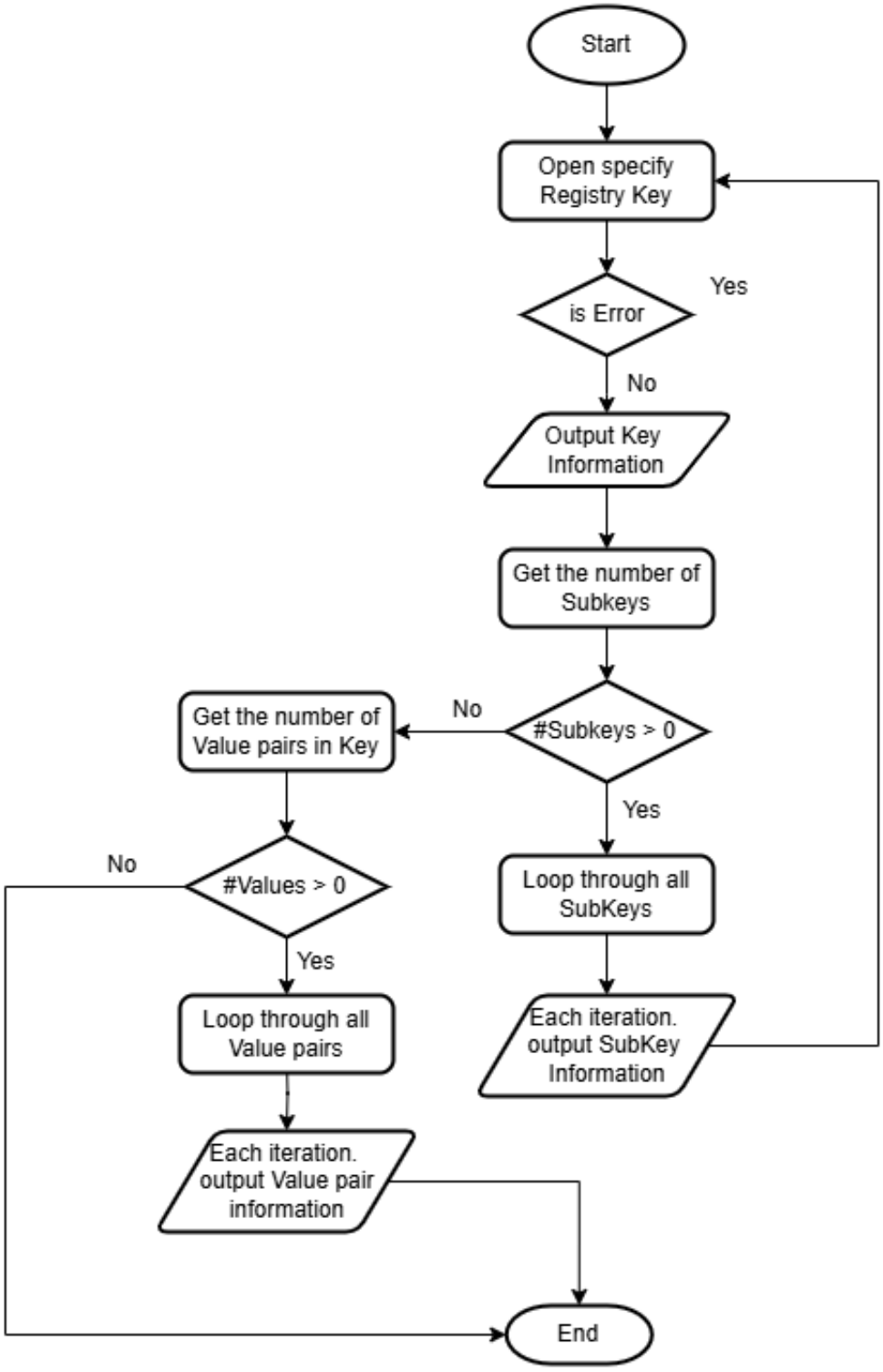

Fig. 12. Parsing Registry Flowchart

3.7 Extracting Processes

To extract the current running process from the compromised network nodes, I have create a function named `dump_process_info` to meet the task. The `dump_process_info` function is designed to collect and save detailed process information from the system into a CSV file. The function constructs a filename using the provided directory and timestamp. It then builds a command string to execute the `tasklist` command with verbose output in CSV format, redirecting the output to the constructed file path. This command is executed using the system function, which runs the command in the system's commandline environment. After the command executes, a message is printed to the console, indicating the successful completion of the process information dump and specifying the location of the output file. This function provides a simple but effective way to capture current process information for later analysis. The source code of the function will be as follow:

```
void dump_process_info(char* dir,
                                    char* timestamp)
{
    std::string fileName = "\\Process_info_dump"
                           + std::string(timestamp)
                           + ".csv";
    std::string command = "tasklist /v /FO csv > ";
    command += dir;
    command += fileName;

    system(command.c_str());
    printf("Dumping process complete successfully:
            %s\n", (dir + fileName).c_str());
}
```

Each process output from the function will have nine related attributes. The name and the brief description of each attribute is in the below table.

Field Name	Description
Image Name	The name of the executable file of the process.
PID (Process Identifier)	A unique number assigned to each process when it is created.
Session Name	The name of the session in which the process is running.
Session#	The numerical identifier of the session.
Mem Usage	The amount of memory in kilobytes (KB) that the process is currently using.
Status	The current status of the process (e.g., Running, Not Responding).
User Name	The name of the user who owns the process.
CPU Time	The amount of CPU time that the process has used since it started.
Window Title	The title of the window associated with the process, if applicable.

Table 1: Fields and descriptions of extracted process information

3.8 Extracting Network Connections

To extract the network information from the compromised network nodes, I have defined a function named `dump_net` to perform the task. The `dum_net` function generates and saves network-related statistics to specified files. It creates a JSON file containing network adapter statistics using PowerShell commands `Get-NetAdapterStatistics` and saves it in the provided directory with a timestamp. Additionally, it captures network connection statistics using the `netstat` command, saves this data to a text file, and then reads and parses the text file to convert the data into JSON format. This JSON data is subsequently saved to a file in the same directory. The function outputs success messages for each completed operation. The flowchart of the process will be as follow:

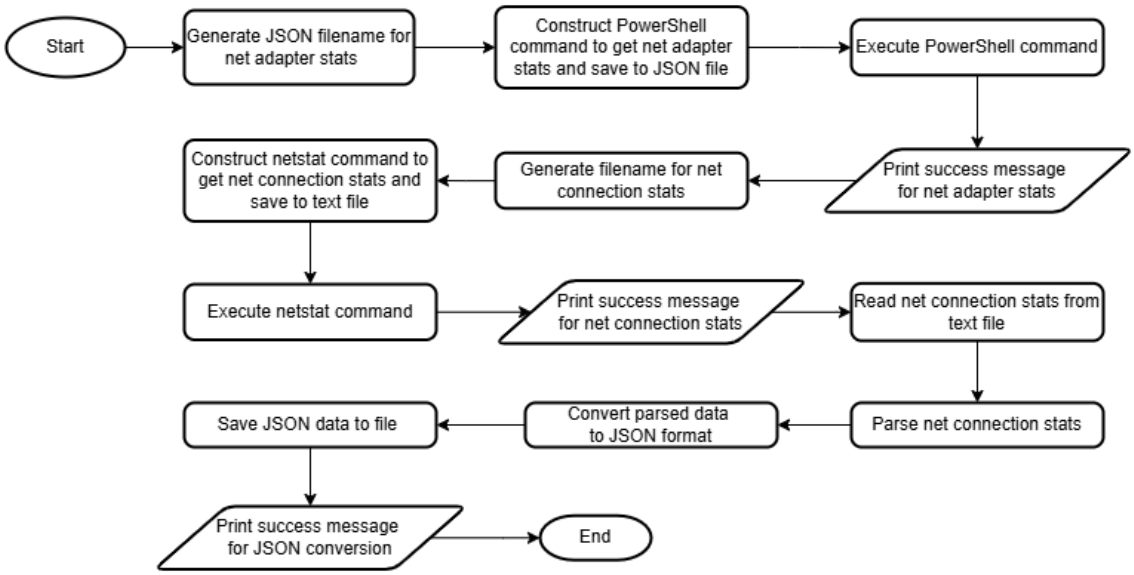

Fig. 13. Extracting Network Flowchart

Since the result returns from the `netstat` command is in text format. Therefore, I have created a function named `parseNetstatOutput`. To make the process easier, I have also created a structure named `NetStatInfo`. This struct represents all the related information of one network connection in the compromise network node. The `parseNetstatOutput` function processes the output of a `netstat` command to extract network connection information. It first skips the header lines and then parses each subsequent line to extract relevant details such as protocol, local address, foreign address, state, and process ID (if available). These details are stored in `NetStatInfo` objects, which are then collected into a vector. The function returns this vector of `NetStatInfo` objects, which represent the parsed network connection statistics.

```
struct NetStatInfo
{
    TString protocol;
    TString localAddress;
    TString foreignAddress;
    TString state;
    TString pid;
};
```

The meaning of each element will be describe in the below table.

Field Name	**Description**
protocol	The protocol used for the connection (e.g., TCP, UDP).
localAddress	The local IP address and port number.
foreignAddress	The remote IP address and port number.
state	The state of the connection (e.g., LISTENING, ESTABLISHED).
pid	The process ID associated with the connection, if available.

Table 2: Description of each element in the `NetStatInfo` struct.

4 Deployment and Testing

4.1 Deployment

In this project, both the client and server are all portable executable. It means that you do not need to require any further library. The only things need to deploy both server and client are the file executable itself.

4.1.1 Server

The server is designed to run in the a Linux operating system machine. Therefore, to run the server, you need a Linux machine.

If you want to build the program from the source, find the file named `server.cpp` in the `server_linux` folder and execute the following command:

```
g++ server.cpp -lpthread -o server
```

The server requires two arguments. The first one is the port (`-p, -port`), you need to specify the port that the server listens on. The second argument (`-i, -interval`) is the interval. This argument will specify the period of time that the dumping signal will be sent again after the previous time. Additionally, the server also has another help argument (`-h, -help`) that briefly describe how to run the server.

```
[+] Usage: ./fileName [OPTIONS]
[+] Description: Execute command and Return result
                 to the client.
[+] Options:
    -h, --help:     Display the usage of the program.
    -p, --port:     Specify the port number the server
                    uses to receive connections
                    from clients.
    -i, --interval: Specify the interval in second
                    that the server sends a dumping
                    signal to clients.
```

4.1.2 Client

The client is designed to run in a Windows operating system machine. Therefore, to run the client, you need a Windows machine system. Additionally, because the client need to extract RAM and registries, the client need to be run in `Administration` mode.

If you want to build the client from source code, open the project in `InfoDumping` folder in Visual Studio, choose `Debug`, `x64` and build it again. In this project, I was used the Visual Studio 2022 version 17.6.2

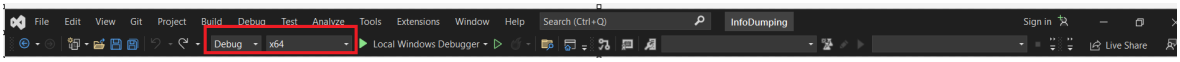

Fig. 14. Configuration

The client requires 3 arguments. The first one is the IP address of the server (`-i`) so that it knows where to connect to. The second one is the port of the server (`-p`), the purpose of this argument is the same as the previous one. The last one is the name of the folder (`-d`)where storing all the dumping data that the client extracted. Additionally the client will default display its usage if you do not enter any argument or the entered arguments are incorrect.

```
[+] Usage: \InfoDumping.exe [option]
[+] Option:
            -i: Specify the server IP address.
            -p: Specify the server port.
            -d: Specify the dump directory name.
```

4.2 Testing

4.2.1 Server Testing

(a) Case 1: When starting the server with full required arguments, the server will run normally and display a message to inform the server listening IP and port. Observing the below result, concluding that the test is passed.

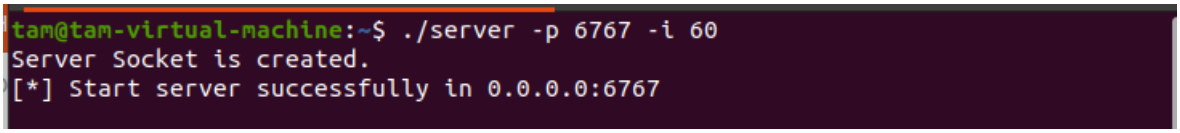

Fig. 15. Full Required Arguments - Server

(b) Case 2: When starting the server, if the required arguments are not provided or provided wrong argument, the server will display a message to inform the user that some required argument are missing. Observing the below result, concluding that the test is passed.

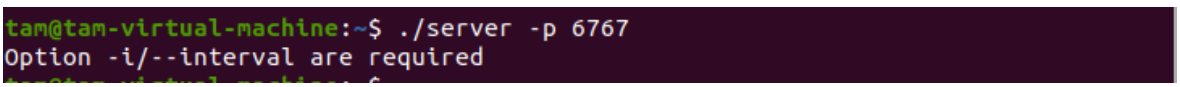

Fig. 16. Missing IP Argument - Server

Fig. 17. Missing Port Argument - Server

```
tam@tam-virtual-machine:~$ ./server fasdaf
Option -p/--port are required
```

Fig. 18. Wrong Argument - Server

(c) Case 3: To test multithread functionality of the server. I will run a server on `192.168.245.132:6767` and use 3 clients trying to connect to. The first client with the IP and port `192.168.245.1:55848`, the second will run on `192.168.245.157:50546` and the third which runs on the same machine with the second on `192.168.245.157:50526`. The server will display the success connection for each client and after the clients send data, a new folder for each one will be created. Observing the results, concluding that the test is passed.

```
tam@tam-virtual-machine:~$ ./server -p 6767 -i 60
Server Socket is created.
[*] Start server successfully in 0.0.0.0:6767
[+] New client connect from 192.168.245.157:50526
[+] New client connect from 192.168.245.157:50546
[+] New client connect from 192.168.245.1:55848
```

Fig. 19. Multi-connection 1

```
drwxrwxr-x 3 tam tam   4096 Thg 6  19 14:39 192.168.245.1_55848
drwxrwxr-x 3 tam tam   4096 Thg 6  19 14:31 192.168.245.157_50526
drwxrwxr-x 3 tam tam   4096 Thg 6  19 14:32 192.168.245.157_50546
```

Fig. 20. Multi-connection 2

```
tam@tam-virtual-machine:~/192.168.245.157_50526/log_20240619_082720$ ls -l
total 5378444
-rw-rw-r-- 1 tam tam        715 Thg 6  19 14:31 HCC_registry_dump_20240619_082720.json
-rw-rw-r-- 1 tam tam   33277276 Thg 6  19 14:31 HCR_registry_dump_20240619_082720.json
-rw-rw-r-- 1 tam tam    3550380 Thg 6  19 14:31 HCU_registry_dump_20240619_082720.json
-rw-rw-r-- 1 tam tam   95685678 Thg 6  19 14:31 HLM_registry_dump_20240619_082720.json
-rw-rw-r-- 1 tam tam    6196286 Thg 6  19 14:31 HU_registry_dump_20240619_082720.json
-rw-rw-r-- 1 tam tam      43204 Thg 6  19 14:31 net_adapter_stats_20240619_082720.json
-rw-rw-r-- 1 tam tam      13722 Thg 6  19 14:31 netstat_20240619_082720.json
-rw-rw-r-- 1 tam tam       6692 Thg 6  19 14:31 netstat_20240619_082720.txt
-rw-rw-r-- 1 tam tam      16526 Thg 6  19 14:31 Process_info_dump_20240619_082720.csv
-rw-rw-r-- 1 tam tam 5368709120 Thg 6  19 14:34 ramDump_20240619_082720.raw
```

Fig. 21. Received Data

(d) Case 4: To test the schedule functionality of the server. I will set the server so that it will resend the dumping signal to the clients after 60 seconds since it finishes receive all the data of the first time. Inside the folder associated with the client will, now, have one more folder storing data for the second dumping signal. Observing the result, concluding that the test is passed.

Fig. 22. Data After the First Signal

Fig. 23. Data After the Second Signal

(e) Case 5: When a client disconnects during the connection, the serve will discard that connection and display a message to inform the event. Observing the result, concluding that the test is passed.

```
[-] ERROR: Connection reset by peer
[+] Disconnected the client from 192.168.245.157:50819
```

Fig. 24. Client Disconnection

4.2.2 Client Testing

(a) Case 1: When starting the client with full required arguments, the client will run normally and display a message to inform the event. Observing the below result, concluding that the test is passed.

```
PS C:\Users\skado\Downloads> .\InfoDumping.exe -i 192.168.245.132 -p 6767 -d machine3
Directory created successfully!
[+] Waiting for dumping signal
```

Fig. 25. Full Required Arguments - Client

(b) Case 2: When starting the client, if the required arguments are not provided, the client will display the usage message. Observing the below result, concluding that the test is passed.

```
PS C:\Users\skado\Downloads> .\InfoDumping.exe afdfa
[+] Usage:  C:\Users\skado\Downloads\InfoDumping.exe [option]
[+] Option:
          -i: Specify the server IP address.
          -p: Specify the server port.
          -d: Specify the dump directory name.
```

Fig. 26. Wrong Argument - Client

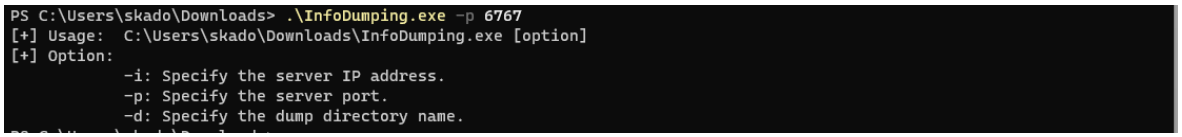

Fig. 27. Missing Argument - Client

(c) Case 3: When the server disconnects during the connection with client, the client will handle the disconnection and display a message to inform the event. Observing the result, concluding that the test is passed.

Fig. 28. Server Disconnection

5 Conclusion

In conclusion, the goal of developing a tool for dumping needed information from compromised network nodes has been achieved successfully. In particular, I have created a server running on the Linux operating system that can handle multiple threads which will allow to serve many clients at the same time. The server can also make a schedule to send dumping signal for client after specified period of time. The server was also built to store receiving data in a separate folder for each client so that the investigator can easily use it. As for the client's part which is designed to run on the Windows operating system, it is capable of establishing a connection with the server, receiving signals, and extracting essential data such as RAM contents, current running processes, system registry, and network connection information. Additionally, I also created some APIs by creating an `InfoDumping` class so that the other can reuse my project by importing this class and calling its methods. There are one more things that I have not achieved in this project, that is, extracting the name of the process related to network information. I have found out how to get the process name of the network information but the method only exports the text format and I have not found a way to parse it. Therefore I will let this parse for future work.

Throughout this project, I have gained substantial knowledge and practical experience in several key areas. I have learned about the complexities of network security and the challenges investigators face when dealing with cyberattacks. The implementation of the server-client model has improved my understanding of network communications and the complicated problems involved in handling multiple connections. Working on how to retrieve the needed information has given me the knowledge about registry in Windows operating system, the information that a process owns, and the connection information in a computer... Moreover, the process of developing an API for client has improved my skills in creating software components.

Despite the achievements in this project, there are many things to do in the future. First, I will try to enhance the information associated with a network connection. For example, as discussed above, adding the process name for the network connection by running `netstat` command with the argument `-anob`. This will give the process name. Then I will try to find a way to parse the result to return json output. The next thing I will do is to integrate my client with an Intrusion Detection System (IDS). I think I can do this by improving my server so that it can connect to the IDS and receive signal from it. As soon as it receving the alert signal, the server will send the dumping signal to clients to begin dumping. Another way is to leverage my available API of the client. The IDS will directly send signal to client through its APIs, then client will start the dumping process. Overall, the knowledge gained by this project provide a solid foundation for my future development and innovation in security tools.

Reference

1. https://vtv.vn/cong-nghe/hon-2300-cuoc-cuoc-tan-cong-mang-vao-cac-he-thong-thong-tin-tai-viet-nam-20240406053256762.htm
2. https://ncsgroup.vn/en/home/
3. https://www.crowdstrike.com/cybersecurity-101/endpoint-security/endpoint-detection-and-response-edr/
4. https://www.trellix.com/security-awareness/endpoint/what-is-endpoint-detection-and-response/
5. https://www.ibm.com/topics/siem
6. https://volatilityfoundation.org/about-volatility/
7. https://en.wikipedia.org/wiki/Thread_(computing)
8. https://www.linkedin.com/pulse/threadprocess-linux-amit-nadiger#:~:text=In%20Linux%2C%20threads%20are%20implemented,other%20more%20easily%20and%20efficiently.
9. https://cloud.z.com/vn/en/news/socket/
10. https://wiki.matbao.net/socket-la-gi-khai-niem-can-biet-ve-giao-thuc-tcp-ip-va-udp/
11. https://www.fortinet.com/resources/cyberglossary/tcp-ip#:~:text=What%20does%20TCP%20mean%3F,data%20in%20digital%20network%20communications.
12. https://tenten.vn/tin-tuc/file-system-la-gi/
13. https://www.techtarget.com/searchstorage/definition/file-system
14. https://www.avast.com/c-what-is-ram-memory#:~:text=RAM%20is%20a%20temporary%20memory,to%20complete%20immediate%20processing%20tasks.
15. https://medium.com/@tojopthomas/winpmem-an-open-source-memory-acquisition-tool-2b25c6eddeab
16. https://en.wikipedia.org/wiki/Process_(computing)
17. https://en.wikipedia.org/wiki/Windows_Registry
18. https://man7.org/linux/man-pages/man3/pthread_create.3.html