Bulgarian Diploma Thesis

Cheating Prevention Software for Online Exams

Vitaliy Konyukhov, ID# 200086768

Student: _____ **, Date:** _____
signature

Supervisor: _____ **, Date:** _____
signature

Department of Computer Science, AUBG

Blagoevgrad, 2021

Title: Cheating Prevention Software for Online Exams

Author: Vitaliy Konyukhov

Abstract: Online exams are a big opportunity for students to cheat. The COVID pandemic forced institutions to switch to online teaching, thereby increasing the occurrences of cheating. This senior project aims at solving this issue by providing the proctors with additional information about students' activities. It consists of two applications, one used by students and the other – by proctors. Students' application (the Client) collects various metrics of the operating system such as opened windows or running applications and sends them to the proctor's Server application, which analyzes the data, as well as displays and records any suspicious activity. This application simplifies the proctoring process and makes it more thorough at the same time. The applications are created using Qt Framework with C++ Programming Language and support multithreading. The communication is done over a custom network protocol with TCP using Windows Sockets and secured using the TLS protocol from the OpenSSL library.

Declaration of authorship:

"The Senior Project/Bulgarian Diploma Thesis presented here is the work of the author solely, without any external help, under the supervision of Prof. Vladimir Georgiev. All sources, used in development, are cited in the text and in the Reference section."

Author: _____

0. Table of Contents

1. Introduction

The idea for my Senior Project came from discussions about online education, which is an extremely convenient way of education since it is available wherever there is an internet connection. However, one of the downfalls of online education is that the instructors cannot properly proctor tests and exams online. Since my university uses Zoom for videoconferences, which does not provide enough information about the student's actions on the computer, I have decided to implement a software package that will expand the scope of Professors' ability to proctor online exams and will simplify their job by highlighting suspicious activity on the test-taker's computer.

1.1. The Problem of Cheating on Online Exams

After the COVID pandemic, most institutions moved their classes online, and, consequently, all the tests switched online as well. Students are much more likely to cheat on online tests than in person. The proctor responsible for the supervision of tests has a very limited view of the environment of the test-taker. In the case of using Zoom, the proctor can only get the webcam video feed, sound from the microphone, and the video of the shared screen. This opens multiple possibilities up for students to cheat during online exams.

Of course, it is impossible to predict all different ways of cheating on online exams, but they include the following: opening banned applications (internet browsers, presentations, etc.), use of multiple monitors, use of remote-control software, use of a virtual machine or a remotely controlled computer. However, there are ways that are impossible to detect with software such as the use of external devices, the use of an impersonator, or cheat sheets on paper, but this project will not take those into account.

Due to the pandemic, online examinations are often not a choice but a necessity. In-person exams are either prohibited or dangerous for the test-takers and it is important to preserve the integrity of examinations. There is no value in grades that students receive if they cheated since their preparedness for the subject is not properly evaluated. To prevent the compromise of online exams' fairness, some universities

and other education institutions decide to use specialized proctoring software which mitigates many cheating possibilities. However, this not only adds additional complexity to the exams but also comes with high costs for the institutions.

1.2. Existing Proctoring Solutions

There are multiple software solutions designed to proctor online examinations and maintain the integrity of tests. For example, ExamOnline, ProctorU, and Proctortrack.

ExamOnline uses artificial intelligence for proctoring and automatically verifies the identity of test-takers with facial recognition. It can track the facial movements to properly monitor the test-taker. It can detect any restricted objects with the camera and any applications on the computer. It covers almost all possible cheating possibilities and offers an easy and comfortable experience of taking tests.

ProctorU uses live proctors to constantly monitor the activity of test-takers. It verifies the identity of the person taking an examination. It detects any suspicious behavior using artificial intelligence. It monitors the activity on the screen, on the camera, and the sound of the microphone. The functionality offered is more than enough to conduct a proctored examination.

Proctortrack allows to automatically verify the identity of the person, proctor the examination with artificial intelligence, monitor and analyze the activity. It is secure, easy to use, and provides sufficient control of cheating during exams.

From the examples, we can see that existing solutions are automated, offer many features, and detect cheating with high certainty. However, all of them are commercial solutions. They require arrangements and contracts before they can be implemented. Every test needs to be arranged in advance, putting additional considerations and planning onto professors. The price of using existing proctoring software for the universities is high, especially during the pandemic. If we consider the number of students, the number of classes, and the number of various quizzes and examinations in each class, the price becomes substantial.

1.3. Cheating Prevention Software

The software package developed in this Diploma Thesis aims at providing a substitute for the proctoring software using the existing university infrastructure. Since Zoom is already widely used for proctoring through its webcam, screen, and microphone sharing capabilities, the software package should complement it by providing additional data (metrics) collected from the test-taker's computer to the proctor and analyzing it.

Cheating prevention software consists of two applications: a server and a client. The client application is intended to be installed at the test-taker's computer and the server application – at the proctor's computer. Both applications display the relevant information to the user through a graphical user interface, which is implemented with the Qt toolkit. All the code is written in C++ using Qt Creator integrated development environment (IDE). The existing applications are intended for use on Windows only.

The server and the client can communicate through the internet or local network. The networking capabilities are implemented with Windows Sockets 2 (Winsock) provided by the ws2tcpip.h header. The client application collects the metrics on the computer and sends them in the form of text to the server using TCP/IP protocol. The connection is encrypted with TLSv1.3 using the OpenSSL library. The server application receives the metrics from all connected clients, displays them to the proctor in the GUI, evaluates them to understand the behavior of the test-taker, and highlights the suspicious activity.

The metrics that are collected include the active (foreground) window's title and process ID, whether the user is using a remote-controlled machine or a virtual machine, the number of monitors that the computer has, the manufacturer and the model of the computer, and the list of all running applications. The server uses a list of banned applications and checks if the running applications include any prohibited software (such as remote-control software). Since the window's title usually includes sufficient information about the activity in that window, the server compares the title of the

window with the list of suspicious keywords to predict cheating behavior and to highlight the student for the proctor. The keywords can be added or removed from the list by clicking on a student’s window’s title shown in the GUI.

2. Specification of the Software Requirements and Their Analysis

Since the project consists of two applications, the Client and the Server, the software requirements are specified for each application individually. For clarity purposes, the user of the Client application will be called the student, and the user of the Server application – the proctor. The following are the functional and non-functional requirements of the applications and their analysis.

2.1. Functional Requirements of the Client Application

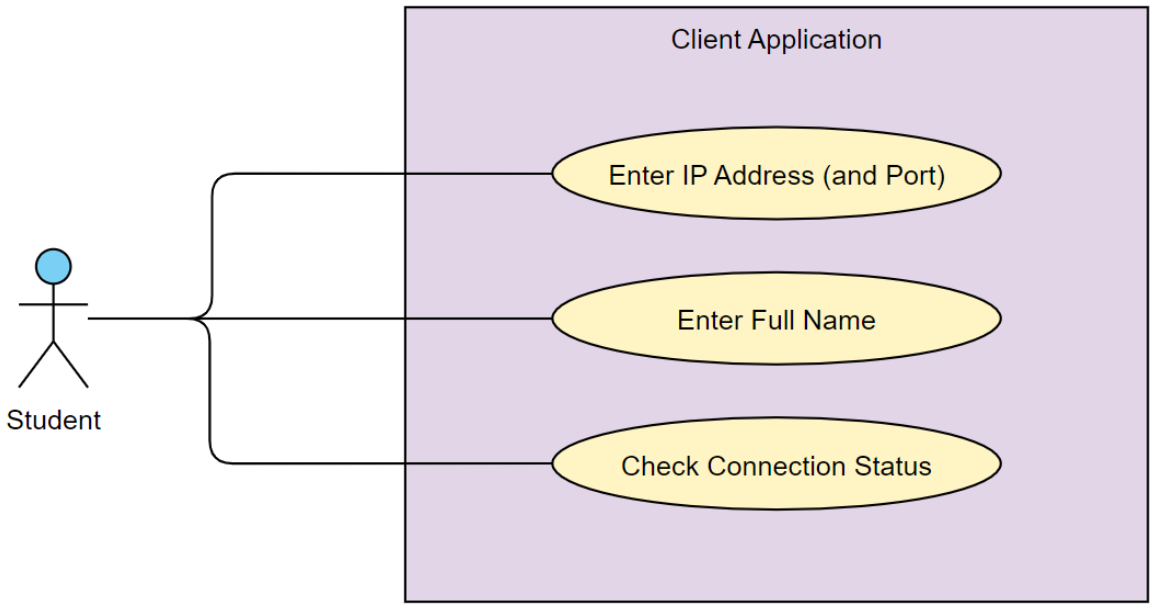

Figure 1 – Use Case Diagram of the Client Application.

The use case diagram in Figure 1 summarizes the user interactions for the client application. The details of the functional requirements are listed below.

1) When the student starts the Client application, a window should prompt to enter the IP address (and optionally, the port) of the server. If the port is not specified, the Client should use the default port of 23000.

To establish the connection between the server and the client, the Client application must know the IP (Internet Protocol) address of the server. An IPv4 (Internet Protocol version 4) address used in the application is a numerical label that consists of four

numbers from 0 to 255, separated by dots. A local or an external IP address can be used for establishing a connection. A local IP address is used when the computers can communicate within the same local network, and a global IP address is used when the computers are connected to the internet. The client and the server must have a direct or an indirect connection involving multiple networks to communicate. A port is a communication endpoint within the computer, which ranges from 0 to 65535. It allows computers to establish multiple connections with other computers while using one IP address. The port must be specified to establish the connection with the server. The default port is chosen to be 23000 as it is not used for any important applications and is typically available. The Windows Sockets 2 (Winsock) is the application programming interface (API) that allows creating a client application in C++ for Windows, and it requires an IP address and a port to establish a connection with a server and to transmit data over computer networks.

2) After the IP address is entered, the student should be prompted with a window to enter the full name.

When a student gets connected to the server, the proctor needs to easily match the displayed data with the person taking an online exam. When the data is personified, it is convenient to display and log (save for future reference). Additionally, the identification of students by name allows the proctor to verify that all students expected to connect have connected. The Client application should not allow the student to enter an empty name and it is the student's responsibility to enter the correct full name.

3) The abovementioned prompts should be stored in a configuration file and automatically typed in the prompts. The student may change them or leave them as they are when prompted.

For the convenience of students, the application should save the previously entered information. If the IP address or port has changed, the student can modify them but if they remained unchanged, the student simply presses the OK button. Since the student using the application is likely to have the same name entered previously, they can press the OK button without entering their name every time. However, if another student wants to use the application, the name can be modified, and it will be automatically updated in the configuration file.

4) A random ID number should be generated upon the start of the application to verify that it is running on the computer it is expected to be running on.

To ensure that the student taking the exam is running the application on the same computer they are using, the proctor may look at the student's shared screen. If the ID numbers do not match, the student must be running the Client application on a different computer and may be suspected of cheating. The number must be random to avoid students modifying the ID number shown in the application. The ID number should contain six digits to prevent collisions (when two or more students have the same ID number) and make it is easy to read and compare.

5) After the setup, a graphical user interface should appear on the screen showing the randomly generated ID number, the name of the student, and the status of the connection (*Initializing*, *Network Error*, *Connecting*, *Connected*).

The application should display the ID number and the name of the student for identification to be checked by the proctor on the screen sharing. The status of the connection should be displayed to the student to identify any problems with the connection and to convey about the successful connection.

6) The statuses *Initializing* and *Network Error* should be highlighted in red color, *Connecting* – in yellow color, and *Connected* – in green color to simplify the identification of the connection status.

Since the student is expected to be connected to the server at all times during an online examination, a color indication should be provided to allow quick and easy detection of any connection problems. In the case of application's or network's malfunction, the proctor should be able to easily verify the problem on the shared screen if the Zoom connection is present.

7) The Client should automatically connect to the server after the setup. If the connection is not established or lost, the Client should automatically retry connecting until it establishes the connection.

The transmission of data from the client to the server, and subsequently, the connection between them, are the main features of the application. Therefore, the Client should attempt to connect to the server as soon as the IP address (and port, if necessary) and the name are entered. Although the students are expected to have a

stable and uninterrupted internet (or local network) connection, there may be connection issues. If the internet connection is lost or has not been established, the application should keep trying to connect to the server until the connection is established or the application is closed.

8) The Client should collect the following data: active window's title, active window's process ID, how many monitors are connected to the computer, whether there is a remote session in process, computer's manufacturer and model, the list of all running processes on the computer.

In order to identify suspicious behavior on the student computer, the Client application needs to collect different kinds of data from the operating system. The active window's title and process ID can be used to find out what application is currently in use. Additionally, many windows have useful information in their titles, such as which website is opened in a browser tab, which document is opened in a word processor, or a presentation program. If the student has multiple monitors and only shares the screen of only one of them, the proctor cannot see what is displayed on the other monitors, so checking the number of monitors can help the proctor identify potential cheating. The computer's manufacturer and the model can be used to identify if the application is running inside of a virtual machine. VirtualBox, VMware, and other virtual machines automatically include their name in the computer's manufacturer and the model when the operating system is started. The presence of a virtual machine can be a sign of cheating since the proctor cannot see what is happening on the host machine. The list of running processes can reveal remote control software and other prohibited applications running on the student's computer.

9) The Client should send the randomly generated ID number, the name, and the abovementioned data to the server in a secure way using TLS encryption.

The data collected from the student's computer along with the ID number and the name of the student should be sent to the server for analysis to be then displayed to the proctor. The application should send the data using Transmission Control Protocol (TCP) to ensure reliable transmission and Transport Layer Security (TLS) to ensure secure transmission. The data should be sent to the server as encrypted text.

2.2. Non-Functional Requirements of the Client Application

1) Sending data to the server should take up only a small amount of network bandwidth.

The data that the Client collects and sends should not be too comprehensive. It should contain only the necessary information for the cheating prevention software's use. Additionally, it should not send the data too often.

2) The messages should be sent to the server four times a second.

The Client application should update the information that the server is showing frequently to detect every switch of a window and every new running program. At the same time, it should not take up too many resources from the central processing unit (CPU), memory, and network. Therefore, a delay of 250 milliseconds between sending data should be introduced.

3) The application should be intuitive and easy to use.

Since the application is intended to be used by students, it should require no special training to operate. The student should need to be provided with the server's IP address only and to expect the application to run without any intervention.

4) The information in GUI should be easily readable during screen sharing.

Since the proctor may need to verify the correct usage of the application using screen sharing, the information displayed by the Client should be easily readable.

5) The Client application should be reliable.

Since the purpose of the application is to be used during an examination, it should cause no distractions for the students and work reliably from the beginning to the end.

2.3. Functional Requirements of the Server Application

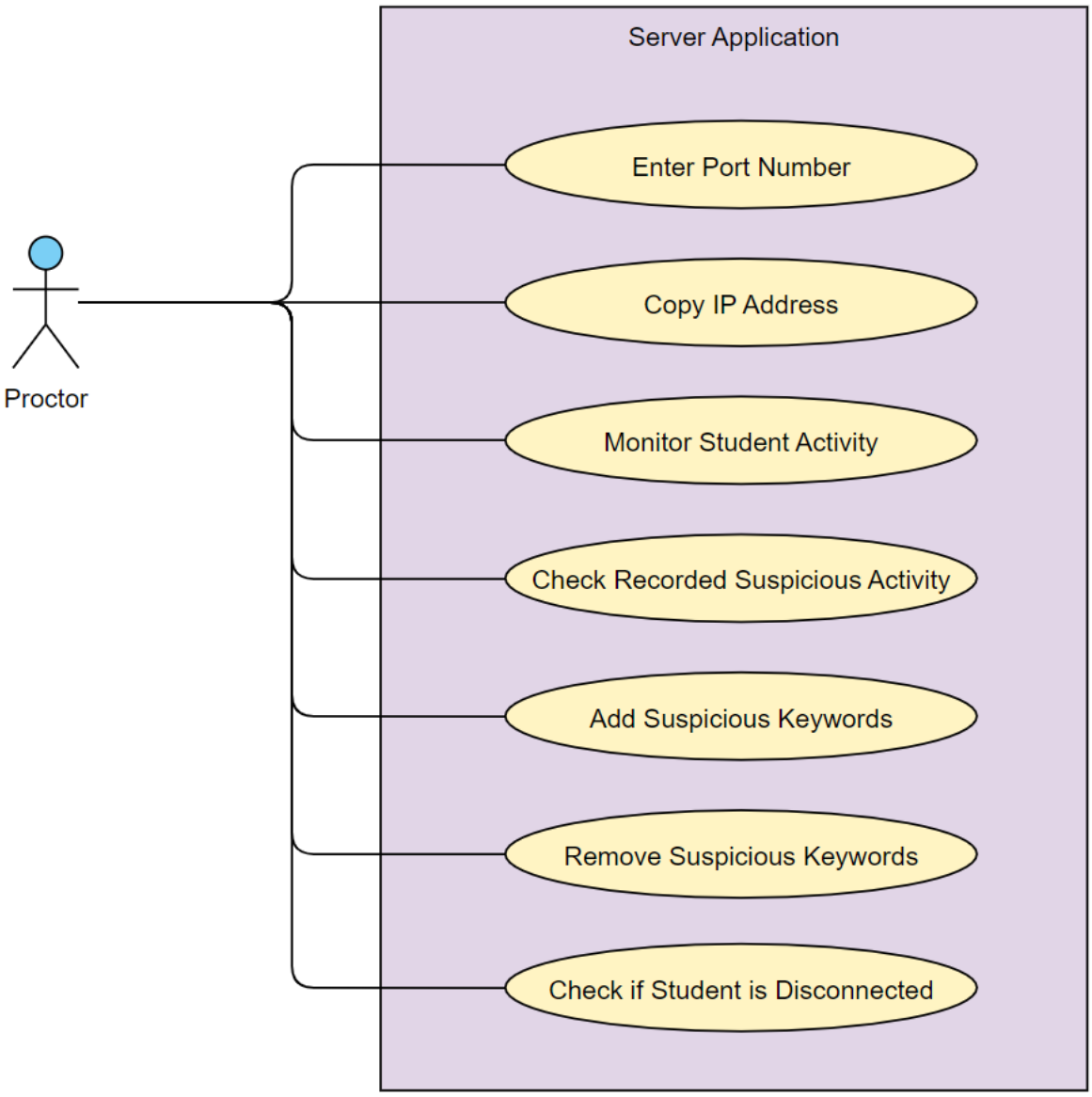

Figure 2 – Use Case Diagram of the Client Application.

The use case diagram in Figure 2 summarizes the user interactions for the server application. The details of the functional requirements are listed below.

1) When the proctor starts the Server application, a window should prompt to enter the server port, and the Server should check if the port is available for use.

In order to establish a listening socket for the server, a port must be specified. This port needs to be unoccupied on the computer, and, therefore, a commonly vacant port of 23000 is suggested as the default port. It can be changed for reasons such as

another program using the port or multiple servers are running simultaneously. Additionally, the selected port must be allowed in the Windows Firewall and open for external connections on the internet router to allow students to connect through the internet network.

2) The application should store the selected port in a configuration file. If the application is started for the first time, the default port (23000) should be prompted, otherwise the last selected port.

For the convenience of the proctor, the last selected port (or the default port) should be saved in a configuration file and suggested in the dialog. It is necessary to allow changing the port for multiple reasons, but the same port is likely to be used every time the application is started.

3) The proctor should be prompted with their public and local IP addresses and the port (unless it is the default port) to be copied and shared with students.

The proctor needs to be able to tell their IP address (and a port, if necessary) to the students who are taking an online exam since the students need to enter the server's IP address to be able to connect to the server. For the convenience of the proctor, the server should display their public and local IP addresses with the port for copying and sharing with students.

4) The graphical user interface of the application should display the students in a grid, dynamically expanding the grid as additional students connect.

After receiving and processing, the information about each student should be displayed to the proctor in the GUI, such that all students were visible at the same time, therefore a grid should be implemented. Since the number of students taking different exams may vary, the window needs to dynamically expand to accommodate all connected students.

5) The application should show if the student is connected or disconnected by highlighting the student's window with green or red color, respectively.

Since the information displayed in the GUI is updated as soon as new information is received, it is necessary to check if the student's information is recent. Therefore, the Server application needs to periodically check the timestamps of the received data. If

the data has not been updated within the last few seconds, the student should be considered disconnected, and their window should be highlighted with red, signifying the connection issues. To reassure that the Server application is working correctly, and the students' received data is being updated, the window should be highlighted in green color.

6) The displayed information should include the random ID of the student, the name, and the student's active window's title and window ID.

To allow the proctor to verify that the student is running the application correctly, the ID number should be displayed to be compared with the one shown on the shared screen of the student. Furthermore, the name of the student should be shown for easy identification of the information shown in the GUI. Since the student's active window's title provides a good understanding of the activity on their computer, the title should be displayed to the proctor. By checking the changes in window ID, the proctor can detect if the student is switching windows. The title is supposed to change when the window ID changes, otherwise, the student is deceiving the cheating prevention software.

7) The application should analyze the title of the student's active window, highlight the title if it is suspicious, and allow the proctor to easily edit the list of suspicious keywords.

To simplify the monitoring of students' activity, the active window's title should be compared with the list of keywords that make it suspicious, and if the similarity is found, the displayed title should be highlighted to focus the attention of the proctor on the suspected student to prevent cheating. Additionally, the proctor should be able to expand the list of suspicious keywords by clicking on them. For example, if the proctor notices a student using a messenger application, he or she should be able to click on the "Messenger" word shown in the window in the application and this word should be added as suspicious and should be highlighted in the future. Similarly, the proctor should be able to remove the words from the suspicious keywords list.

8) The application should warn the proctor if the student has multiple monitors or if the student is using a remotely controlled machine.

Having multiple monitors may be a sign of cheating, since the proctor cannot view what is displayed in them, therefore, a warning should show up to notify the proctor of usage

of multiple monitors. If the application detects that the student is running the Client application on a remotely controlled machine, the proctor should consider this behavior suspicious since the proctor cannot see what is happening on the host machine.

9) The Server should identify if the student is running a virtual machine by examining the computer's manufacturer and model. It should warn the proctor if virtual machine presence is detected.

If the application is running on a virtual machine, the computer's manufacturer and model will include the keywords of the virtual machine's product name. If those keywords are found, the student must be using a virtual machine which should be considered suspicious as the student may be cheating on the host machine which would be invisible to the proctor. A message should be displayed in the GUI if this scenario is detected.

10) The Server should compare the list of the student's running applications with the list of prohibited applications and warn the proctor if any are found.

Numerous applications may be used for cheating on the exams, such as remote desktop control software, screen sharing software, and others. The application should include a list of such applications and allow the proctor to expand the list inside the config file if needed. If the student's list of running applications contains any prohibited applications, the Server application should warn the proctor.

11) The server should keep a log file (recording) of any suspicious activity that happened during an online exam.

The log file should be stored in a folder and contain the timestamps and the relevant information about the suspicious activity. The log files should be opened with a browser since every Windows computer has a browser pre-installed.

2.4. Non-Functional Requirements of the Server Application

1) The application should be informative and intuitive.

The application should be used with little to no training required. It should provide only the necessary information and the students' data should provide enough information to detect any cheating behavior.

2) The Server should be scalable and should accommodate a minimum of thirty students.

Since the online examinations are conducted for a class of students, which normally consists of thirty students, the Server application should be able to display at least thirty simultaneously connected students.

3) The Server application should be reliable.

The application should be designed to withstand loads of data coming in simultaneously. It should process and show the data without interruptions for the duration of an examination.

4) The window of the application should be readable to display all students.

For the convenience of the proctor, the application' window should be adjusted automatically to allow monitoring all students' data simultaneously.

5) The application should have low resource consumption.

The application should work efficiently, without taking up much of the computer's resources, since the proctor may be using other applications, such as Zoom, at the same time.

3. Design of the Software Solution

This section describes the design of the applications, the software architecture, description of the software solution, special features and considerations made in the design of the applications, and the design principles employed throughout the creation of the software solution.

The base architecture for the senior project was chosen to be the client-server architecture where one party initiates the communication, and the other party responds to it [12]. It entails the division of the functions into two parts of the solution, the client, and the server. The architecture implies that multiple clients send requests and receive responses from one central hub – the server [12]. The server provides the ability to store the data received from the clients and send the data to the clients. In other words, the server is the hub of all resources that the clients need. Clients and servers exchange messages in a request-response pattern. The client sends the request, and the server returns the response.

There are several types of client-server architecture [12]:

1) 1-Tier, when user applications make requests to the server, such as database server or a file server
2) 2-Tier, when applications are stored on the server, and the user applications only provide an interface to interact with those applications, such as CRM server
3) 3-Tier, which is a combination of 1-Tier and 2-Tier architectures, when the applications stored on the server interact with another server, and the user applications provide an interface to interact with the server's applications, such as a web server with a database
4) N-Tier, which is similar to the 3-Tier architecture but multiple application servers interact with a data-storing server, and the clients use interface applications to interact with a server application that provides access to other servers' applications.

The advantages of using a client-server architecture are no duplication of the server program code by client programs, scalability, low resource requirements for the

computers running the client since the data processing is done by the server and storing the data on the server which provides better security and permission control. The disadvantages are that if the server fails, the entire system becomes non-functional, sometimes a high cost of the server equipment, and the requirement to have an administrator to support the system.

The basic idea of the client-server architecture is dividing a network application into several applications, where each application provides a specific set of services. Such applications can run on different computers, performing functions of a server or a client. This allows increasing the reliability, security, and performance of network applications.

For the project, the 1-Tier client-server architecture was chosen. In essence, the clients make a connection request to the server, the client and the server perform a TLS handshake to encrypt the connection, and the clients regularly send the data to the server without making any requests. The communications between the client and the server are done over the internet or the local area network.

Both parts of the software solution, the Client and the Server, are written in C++ with the help of the Qt framework to implement graphical user interfaces. For both applications to be interactive and informative, the GUI is necessary. It simplifies the user interaction with applications and makes them more intuitive to use.

Since Qt is a lightweight and extremely capable approach for creating GUI applications for Windows with C++ programming language, it was chosen as part of the design of the applications. In addition to the GUI, the Qt framework offers multiple useful libraries and techniques that differ from the standard C++ programming and allow the creation of complex applications.

3.1. Design of the Client Application

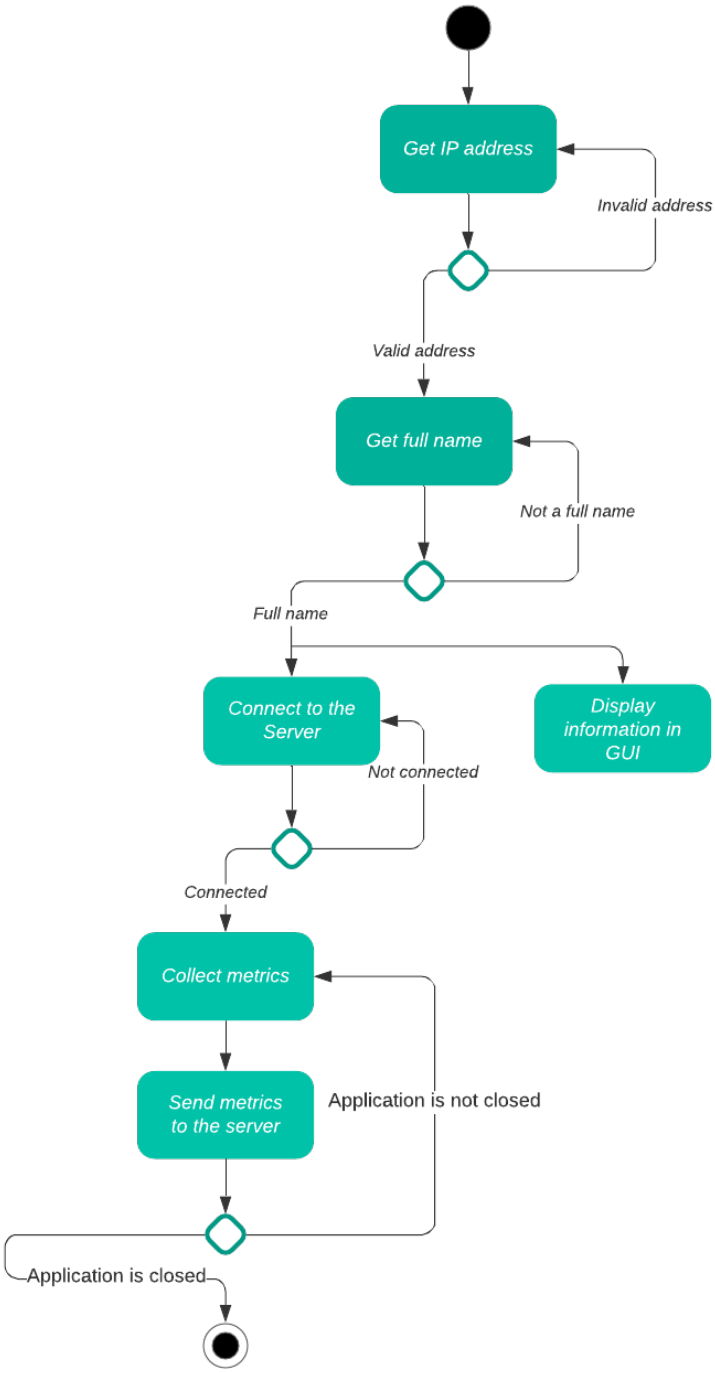

Figure 3 – Activity Diagram of the Client Application.

The activity diagram in Figure 3 represents the workflow of the Client application.

3.1.1. User Interface

The client application's user interface provides two main functions: accepting user input and displaying relevant output.

The application requests the IP address and, optionally, the port of the server and the student's full name from the user. After the user enters the information, it is checked for validity. The IP address should be in the form of A.B.C.D where A must be between 1 and 255, and B, C, D must be between 0 and 255. If the port is specified, it must be separated from the IP address with a colon. The name should consist of at least two words, to make sure a full name is entered. If any of the checks are not satisfied, the application requests the information again.

After the information is correctly entered, the application shows a small window that always stays on top of other windows, is not resizable and is not minimizable. It shows a randomly generated six-digit ID number, the name of the student, and the status of the connection which is highlighted with the appropriate color.

The window serves the purpose of displaying whether or not the application is working correctly and allows the proctor to verify that the student is running the application on the same computer that the test is taken from. By comparing the ID numbers shown on the proctor's server and the student's screen, the proctor can be certain that the student is properly using the application.

The student is not expected to interact with the application after the connection is established to avoid any additional distractions during an exam, therefore the user interface does not include any interactive elements for the user.

3.1.2. Data Collection

The application collects data (metrics) from the computer to be sent to the server. Those metrics include the active window's title and process ID, number of monitors, remote session presence, computer's manufacturer and model, list of running

processes. The active window's title and process ID, number of monitors, and remote session presence are collected using functions in WinAPI. The application gets the computer's manufacturer and model using the entries in the Windows Registry. The list of running processes is collected by creating a snapshot of processes and extracting the names of processes. The collected data is supplemented with the random ID number, the student's name, and the timestamp which is generated by a Qt function that gives the current number of seconds since Epoch (since January 1, 1970, at midnight UTC/GMT). The data is then sent to the server using the networking.

Data collection serves the primary role of the Client application. The abovementioned metrics are chosen specifically to ensure reliable detection of cheating during online exams. The metrics are sufficient for the analysis by the server application.

Due to the amount of data collected on the student's computer, it should be considered sensitive information. Student's name, the title of the window, the computer's manufacturer and model, the list of running processes reveal personal information that can be used for unintended purposes, so the data should be encrypted before it is sent.

3.1.3. Networking

The networking in the client application is implemented using Windows Sockets API. After initiating the Windows Sockets, the application creates a new IPv4 socket and connects to the server with TCP (transport control protocol). The networking incorporates the OpenSSL library that allows establishing a secure connection. After the initial TCP connection is established, the server sends a certificate to the client, and then a TLS handshake occurs. The client uses TLS 1.3 to encrypt the data before sending it to the server. Every 250 milliseconds, the application sends the encrypted data to the server.

3.1.4. Software Architecture

The client application consists of two main classes: mainwindow, and clientnetworking. The class diagram presented in Figure 4 visualizes the classes, attributes, functions, and relationships between classes implemented in the client application.

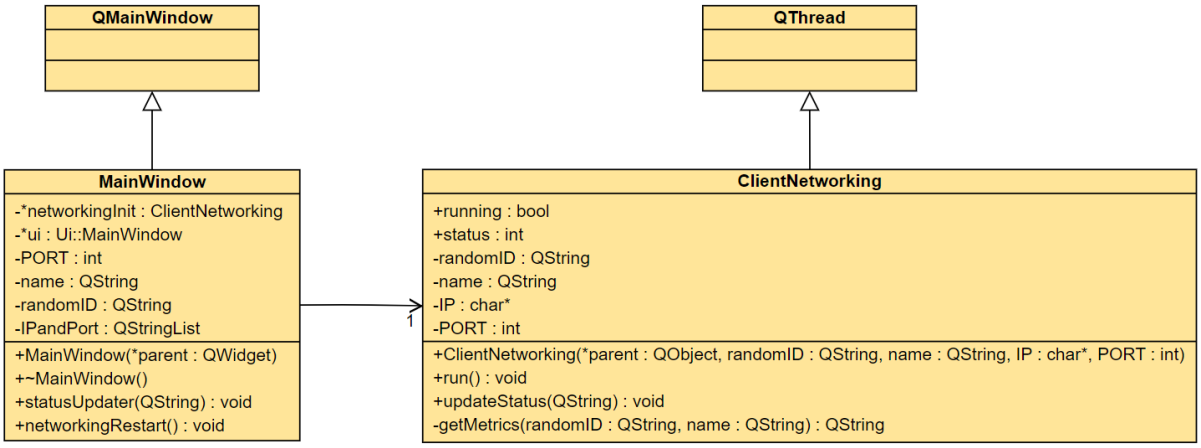

Figure 4 – Class Diagram of the Client Application.

The mainwindow class performs all functions of the applications. It initializes every component of the application, prompts the user with name and IP address windows, creates the networking thread, keeps track of the networking's status using the statusUpdater() function, and restarts it when needed with the networkingRestart() function. Mainwindow class creates a thread for the clientnetworking class and indicates to that class the randomly generated ID, the name, the IP address, and the port.

The clientnetworking class runs in a thread to show GUI and run the data collection and sending at the same time. The clientnetworking class performs data collection through calls to the Windows API. It establishes the connection with the server and exchanges TLS keys for encryption. Afterward, it sends the collected data to the server as encrypted text via the sockets. Clientnetworking class also indicates the status of the connection to the mainwindow class which modifies the respective GUI element. For better code organization, the collection of data is done inside a separate getMetrics function.

3.2. Design of the Server Application

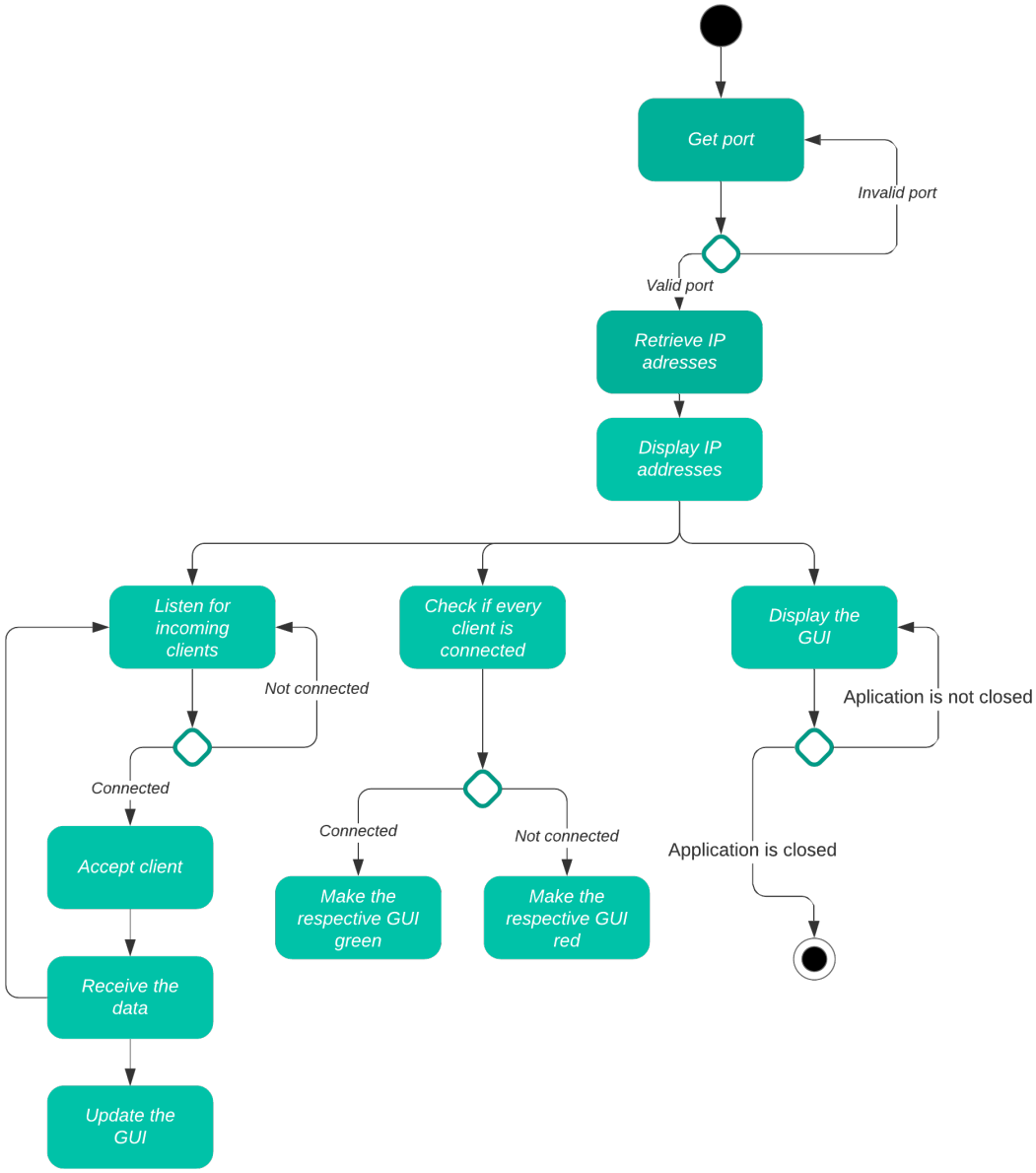

Figure 5 – Activity Diagram of the Server Application.

The Activity Diagram of the Server Application represents how the Server application behaves.

3.2.1. User Interface

When the application is launched, the user interface shows a prompt to select the server's port. Since the port value must an integer between 1 and 65535, the application will show the prompt again if an invalid value is entered.

Afterward, the list of IP addresses is shown. If the port is not the default one (23000), the shown IP addresses will be concatenated with the port value, separated by a colon. The list includes the global IP address of the computer and the IP addresses of all connected network interfaces.

The main window of the server application shows the "Waiting for connections…" sign on startup. After some student is connected, the user interface shows the information received. Each student's information is shown in a box, and all boxes are arranged in a grid. The information always includes the ID number, the name of the student, and the active window's title. If any suspicious activity is detected, a warning is displayed in addition to the shown information. The user interface shows whether a student is connected or disconnected by highlighting the student's displayed box in green or red color, respectively.

The user interface allows the proctor to resize the application for better data monitoring. It offers only one interactive element for the user: the suspicious keywords editing button, done in a form of a hyperlink. The students' window titles shown in the graphical user interface of the application can be clicked. This will create a window with the words that are in the title. Every separate word is presented in the form of a button in that window. If the word is already in the suspicious keyword list, it will be highlighted in red, signifying that activation of such a word's button will remove the word from the list. Otherwise, the activation of the button will add the corresponding word to the list. All buttons with words are situated in a column. The keyword list is updated in the config file after pressing any of the buttons.

3.2.2. Networking

Like in the client application, the networking in the server application is implemented using Windows Sockets API. After initiating the Windows Sockets, the application creates a new IPv4 listening socket and accepts any TCP connections. The networking also uses the OpenSSL library for establishing a secure connection. The application uses a self-generated certificate and a key to encrypt the traffic. When a client is connected to the server, the latter sends a certificate to the client, and then a TLS handshake occurs. The server uses TLS 1.3 to decrypt the data received from the clients.

3.2.3. Data Analysis

The server application analyses the data received from the students to detect any suspicious behavior.

The active window's title is checked for any suspicious keywords (for example, search, messenger, or Quizlet) by comparing the title with the list of suspicious keywords. If any such keywords are found, the application highlights the shown title in red.

If the number of monitors is more than one, the application shows a message showing the number of monitors highlighted in red.

If the remote (RDP) session is detected, the server shows the "REMOTE SESSION DETECTED!" warning.

If the computer's manufacturer and model contain words signifying the use of a virtual machine, such as VMware or VirtualBox, the message "VIRTUAL MACHINE DETECTED!" highlighted in red appears.

If the list of running processes contains any forbidden apps from the corresponding list, the server shows a message "FORBIDDEN APPS DETECTED:", followed by the names of running forbidden applications.

The design of the data analysis is specifically planned to be open for extension. The existing data provides a sufficient amount of information about students' activity to analyze any cheating behavior. If there is a new algorithm to be implemented for analyzing the data, it can be easily added to the data analysis code.

3.2.4. Logging Suspicious Activity

The proctor must be able to check the logs of the suspicious activity after an online exam. It can be used either for checking if there was any such activity that the proctor missed or to verify that the proctor has seen such activity to prove the student's dishonesty.

In order to implement the logging functionality, the design of the server application includes the use of messages that are ready for logging. Since the label widget that displays the students' activity supports HTML tags, the main format for the data was chosen to be HTML. If we save the HTML formatted in an HTML file, we get a ready-to-use log file with no need for external software for displaying the data, since browsers are pre-installed on every computer nowadays. The proctor can open the log file and scroll it to view all the suspicious activity detected during the exam. The log file also includes the timestamps of the activities for informativity.

By checking all instances of the class dataclass, which store a boolean makeLogEntry variable signifying that the data must be stored in a log file, we can periodically save suspicious activity to a log file. During the execution of the getResult() function, which analyzes the students' incoming data, the application changes the makeLogEntry variable to true if any suspicious activity is found.

The log file preserves the same format of the data as the application's GUI, as well as highlights the suspicious activity in red color. These files are saved in the logs folder of the server. The file names are formatted as YYYY.MM.DD_HH.mm.html, with the application's startup year, month, day, hour, and minute, respectively.

3.2.5. Software Architecture

The server application consists of three classes: mainwindow, networking, dataclass. It uses multiple classes for better code structuring and software implementation. The class diagram presented in Figure 5 visualizes the classes, their attributes, functions, and relationships between classes implemented in the server application.

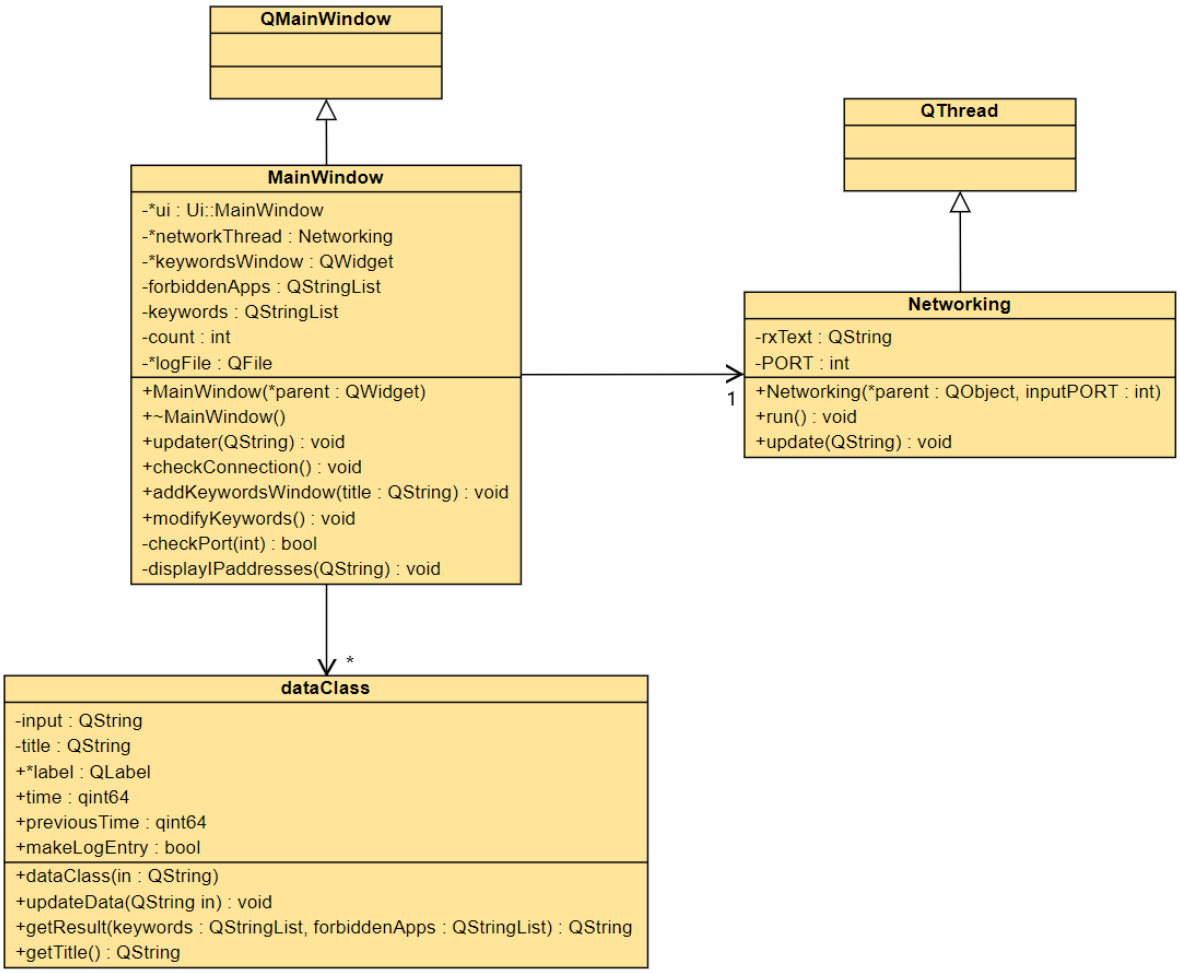

Figure 6 – Class Diagram of the Server Application.

The networking class runs in a separate thread to allow simultaneous execution of the components of the application. The class performs the initialization of Windows sockets and starts listening for connecting clients. It exchanges the TLS certificates with the clients to establish an encrypted connection. The main function of the class is to receive the data from all connected clients and send it to the mainwindow class.

The class dataclass provides a convenient place to store the data received from the clients. It parses the data, analyzes it, and generates commands for displaying information in the GUI when a getResult() function is called. One instance of this class stores information of one student by their ID.

The mainwindow class performs the main processing for the application. It creates and runs the networking thread. It uses the class dataclass to store and process the connected clients' data by storing the instances of the class in a vector. The class checks whether there are certificates required for the TLS encryption, and if they are not found, new certificates are automatically generated using the OpenSSL binary. The mainwindow class performs logging of suspicious activity into a file. It prompts the user for the port. It keeps the forbidden applications and suspicious keywords settings in a config file. The function displayIPaddresses() shows the IP addresses of the computer (both global IP and local IP addresses) in the GUI window. The function updater() updates the information in the GUI when the networking class receives new data. The function checkConnection() keeps track of clients and highlights the ones that are disconnected. The function addKeywordsWindow() allows the user to add and remove suspicious keywords by displaying a new window for choosing the words in the selected sentence. The function checkPort() attempts to connect to the specified port on the computer thus checking if the port is available.

3.3. Security Considerations

The data collected from the student's computer should be considered sensitive and should not be visible to anyone during the transfer. It should not be sent as plaintext, therefore, some encryption algorithm needs to be implemented. One of the most widely used and secure approaches to encrypting network data is Transport Layer Security (TLS) [13]. It is used in web browsing in the form of HTTPS (Hypertext Transfer Protocol Secure), in email transfer, in messaging applications.

TLS encryption provides secure end-to-end communications and ensures that the data that is exchanged will not be readable to anyone on the network. Furthermore, it prevents the data from getting modified during the transfer.

In order to establish a TLS connection, the server needs to have a digital certificate, which can be generated automatically. After a client connects to the server, it requests a secure connection and specifies TLS 1.3 connection. The server responds with the hash function specified for the encryption and sends the public encryption key to the client. The client then validates the certificate and generates the session keys to establish a secure connection.

TLS 1.3 supports the following key exchange algorithms: DHE-RSA, ECDHE-RSA, ECDHE-ECDSA, ECDHE-EdDSA, DHE-PSK, and ECDHE-PSK. All of the algorithms incorporate forward secrecy that ensures that the session key will not be compromised if the private key ever gets compromised. The cipher algorithms supported by TLS 1.3 are AES GCM, AES CCM, and ChaCha20-Poly1305. These algorithms are secure against all publicly known attacks.

In conclusion, Transport Layer Security provides a very secure and fast way of transmitting sensitive data [13]. It is easy to implement using the OpenSSL library and adds very little load on modern computers. However, in order to use TLS encryption, the server must have TLS certificates or keys.

In end-to-end encryption using Transport Layer Security, the encryption and decryption of data are performed using keys [13]. There are three keys involved: a public key that encrypts messages, a private key that decrypts messages, and a session key that can do both encryption and decryption but can only be used for one session.

TLS certificates usually contain:

- The name of the issuing Certificate Authority
- Issue and expiry date
- The domain name
- The organization
- Additional subject domain names, including subdomains
- The public key
- The digital signature by the Certificate Authority

When the client connects the server, it receives a TLS certificate with the public key. It verifies that the certificate is valid and if so, it sends a symmetric session key to the server, which decrypts that key using the private key. The client uses the public key to encrypt the outgoing messages and the session key to decrypt the incoming messages.

The server application uses OpenSSL to generate self-signed SSL/TLS certificates for the secure connection. It generates them during the first launch and creates new ones at the beginning of the next year to keep the certificates secure. The certificates are stored in the same folder as the application, and the file names are certXXXX.pem for the certificate containing the public key and keyXXXX.pem for the private key, where XXXX is the current year, i.e., cert2021.pem and key2021.pem. The parameters for the certificate generation are:

- 365 days of validity, starting from the generation day
- 4096 bit RSA key
- Common Name – CheatingPreventionSoftware
- Organization – AUBG
- Locality – Blagoevgrad
- State – Blagoevgrad
- Country Name – BG.

3.4. Communication Protocol

The client and the server applications use a custom socket-based communication protocol designed specifically for this senior project. It is connection-oriented and ensures reliable transmission of data between the server and the client by establishing a TCP connection over IPv4. The use of TLS encryption is omitted since it was discussed in the previous section.

A network socket is a software interface that serves as an endpoint for receiving and sending data in a computer network. Sockets use a TCP/IP data transmission model to allow communication between devices with specified IP addresses and ports. There

are two sockets involved during communication, a client socket, and a server socket (listening socket). There are reliable stream sockets (TCP, connection-oriented) when the delivery of the data is ensured and datagram sockets (UDP, connectionless) when the delivery and reliability are not guaranteed, and raw sockets when there is no specific transport layer formatting [10]. Sockets can communicate over IPv4 or IPv6 network protocols.

The protocol's design objectives are to be simple, efficient, and to minimize the traffic load. The protocol is chosen to be text-oriented, where the messages are readable character strings. The message structure is pre-defined and sends the data in a particular order. Thanks to the use of TCP, the protocol provides reliable delivery of messages and ensures that the data is not damaged, modified, or lost [10]. In the case of connection loss, the protocol provides a detection mechanism that can be implemented both by the client and the server.

The protocol is designed to perform the following sequence on the server:

1) Create a listening socket that works over IPv4
2) Bind the socket to the sockaddr_in structure configured to accept connections from any network adapters
3) Listen for incoming client connections with the limit of clients set to maximum
4) Initialize an fd_set structure to allow multiple clients
5) Iterate through all fd_set elements, allowing some clients to connect and the other clients to communicate with the server at the same time
6) Accept all incoming connection requests and add new client's socket to the fd_set
7) Receive the incoming data from a connected client and move on to the next client

The protocol implementation for the client is as follows:

1) Create a client socket working with IPv4
2) Bind the socket to the sockaddr_in structure configured with the given IP address and port
3) Connect to the server
4) Regularly send 8192 bytes of data to the server

The data that is sent using this protocol has a specific structure. The data is presented as text with a fixed length of 8192 characters, so the size of the data is 8192 bytes. There are nine entries in the data's structure, separated by a semicolon (;). The data entries are as follows:

1) Student ID represented as six digits
2) Student's full name, that is no longer than 50 characters, represented as at least two words
3) The active window's title, which is no longer than 128 characters
4) The active window's ID number represented as an integer
5) The number of monitors represented as an integer
6) The computer's manufacturer and model, each no longer than 255 characters long
7) The flag that a remote session is active, in the form of a digit 0 or 1
8) The timestamp in the form of an integer number of seconds since Epoch
9) The list of all running applications in the form of names of executables, separated by a comma

If the resulting message is longer than 8192 characters, part of the running processes list will be truncated to fit the message for the protocol. However, there has not been a case that the message would not fit during the development, so the chance that some processes will be sacrificed is very unlikely.

3.5. Design Features Imposed by the Qt Framework

Qt framework is highly multithreaded since it is a GUI-oriented framework. Different components of a Qt program are running simultaneously and exchanging the data with each other. It has mechanisms for detecting user interaction with the GUI and responding to those interactions in the code. Observable objects such as buttons, sliders, text fields can respond to interactions by creating windows, changing variables, and so on.

The components of the application can transmit information to each other by "emitting" a "signal" [6]. The signal is declared in the header file for the class, and the signal can

be called by a function emit() within the code. The recipient of the signal must have a “slot” declared in its header, which also must be implemented in the recipient code. The slot function is launched when the corresponding signal is emitted and the function receives the parameters from the signal. Multiple signals can be connected to one slot and vice versa. This mechanism allows the programmer to respond to any event within different threads or classes in the code.

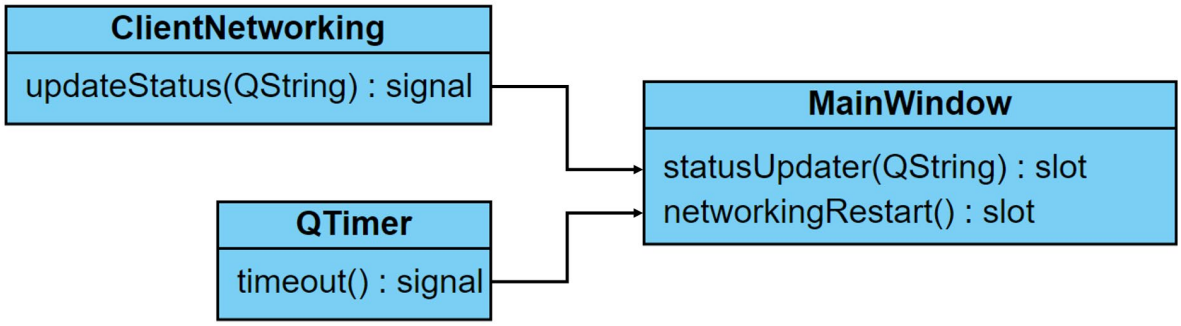

Figure 7 – Signal-Slot Relationship of the Classes in the Client Application.

The signal-slot relationship diagram in Figure 7 visualizes the connections between class objects. As you can see, the client application uses this mechanism to update the status of the network connection in the GUI window from the ClientNetworking class by emitting an updateStatus(QString) signal which invokes a statusUpdater(QString) slot. Also, timeout() signal invokes a networkingRestart() slot.

The server application uses the mechanism more extensively, as you can see in Figure 8. The networking class emits the update() signal to send the incoming data from the clients to the mainwindow class that is responsible for data processing and output to the GUI. Additionally, a timer can be used to track any changes or perform actions periodically. The server application utilizes a timer to check if the clients are connected by comparing the timestamps of the clients’ last updates. If the timestamp has not changed within some period of time, then the client is disconnected. In order to create the functionality of addition and removal of suspicious keywords, buttons were used. The clicked() signal of a button calls a modifyKeywords() function that adds or removes a keyword respective to the button. To open a window with the abovementioned buttons, the user clicks a text link in the GUI, which triggers a linkActivated() signal that causes the new window to pop up.

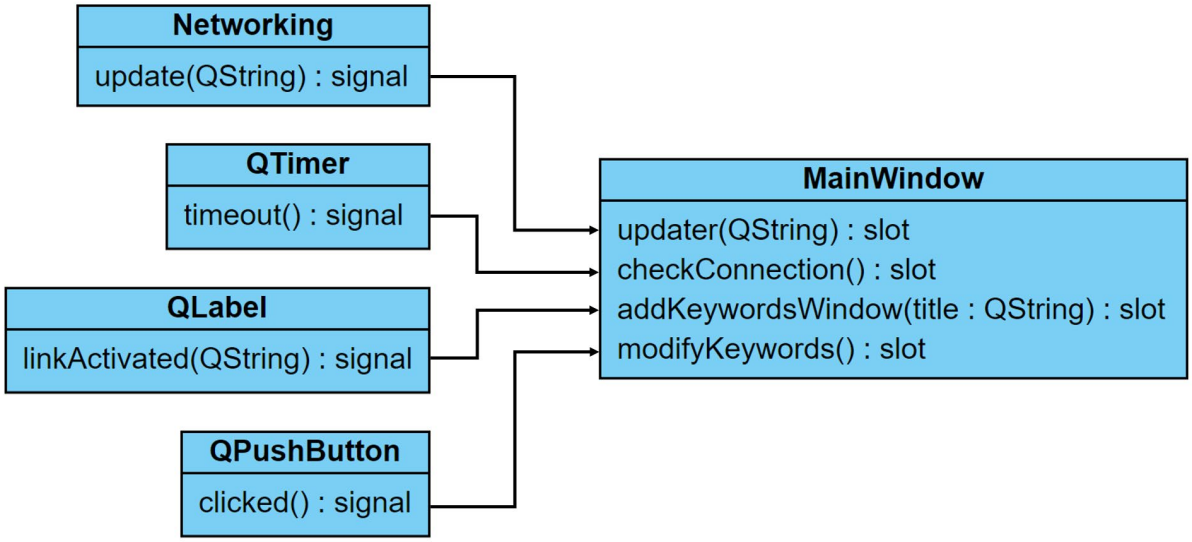

Figure 8 – Signal-Slot Relationship of the Classes in the Server Application.

Another notable component of the Qt is widgets, which provide the programmer with various GUI elements. These include buttons, spacers, item views, containers, input and output widgets [6]. The framework allows adding these widgets in advance, during the development, as well as on the fly, in code. This allows creating an application with a static GUI or a dynamically changing GUI. The client application has a static GUI, meaning that the elements are created when the program is launched and their properties, such as the text of a label, are modified during the program's execution. The server application's GUI, on the other hand, is largely dynamic. When the application is launched, it shows the IP addresses of the computer in a text box inside a standalone window. The main window can add new elements as more clients connect, allowing for an arbitrary number of simultaneous clients. Additionally, the server application creates an additional window on the fly, if the user needs to add or remove the keywords. The Qt widgets allow the programs to have an intuitive and useful graphical user interface with very easy code.

3.6. Reliability Considerations

The goal of the applications is to detect and prevent cheating during online exams, so reliable execution of the applications is vital. They need to work without any crashes, handle multiple connected students, and ensure a reliable connection. The client application must remain open throughout the examination. If the student's internet connection gets interrupted, the client application must inform the student in the GUI and try to reconnect until the connection is established. Even if the network routines fail, the application will restart the network subsystem, since the random ID is meant to stay the same throughout the exam and the student should not be distracted from the exam.

The server application is designed with similar principles. The proctor should be able to reliably monitor the students in the server application, therefore, the server should work without any freezes or crashes. It must accommodate any reasonable number of connected clients and process the incoming information without any delays. The design of the algorithms and data processing is based on the principle of efficiency and low resource consumption. The application is optimized to work on any low-performance computer thanks to the use of the C++ programming language, efficient libraries, and fast Qt framework's mechanisms.

Additionally, the programs verify that the student is using the application on their respective computer by displaying the random ID. If the student launches the client application on a different computer, the proctor will notice that on the student's shared screen. If the student restarts the program, the proctor will see that a new student is connected since the ID will be new. Reliable communication between the client and the server ensures that the student is not cheating the cheating prevention system.

3.7. Cross-Platform Portability

The students use different computers when taking online exams, and the operating systems they use also vary. The project is designed such that the applications will work on any computer running the Windows Operating System to maximize the usability and

the reach of users of the applications. Since the project is meant to be used during any online exam, the software design aims to allow every student to run the client application on their computer.

At the time of the creation of this senior project, the client and the server applications are designed for the Windows platform, since all computers on the AUBG campus (except in Mac Lab) are running Windows, and the majority of the professors use Windows as well. Starting with the compiling standpoint to the choice of libraries, the applications are not cross-platform and cannot be recompiled for a different operating system. The core libraries are tied to the Windows networking stack, with the calls and routines that can only be compiled for a computer running Windows Operating System.

The applications are written using the Qt Framework, one of the main features of which is cross-platform development [6]. This means that the code written for Windows that does not involve Windows-specific libraries can be easily ported to another platform. If we replace those libraries with the libraries that work on another platform and replace the calls to the functions, the code can be ported without much trouble, keeping the majority of the code unchanged. Since the Qt Framework provides a cross-platform GUI implementation, the graphical user interface of applications will look and work the same across different platforms, without any modifications in the code.

In the server application, the Windows-specific libraries used are the Windows Sockets 2 (Winsock) API which is provided by the ws2tcpip.h header file and the Windows Internet (WinINet) API which is provided by the wininet.h header file. They can be replaced by, for example, the Linux sockets from sys/socket.h header and a libcurl API. Another Windows-specific call is the server's command to generate new certificates for TLS encryption, where the application runs the OpenSSL executable with the specific arguments. The OpenSSL Windows executable can be replaced with the one for another platform. Since OpenSSL is cross-platform, the code modifications will only include a replacement of the function. The server application can be ported to another platform if needed while leaving most of the code unchanged.

The client application, on the other hand, involves multiple Windows API calls in addition to the Windows Sockets 2 since it collects the metrics from the operating

system. This means that porting the client application to another platform will require most of the source code to be modified in order to implement data collection on a different platform. However, the GUI and user interactions can be preserved since they are written using the Qt framework.

3.8. Employed Design Principles

Throughout the client and the server applications' code, there are several fundamental software design principles employed. These principles ensure the readability and simplicity of the code and allow the applications to be extended and modified in the future.

One of the main principles of object-oriented programming is the SOLID. It includes the following concepts [3]:

1) The Single-responsibility principle, which states that any class should have only one responsibility
2) The Open–closed principle, which states that any software should be open for extension but closed for modification
3) The Liskov substitution, which states that functions using pointers or references to base classes must be able to use objects of derived classes without knowing it
4) The Interface segregation principle, which states that multiple client-specific interfaces are better than one general-purpose interface
5) The Dependency inversion principle, which requires to depend upon abstractions not concretions

In addition to the principles above, the following principles are employed throughout this senior project: "don't repeat yourself" or DRY principle that reduces the repetition of software patterns, the principle of least astonishment that ensures that a solution or approach would not surprise a reasonably knowledgeable person in the subject area when encountered for the first time, the separation of concerns principle that requires separating each distinct section that addresses different individual concerns [3].

4. Implementation

This section dives into the implementation of the software solution, the technologies and approaches used to implement the applications, and the detailed descriptions of the choices made throughout the making of the applications' code.

The client and the server applications were written using the C++ programming language, the Qt framework, and several libraries. The Qt Creator integrated development environment was chosen for development since it offers multiple Qt-related features for GUI development.

4.1. C++ Programming Language

From the beginning of this project, it was clear that the applications should be written in C++. It is fast and the code will run on any modern computer with little to no impact on the performance. It allows creating complex applications and using a variety of libraries and APIs. Built-in low-level calls allow the programmer to interact with the operating system's parameters and with the hardware.

C++ is a compiled statically typed general-purpose programming language [1]. Statically typed means that variable types are set at compile-time and cannot be changed. Modern C++ has object-oriented, generic, and functional features [1]. It provides security features and simplifies the code by function and operator overloading.

The main reason for choosing C++ was the ability to get full access to the operating system's APIs. Since the client's main objective is to collect metrics from the Windows OS, it is necessary to interact with Windows without any intermediate layers. The User32 and Kernel32 libraries provide the Windows API for C++ [4]. The Win32 Application Programming Interface is a set of functions running under the Windows operating system, contained in the windows.h library. In addition, C++ allows creating the custom network protocol using sockets provided by the Winsock API provided by the ws2tcpip.h C++ header file [14].

4.2. Qt Framework

To create graphical user interfaces for C++ applications, we need special tools. Qt Framework provides such tools for multiple platforms, including Windows. Qt offers developers a modular library of over 700 C++ classes, Qt Quick technology for creating UI using the declarative QML language, and a professional toolkit [6]. Qt allows developers to speed up the process of software development, to increase code's efficiency, considerably reducing the development duration.

Qt allows one code to run on most modern operating systems by simply recompiling it or each system without making any changes. It includes multiple basic classes that may be needed when developing applications, from GUI elements to networking, database, and XML classes. It is fully object-oriented, extensible, and supports component-oriented programming techniques.

Qt includes a visual GUI development environment called Qt Designer that allows creating dialogs and forms in WYSIWYG (What You See Is What You Get) mode [6]. Qt comes with a Qt Linguist tool to simplify localization and translation of the program into other languages, and a Qt Assistant that provides a tool that simplifies creating documentation for the libraries and makes it possible to create cross-platform documentation for Qt-based applications. The Qt Creator includes a development environment, a code editor, assistance tools, the Qt Designer graphical tools, and debug tools. Qt Creator can use GCC or Microsoft VC++ as a compiler and GDB as a debugger. Windows versions of Qt Creator include a compiler and MinGW files.

Qt Widgets is a C++ library for creating user interfaces with a native look for each desktop platform. It is commonly used to develop large-scale interfaces for desktop platforms. Qt Widgets are easily customizable, extensible, and can be adapted to accommodate any interface.

Qt Framework offers the following benefits:

- Use of a single framework for development

- Use of C++ and QML
- High performance due to the native platform
- Quick start of development: convenient installers, ready-made tools, the ability to create prototypes
- Qt Creator development environment with the ability to run and debug directly on the device with additional tools
- Detailed API documentation, examples, and tutorials
- A single platform for creating any application

4.3. Qt Modules and Libraries

Qt is more than just GUI elements. The Qt Framework is an interconnected system, whose objects are related through inheritance of the QObject class. Qt Framework includes several modules that provide different functionality for the applications. The following modules are used in the client and the server applications:

1) Qt Core which includes core non-graphical classes used by other modules
2) Qt GUI which includes base classes for GUI components
3) Qt Widgets module which extends Qt GUI with C++ widgets

The abovementioned modules provide all the Qt-related functions and libraries used in the applications. They are provided as a part of the Qt Framework.

The Qt libraries used by the client application are the following [5]:

1) QtCore, which adds multiple Qt-specific features to C++. It provides object communication through signals and slots, queryable and designable object properties, guarded pointers, a dynamic cast across library boundaries
2) QMainWindow, which provides the main application window
3) QSettings, which provides persistent application settings
4) QInputDialog, which provides a simple dialog to get a single value from the user
5) QThread, which provides threads management
6) QTimer, which is a class that provides repetitive and single-shot timers

The server application uses the same libraries as the client but additionally uses the following [5]:

1) QLabel, which is a widget that provides a text or image display
2) QGridLayout, which is a class that allows arranging widgets in a grid
3) QVBoxLayout, which is a class that allows arranging widgets vertically
4) QVector, which is a template class that provides a dynamic array
5) QMessageBox, which is a class that provides a dialog for informing the user or for asking the user a question and receiving an answer
6) QPushButton, which is a widget that provides a command button
7) QFile, which is a class that provides an interface for reading from and writing to files
8) QProcess, which is a class that is used to start external programs and to communicate with them

The Qt libraries used in the applications are necessary for the successful execution of the fulfillment of the requirements. The client application uses QThread class to run the client's clientnetworking class or the server's networking class in a separate thread, allowing multithreading and simultaneously operating the GUI window and communicating with the server. QSettings class is used to store configuration variables in a file and restore them during the next run of an application. QTimer class is used to regularly invoke functions in the application code. QFile class is used to check the existence of files, such as certificates for TLS encryption. QInputDialog class is used to display any errors to the user during the application's execution. QVector class is used to dynamically store instances of the class dataClass in the server application. QProcess class allows the server application to generate self-signed TLS certificates by running the OpenSSL executable and passing the appropriate parameters.

All Qt objects can have signals and slots, which allow the objects to interact with each other. A slot is called when it receives a corresponding signal. A signal is emitted by an object that wants to invoke a function that corresponds to a slot. The parameters passed to an emitted signal can be further passed to the slot's function. For example, a QTimer object emits a timeout() signal when the specified time has passed, then we can connect that signal to a networkRestart() slot in the client application, resulting in invoking the function networkRestart() every specified time.

4.4. Windows Sockets 2

Windows Sockets 2, or Winsock, is an application programming interface (API) that provides access to the network functions and network services, such as TCP/IP, on Windows. Winsock is the interface between the application and the transport protocol that handles the data transfer. Winsock is used on the client to connect to the server and send the data, and on the server to accept incoming clients and receive the data.

In C++, Winsock can be included in the code as a ws2tcpip.h header file, which includes winsock2.h header, and implements Winsock API's functions. Additionally, the wsock32 and Ws2_32 libraries must be linked for implementing the API calls. Winsock is pre-installed on Windows Operating System and can be accessed by applications through the wsock32.dll Dynamic-link library.

The following steps are taken by the client to establish a connection [7]:

1) Initialize Winsock by a WSAStartup() function, specifying version 2.2 of the API
2) Create a connection socket with IPv4 protocol and socket type TCP
3) Setup the connection type and destination by specifying the address type IPv4, the server's address, and port in a sockaddr_in variable
4) Connect to the server by a connect() function
5) Send the data using a send() function (due to TLS, use SSL_write instead)
6) Close the connection using closesocket() function and perform a cleanup with WSACleanup() function

The server takes the following approach [7]:

1) Initialize Winsock by a WSAStartup() function, specifying version 2.2 of the API
2) Create a listening socket with specified protocol IPv4 and socket type TCP
3) Setup the connection type by specifying the address type IPv4, the interfaces which will accept connections to be any, and the port in a sockaddr_in variable
4) Listen for connecting clients with a listening socket
5) Initialize an fd_set structure that stores listening socket and connected clients' sockets to allow multiple clients to connect simultaneously

6) Iterate through all fd_set elements, if the current element is a listening socket, then accept a new client and add to the set, otherwise receive the message from a client using recv() function (due to TLS, use SSL_read instead)

Windows Sockets 2 is the only API that allows interacting with networking on Windows. Microsoft provides detailed documentation that includes examples of client and server implementation in C++. In addition, Winsock provides the programmers with the opportunity to create their unique communication protocol using sockets, which allowed creating one for this project.

Furthermore, the Winsock API allows the server to list the local IP addresses of the computer's network interfaces. After initializing the API with WSAStartup() function, we read the hostname of the computer using gethostname(), then we write the result of the gethostbyname() function into a hostent structure. We read the h_addr_list parameter from each element in the structure into an in_addr structure and translate it into an IP address using the inet_ntoa() function. As a result, we get the local IP addresses of the computer.

Another use of the Winsock API on the server is checking if the chosen port is available using the checkPort() function. It initializes a client socket and attempts to connect to the specified port on the localhost (127.0.0.1), and returns true if the connection is successful, signifying that the port is not available.

4.5. OpenSSL

OpenSSL is an open-source software library that allows applications to establish secure communications over computer networks. It supports almost every hashing, encryption, and electronic signature low-level algorithms, and implements the most popular cryptographic standards, including RSA, DH, DSA keys, X.509 certificates, generating them, testing, and data encryption over SSL/TLS connections [2]. The main reason that the OpenSSL library was chosen for this project is its ability to encrypt and decrypt messages ensuring the security of the communication using TLS protocol and the function of generating self-signed certificates for the TLS communication [2].

To use OpenSSL in a C++ application, we need to include openssl/ssl.h and openssl/err.h header files as well as libssl and libcrypto libraries in the project.

The secure communication protocol chosen for the project is TLSv1.3 (the latest version of the successor of SSL), and the cipher is TLS_AES_256_GCM_SHA384. The data is encrypted on the client, then sent to the server, where it is later decrypted.

The automatic generation of self-signed certificates is performed by running the ".\openssl.exe req -new -newkey rsa:4096 -days 365 -nodes -x509 -subj "/C=BG/ST=Blagoevgrad/L=Blagoevgrad/O=AUBG/CN=CheatingPreventionSoftware " -keyout keyXXXX.pem -out certXXXX.pem" command which outputs a public key certificate certXXXX.pem and a private key keyXXXX.pem, where XXXX is the current year. The certificate uses a 4096-bit key and expires in 365 days since generation.

The implementation of OpenSSL on the server is as follows [8]:

1) Initialize the OpenSSL routines by performing SSL_load_error_strings(), SSL_library_init(), and OpenSSL_add_all_algorithms()
2) Create SSL_CTX object with SSL_CTX_new() specifying TLS protocol and require it to be version 1.3 using SSL_CTX_set_min_proto_version()
3) Load the certificate using SSL_CTX_use_certificate_file() and the private key using SSL_CTX_use_PrivateKey_file()
4) Create an SSL object using SSL_new(), and assign it to the client's socket using SSL_set_fd()
5) Perform a TLS handshake using SSL_accept()
6) Receive and decrypt messages using SSL_read()

The client application performs a similar procedure [9]:

1) Initialize the OpenSSL routines by performing SSL_load_error_strings(), SSL_library_init(), and OpenSSL_add_all_algorithms()
2) Create SSL_CTX object with SSL_CTX_new() specifying TLS protocol and require it to be version 1.3 using SSL_CTX_set_min_proto_version()
3) Create an SSL object using SSL_new(), and assign it to the client socket using SSL_set_fd()

4) Initiate a TLS handshake using SSL_connect()
5) Encrypt and send the data using SSL_write()

A TLS handshake is the process of initiating and establishing secure communication using TLS encryption. It is performed as follows [11]:

1) First, the and the server establish a TCP connection
2) The client sends a “client hello” message specifying the supported TLS versions, the supported cipher suites, and a random string
3) The server responds with a “server hello” message specifying the chosen TLS version, the certificate with a public key, and a random string
4) The client verifies the server’s certificate and sends an encrypted “premaster secret” to the server
5) The server decrypts that secret using the private key
6) Both parties generate the session keys using previously exchanged messages
7) The client sends a “finished” message encrypted with the session key
8) The server sends a “finished” message encrypted with the session key

After the handshake is complete, the client and the server can communicate securely using the session key for encryption and decryption.

4.6. Other Libraries

In addition to the abovementioned libraries, there are several other libraries used in the applications due to their specific feature set.

To generate a random ID number, the client application uses the standard C++ library’s mt19937 object that uses time as a seed for generating random numbers. An integer number between 0 and 999999 is generated with a uniform_int_distribution<int> object that requires passing the mt19937 object to it.

The client application uses a tlhelp32.h header file and a Kernel32 library which provide WIN32 tool help functions, types, and definitions. They are used to get the list of running processes by creating a snapshot with CreateToolhelp32Snapshot() and reading the processes from it with the Process32Next() function [4].

The other metrics are collected by the client application using GetWindowText() to read the title of the window, GetWindowThreadProcessId() to get its process ID, RegGetValue() to get the computer's manufacturer and model from the Windows Registry's SYSTEM\CurrentControlSet\Control\SystemInformation path using SystemManufacturer and SystemProductName entries respectively, GetSystemMetrics(SM_CMONITORS) to get the number of monitors and GetSystemMetrics(SM_REMOTESESSION) to check if the remote session is running. These functions are provided by the User32 Windows library.

The server application uses a wininet.h header file and a wininet library which provide the Windows Internet API for C++. WinINet allows the server to access standard Internet protocols such as HTTP, and it is used to request the global IP address of the computer [14]. To do that, the server uses InternetOpen() function to initialize the Windows Internet API, then performs InternetOpenUrlA() that opens a website, which returns the global IP address as a string that we can read using InternetReadFile() function and perform InternetCloseHandle() afterward to close the API.

4.7. Compilation and Deployment

Since the project is created in the Qt Creator IDE, which provides built-in mechanisms for compiling and deploying applications, the executables for the client and the server are built in a release mode using the IDE. Qt Creator uses project files that have a .pro format for the configurations of the application. These files contain information for the qmake that builds the application and specifies the resources in the project. The client application is configured as follows:

1) Qt libraries used are core, gui, and widgets
2) The C++ programming language version C++11
3) The source files are clientnetworking.cpp, main.cpp, and mainwindow.cpp
4) The header files are clientnetworking.h and mainwindow.h
5) The form (user interface markup) file is mainwindow.ui
6) Libraries used are wsock32, Ws2_32, User32, Kernel32, and from the lib folder of the OpenSSL library: libssl and libcrypto

7) Include the headers from the include folder of the OpenSSL library
8) Define the Unicode standard for encoding text in the application
9) Set the icon of the application to be icon.ico from the project's folder
10) Define the default rules for deployment

The server application is configured slightly differently:

1) Qt libraries used are core, gui, and widgets
2) The C++ programming language version C++11
3) The source files are dataclass.cpp, main.cpp, mainwindow.cpp, and networking.cpp
4) The header files are dataclass.h, mainwindow.h, and networking.h
5) The form (user interface markup) file is mainwindow.ui
6) Libraries used are wsock32, Ws2_32, wininet, and from the lib folder of the OpenSSL library: libssl and libcrypto
7) Include the headers from the include folder of the OpenSSL library
8) Set the icon of the application to be icon.ico from the project's folder
9) Define the default rules for deployment

Using the abovementioned project files, the Qt Creator automatically executes mingw32-make.exe when the project is run or built. The produced executable file can be launched inside the IDE but it lacks the necessary libraries to run standalone, so we need to perform the deployment.

The executable files should then be deployed using the built-in Qt 6.1.3 (MinGW 8.1.0 64-bit) tool that provides a windeployqt.exe application. The Qt Framework provides two ways of deploying applications: static linking and shared libraries, and the latter was chosen for this project to provide flexibility. To perform the deployment, we need to open the location of the compiled executable of a Qt application and run the "windeployqt.exe --quick --no-translations ." command. This tool will add all necessary Qt files that the application requires to run.

In addition, the server application must have the OpenSSL executable openssl.exe in the same folder to allow generating certificates. It is free and open-source, and the OpenSSL version for Windows can be downloaded from

https://slproweb.com/products/Win32OpenSSL.html or the OpenSSL source files from https://www.openssl.org/source/ that can be compiled for Windows. The executable is provided with the server files in the Server folder for the project. The OpenSSL libraries libcrypto-3-x64.dll and libssl-3-x64.dll need to be added to the applications' folders. Additionally, the openssl.cfg configuration file needs to be provided in the same folder. These files are included with the applications for the project.

4.8. Installation and Hardware Requirements

There is no installation needed to run the client application. The user should open the respective folder with the application and run the Client.exe file. The application will start without any additional steps from the user.

The server requires some files from the OpenSSL to be provided with the application since there are dependencies that the openssl.exe file requires for self-signed key generation. In this project, all necessary files are provided for the application, so it can be run without any additional steps from the user.

In addition, the applications need to run on a computer without a software network and execution locks such as the ones implemented on AUBG computers that are imposed by policies. The networking cannot properly initialize on such computers due to the restrictions.

The minimum hardware requirements for the applications are the following:

- Windows 7 or newer Windows OS
- 1 GB of RAM
- 1 GHz CPU
- OpenGL ES 2.0 support
- 100 MB of free space
- Stable network connection with the speed of 4 Mbps

5. Testing

The applications have been thoroughly tested throughout the development process to verify that all requirements, both functional and non-functional, have been met. This section explains the testing process and any findings discovered during testing.

5.1. Client Application Testing

The testing of the client application starts with the user prompts and various values passed to it. When the application is launched, the user is prompted to enter the IP address in the form of A.B.C.D or A.B.C.D:PORT. If the user enters an incorrect value, the prompt will show the prompt again.

Several tests were performed to test the checks, which have presented the prompt again (wrong IP address):

- Check
- localhost:25565
- 0.0.0.0
- 500.500.500.500:25000
- 1.2.3
- check.check.check.check
- An empty value

The examples of a suitable IP address are 127.0.0.1 and 127.0.0.1:23000

The next prompt asks the user to enter a full name in the form of First Name Last Name. If the user enters only the first or only the last name, they will be prompted again. Testing of these verifications reveals that “Vitaliy” and “Konyukhov” will prompt the user again, but “Vitaliy Konyukhov” will be accepted.

The IP address (and, potentially, the port) and the full name are stored in the config.ini configuration file in the same folder as the application, and the stored information is presented to the user in the prompts in the future.

After the prompts, the application shows a window. This window cannot be resized, minimized, and always stays on top. This has been tested when the program launched.

By launching the application multiple times, it was verified that the ID is being generated randomly. The examples of the IDs are 794414, 099130, 145541, 389899, 071568, which demonstrates that ID numbers are random and consist of 6 digits.

The application attempts to connect to the server using the specified IP address (and port, if specified) and shows the user the yellow “Connecting” message and tries connecting indefinitely until the connection is established. After the connection is established, the application window shows a green “Connected” message. If the connection disappears, the application shows the yellow “Connecting” message again and tries to reconnect. When the server becomes reachable, the application successfully connects to it and shows a green “Connected” message again.

5.2. Server Application Testing

During the first launch of the application, the application shows the following message: “The certificates for secure TLS connection are not found. They will be generated now. This should take less than a minute.” This means that the application will launch the OpenSSL application to generate self-signed certificates that are required for the TLS secure connection. After a couple of seconds, the application shows the next prompt.

If the openssl.exe file is missing or the server is unable to generate those certificates, it shows the following message: “The certificates cannot be generated. Check if the openssl.exe file is present and the path does not contain spaces.” In this case, the application exits since it is unable to function without the certificates.

When the server application is launched, the user is prompted to choose a port for the server. By default, the user is presented with the port of 23000 but it can be changed. The port number must be an integer between 1 and 65535.

Testing the application with the following values as the port resulted in getting another prompt:

- check
- 0
- 65536
- An empty value

The selected port is stored in the config.ini configuration file and suggested to the user in the prompt in the future.

After the port is specified, the application displays the global IP address of the computer as well as the local IP addresses. The proctor can select one, copy it from the window, and share it with the students. If there is no internet connection or the application cannot get the global IP address, the server will show the message “Unable to detect your global IP address”.

After the IP addresses are shown, the application displays the main GUI that shows the “Waiting for connections…” sign. If a student is connected, the sign disappears, and the window shows the data about the student. If another student is connected, the window splits into two sections located horizontally, where the section on the right displays the data about the second student. If the third student is connected, the window becomes a 2x2 grid, and the data is displayed below the first student. The fourth student will be displayed in the bottom right corner. If the fifth student is connected, the third column will appear, and so on. The application locates the data in a grid, keeping the window square.

Each student’s window is highlighted in either green or red color, where green means that the student is connected and red means that the student got disconnected. This was tested by connecting to the server, verifying that it shows a green window, then closing the client and the window would become red.

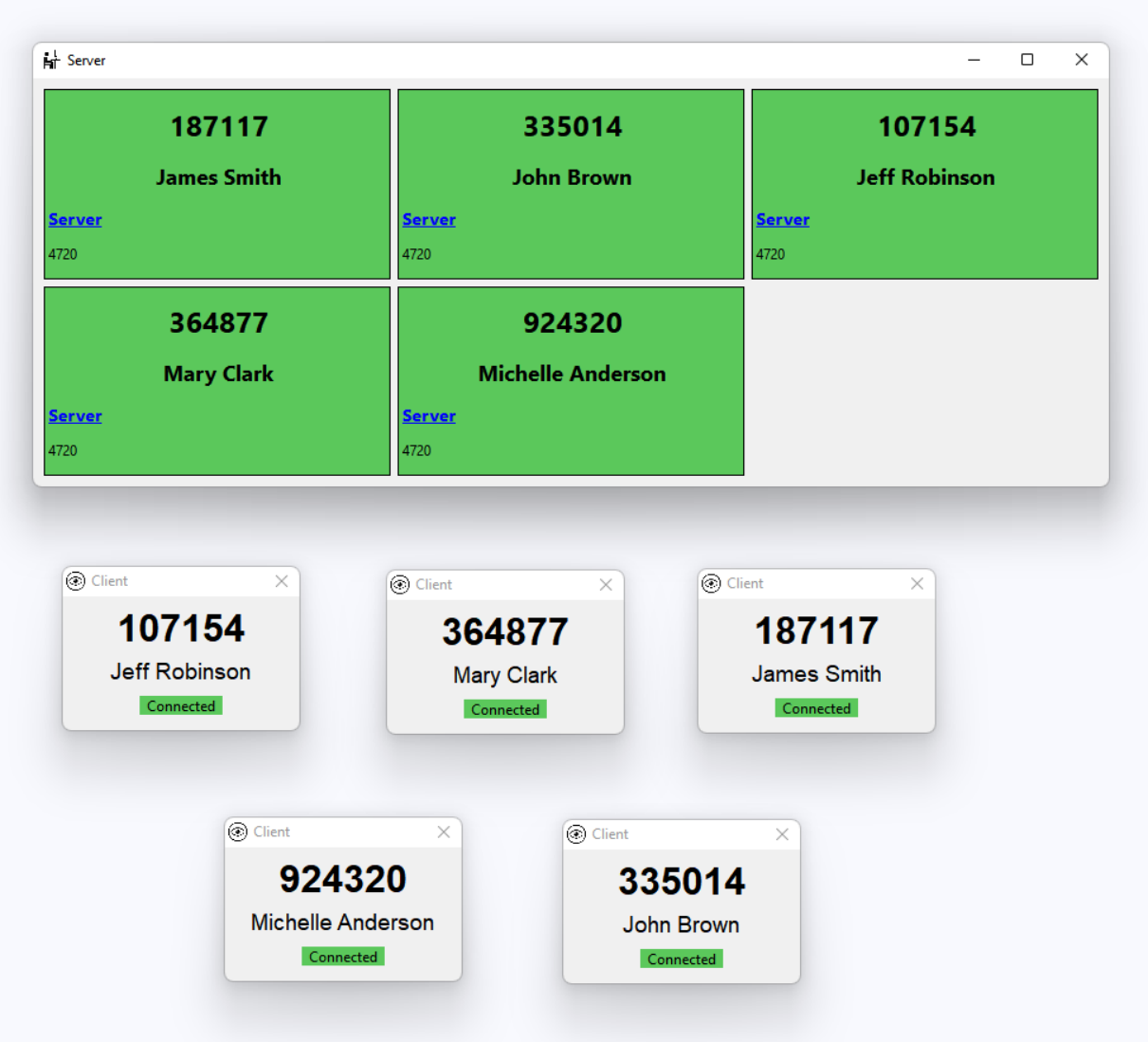

Figure 9 – Example of Running the Server and Multiple Client Applications Simultaneously.

Figure 9 demonstrates that five clients are successfully connected to the server and are sending the metrics.

5.3. Testing of the Data Collection and Processing

After the client is connected to the server, it starts collecting and sending multiple metrics to the server. The server interprets and processes the received data and displays the results in the graphical user interface. The testing process includes verifying that each of the metrics produces the expected result on the server application.

The first entry in the transmitted data that was tested is the random ID. The client application displays the random ID in the GUI and the server displays the same ID in the client's window, which confirms that the random ID transmission is working correctly.

The next entry is the student's full name. It is also displayed on the client's user interface and the same name is being displayed on the server, which proves that the transmission of the full name is working as intended.

Another metric sent to the server is the window title. The client application reads the title of the active window and sends it to the server, which checks it for suspicious keywords. To test this aspect, the student is connected to the server and opens different windows. The windows' titles are updated in the student's window on the server's GUI, always showing the title of the student's currently active window, which is demonstrated in Figure 10.

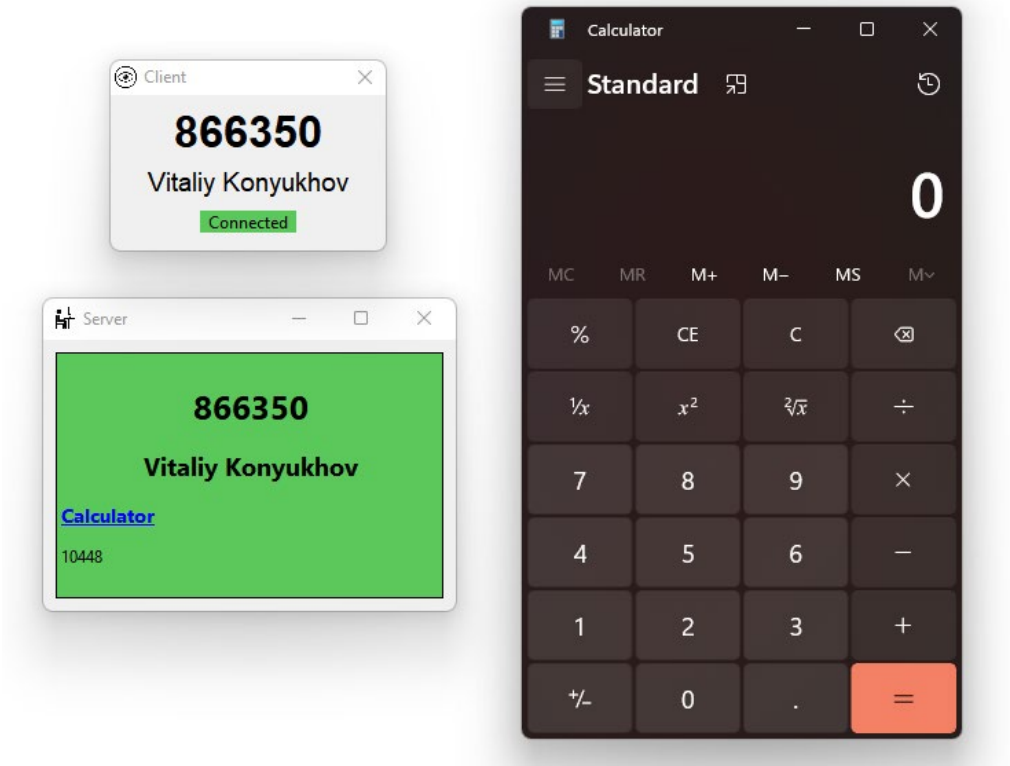

Figure 10 – Example of the Active Window Title Transmission.

One caveat of the current code implementation is that the Cyrillic alphabet is not supported. Since the Cyrillic alphabet requires two bytes to encode a letter and the Latin alphabet requires only one byte, the client sends the text as ASCII (American standard code for information interchange) encoded string which reduces the message size.

The testing verified that the window title is highlighted in red if it contains a word that is stored in the suspicious keywords list, and not highlighted otherwise. This list contains a few words by default. Existing words can be removed from the list, and new words can be added.

When the student window is displayed and is showing the student information, the window title is always written as blue underlined text signifying that it is a clickable link. When the user clicks on it, a window will appear. The words that make up the displayed window title appear in this window as separate buttons stacked vertically. The words that are in the suspicious keyword list are written in red font color and others are in black font color. When the button with the red font is clicked, the keyword is removed from the list and the window titles containing it are no longer highlighted in red. When the button with black font is clicked, the corresponding word is added to the list, and the window titles containing this word are highlighted in red. Whenever a word is added or removed from the list, the corresponding variable with the list of suspicious keywords in the config file is updated, which is verified by opening the config.ini file. The suspicious keyword modification functionality is demonstrated step by step in Figure 11:

1) The proctor clicks on the blue underlined window title
2) Selects the “Quizlet” button
3) The window title is considered suspicious since it contains the “Quizlet” keyword
4) The proctor clicks on the title again and selects the red “Quizlet” button
5) The window title is not suspicious since the “Quizlet” keyword was removed

Figure 11 – Example of Adding and Removing Suspicious Keywords.

The next entry in the transmitted data is the active window's ID number, which is displayed in the server's GUI inside the student's window. It is verified by switching windows on the student's computer and observing the change of the window ID number on the server.

Another metric is the number of monitors, which is sent to the server as an integer number. If the number of monitors is one, the server does not warn the proctor, and if the number is greater than one, the server displays a red message stating the number of monitors, which has been tested using a computer with two monitors.

Next is the information about remote session detection. If the client application is launched from a remotely controlled computer over Remote Desktop Protocol (RDP),

the server displays a “REMOTE SESSION DETECTED!” message in the GUI, which was tested by running the server in a remote session, as shown in Figure 12.

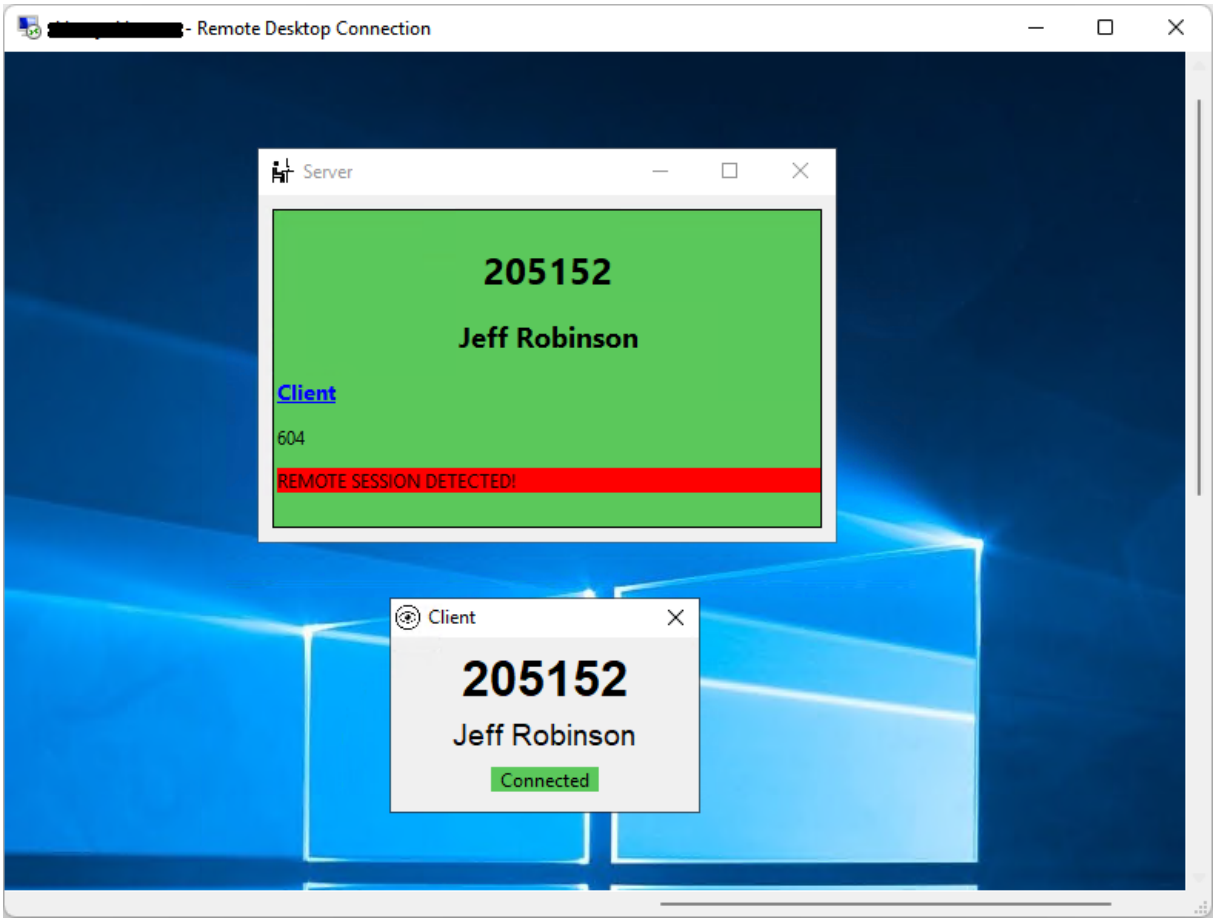

Figure 12 – Example of the Detection of a Remote Session.

The next metric sent from the client to the server is the manufacturer and the model of the computer. This data allows the server to detect if the client application is launched on a virtual machine by checking if the metric contains words like VirtualBox or VMware, which signify the use of a respective virtual hypervisor. This functionality was tested by running the client application inside a VirtualBox VM, as shown in Figure 13.

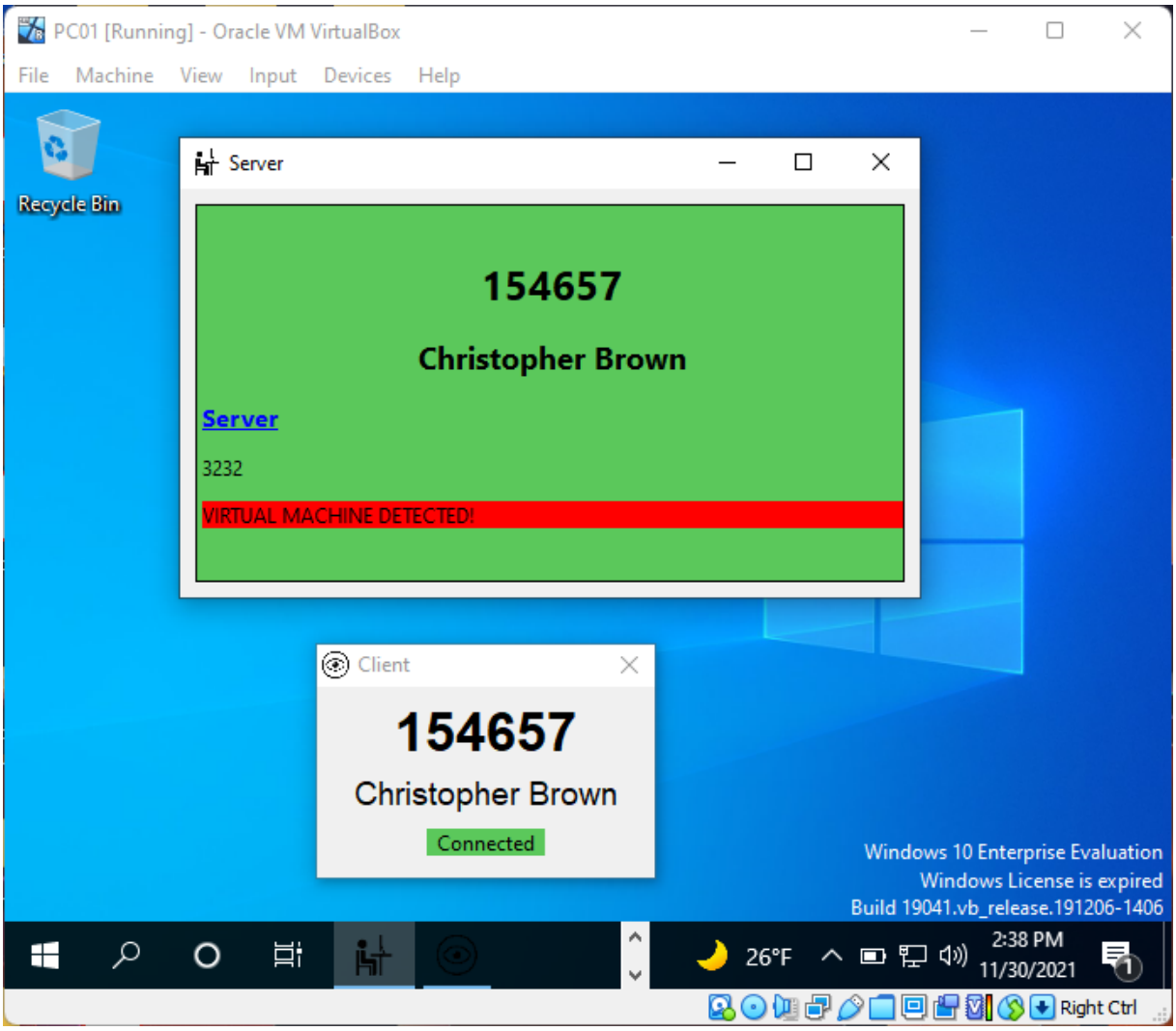

Figure 13 – Example of the Detection of a Virtual Machine.

The next data entry is the timestamp in seconds since Epoch, which is used to detect if the client has disconnected. This was verified by connecting to the server and closing the client application. The server turned the respective student's window red within a few seconds.

The last metric sent from the client is the list of running processes. When the server receives this metric, it checks if the list contains any processes from the forbidden applications list. If any forbidden applications are present, it displays the "FORBIDDEN APPS DETECTED:" message along with the names of such applications. This was tested by running the applications from the forbidden list on the computer that runs the client application, shown in Figure 14.

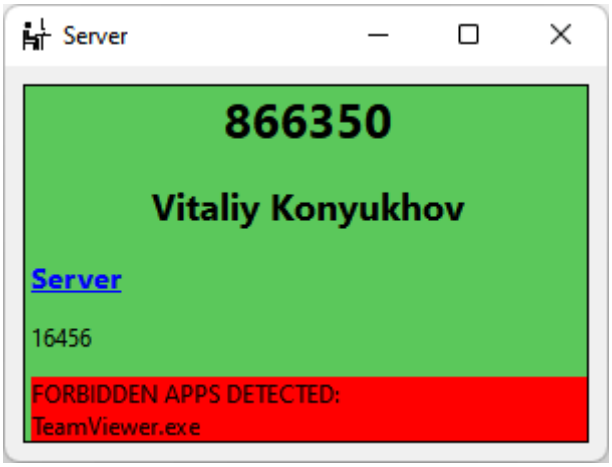

Figure 14 - Example of the Detection of a Running Suspicious Application.

If any of the metrics are suspicious or, in other words, if there are any red messages on the server regarding any student, the information regarding that student is saved in the log file in the logs folder of the server. The file can be opened with a browser to view any recorded suspicious activity, which is demonstrated in Figure 15.

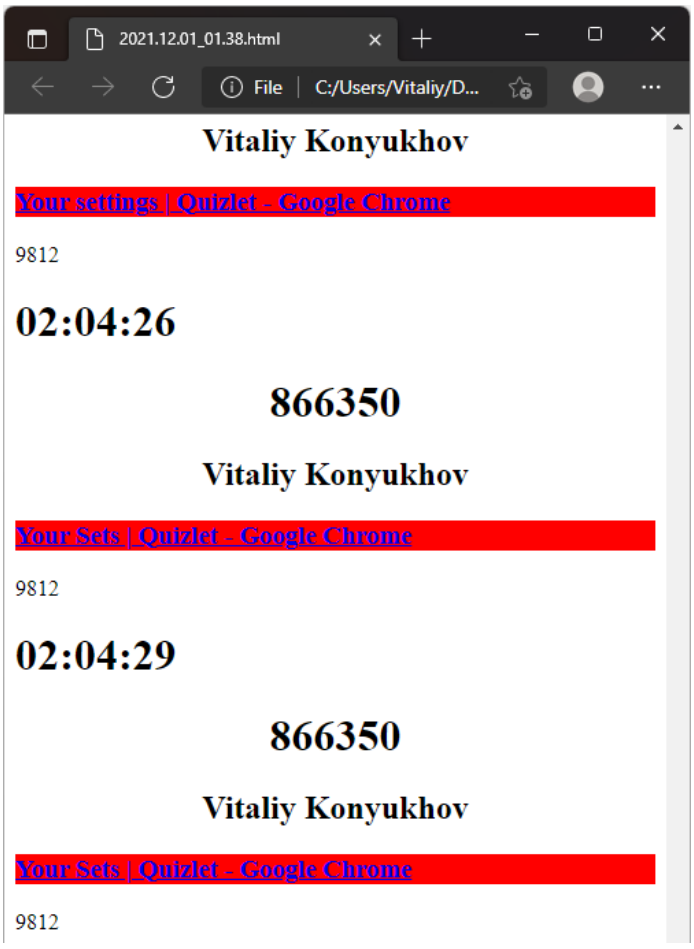

Figure 15 – View of the Recorded Suspicious Activity in the Log File.

5.4. Secure Connection Testing

The communication between the client and the server is encrypted using TLS protocol version 1.3. This is a requirement imposed by the sensitivity of the transmitted data. The connection must be tested to confirm that the data is sent securely.

To verify that the TLS is working properly on the server application, the OpenSSL client tool was used. The command “openssl.exe s_client -connect 127.0.0.1:23000” allows us to connect to the server, perform TLS handshake, verify the certificates, and confirm that TLSv1.3 is working correctly.

To verify that the data transmission is done over TLS version 1.3, the Wireshark software was used. It allows us to intercept the network traffic and examine it. To test the secure data transmission, we start the Wireshark software and configure it to monitor local loopback traffic. Then we start and configure the client and the server on the same computer and monitor the traffic in Wireshark. We notice the exchange of data registered in Wireshark, and we can inspect this data. The first few packets are the TLS handshake process, and the following packets are the data sent from the client to the server. Reading the packets with the data, we see that they are encrypted with TLSv1.3, and it is impossible to read the messages sent by the client since TLS encrypts the plaintext communication channel using Advanced Encryption Standard (AES), as shown in Figure 16.

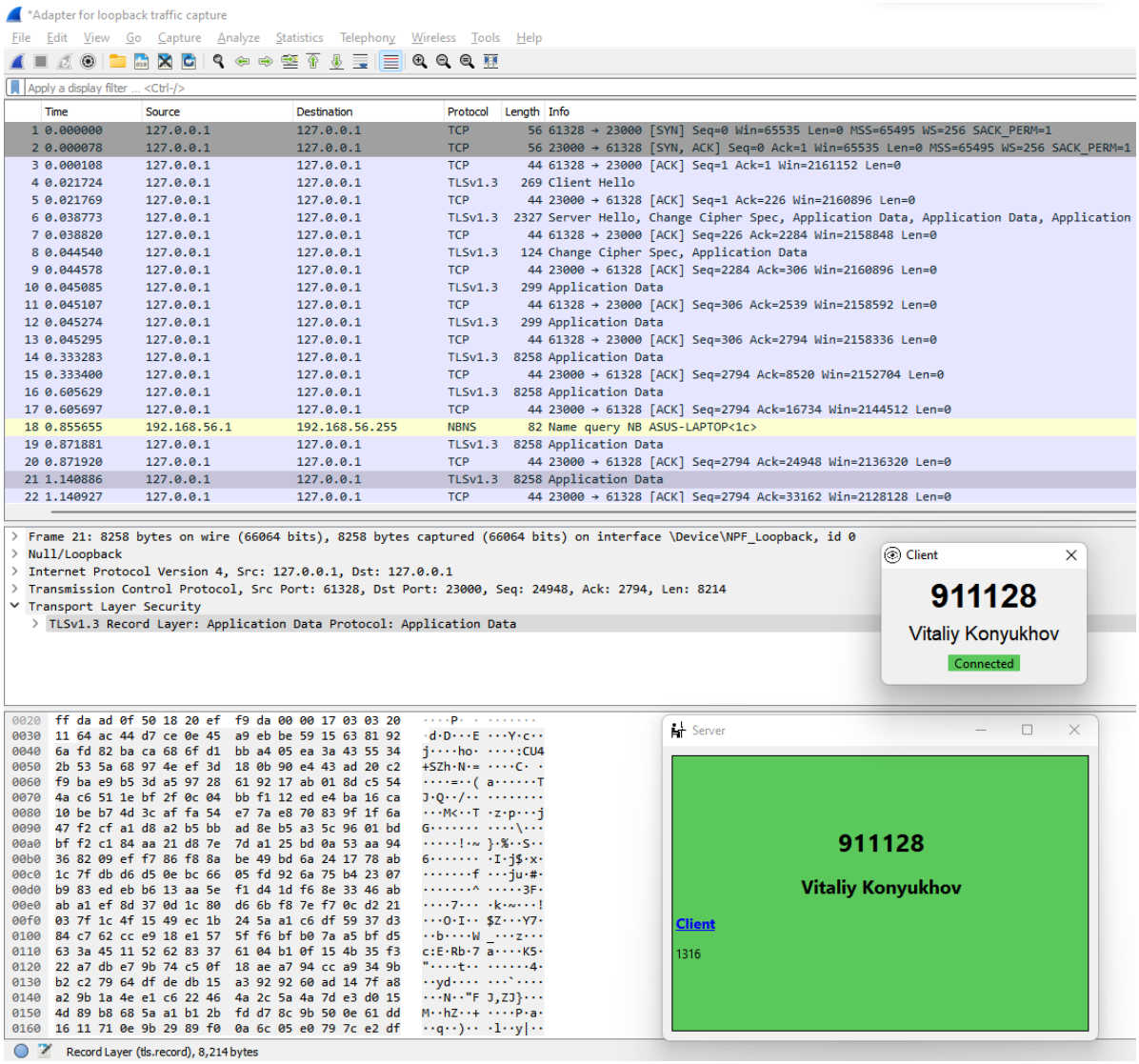

Figure 16 – Wireshark Packet Recording That Demonstrates the Use of TLS Version 1.3.

6. Results and Conclusion

Apart from creating the client-server applications for cheating prevention on online exams, this project allowed me to master my abilities to properly structure and develop programming code. It gave me firsthand experience of following the code design principles and finding and fixing bugs throughout the development. It allowed me to learn Windows APIs such as Winsock and WinINet, Windows libraries such as tlhelp32, wininet. It introduced me to the open-source cryptography and secure communication library OpenSSL that allows implementing TLS protocol, and most importantly, this project allowed me to learn the Qt Framework that I have not used before. I realized that Qt Framework is a powerful tool that allows creating advanced

applications with C++ that have flexible graphical user interfaces and use a variety of Qt libraries that provide many features to the programmer.

There were several problems that I encountered throughout the development of the application. All of the used libraries and the framework have been new to me, and I needed to learn them from scratch. The only part that I had experience in was the C++ programming language, although due to the Qt Framework's implementation, many of the familiar variable types and functions required for the code were replaced with different ones that work better or provide more functionality in Qt. After some preliminary research about the subjects, I learned how to use the framework and the libraries. I used the online documentations provided with the framework, the APIs, the libraries to write the code.

There were multiple issues discovered during the development:

- The client would not connect to the server
- The TLS implementation would cause the applications to crash
- The server would interpret the data incorrectly
- The server would not generate self-signed certificates

However, all issues and bugs were fixed in the final version of the applications.

The client application was developed to the full extent, implementing all functional and non-functional requirements proposed in section 2. The process of running the application is shown in Figure 17, which shows that it works as intended and provides an intuitive and useful user interface. I am convinced this application can be distributed to students in the future to be used during online exams.

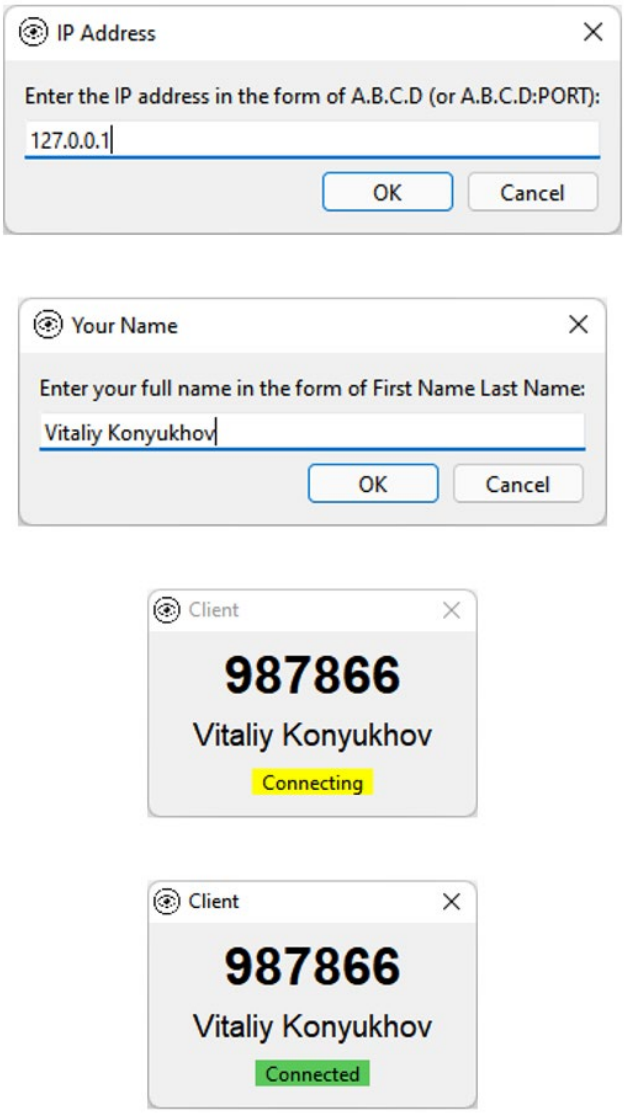

Figure 17 – Example of the Process of Running the Client Application.

The server application was also fully developed, meeting all functional and non-functional requirements proposed in the second section of this report. The application works even better than I anticipated thanks to the Qt Framework, providing a functional and intuitive graphical user interface demonstrated in Figure 18. This application is intended to be used by proctors, and I think that it performs all the tasks assigned to it. Due to the simplicity of the use of the application, it can be provided to professors for use during online exams, and it will simplify the process of proctoring such exams.

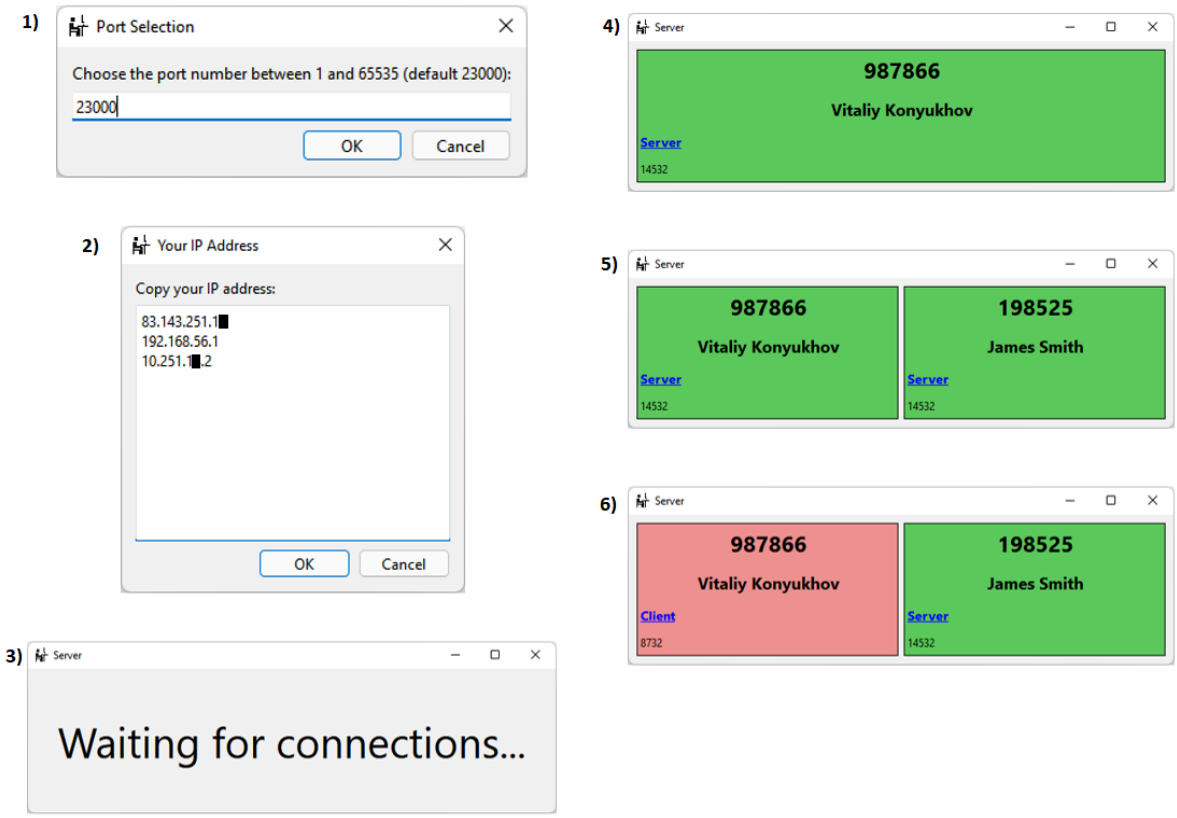

Figure 18 - Example of the Process of Running the Server Application with One Disconnected Client.

The client and the server applications include all features that I found useful for their functioning. However, additional features may be added if Professors find them necessary for their needs. One example that would improve the application is the use of a centralized server with a web platform for all proctors, which would allow accessing the student activity through a web browser.

The software package allows the proctor to see the names of connected students and the titles and ID numbers of students' active windows. The proctor is warned if the application detects any suspicious keywords in the title of the active window and allows adding and removal of these keywords. If the student is using multiple monitors, running inside a remote session or a virtual machine, or has any forbidden applications running, the proctor will be warned. All of these warnings along with a timestamp will be saved in a log file, which can be opened in a browser.

The software package developed throughout this senior project provides a powerful tool for cheating prevention during online exams. It is free, open-source, reliable, and developed within the walls of AUBG. The information that the server application provides to the proctor reliably detects multiple forms of cheating, therefore, the software can decrease or completely get rid of any cheating by students during online exams. I think that this software can be used by the University to simplify the process of proctoring online examinations for Professors and prevent students from cheating.

7. References

[1] “C and C++ reference,” *cppreference.com.* [Online]. Available: https://en.cppreference.com/w/. [Accessed: 21-Nov-2021].

[2] *OpenSSL Documentation.* [Online]. Available: https://www.openssl.org/docs/. [Accessed: 21-Oct-2021].

[3] O. Shrestha, “Design principles in software architecture,” *C# Corner.* [Online]. Available: https://www.c-sharpcorner.com/article/design-principles-in-software-architecture/. [Accessed: 24-Nov-2021].

[4] *Programming reference for the win32 API.* [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/api/. [Accessed: 21-Oct-2021].

[5] *Qt Documentation.* [Online]. Available: https://doc.qt.io/. [Accessed: 21-Oct-2021].

[6] *Qt Wiki.* [Online]. Available: https://wiki.qt.io/Main/. [Accessed: 21-Oct-2021].

[7] “Running the Winsock Client and Server Code Sample,” *Win32 apps | Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/winsock/finished-server-and-client-code. [Accessed: 21-Oct-2021].

[8] “Simple TLS server,” *OpenSSLWiki.* [Online]. Available: https://wiki.openssl.org/index.php/Simple_TLS_Server. [Accessed: 21-Oct-2021].

[9] “SSL/TLS Client,” *OpenSSLWiki.* [Online]. Available: https://wiki.openssl.org/index.php/SSL/TLS_Client. [Accessed: 21-Oct-2021].

[10] “Transmission Control Protocol,” *IBM Documentation.* [Online]. Available: https://www.ibm.com/docs/ro/aix/7.1?topic=protocols-transmission-control-protocol. [Accessed: 29-Nov-2021].

[11] “What happens in a TLS handshake? | SSL handshake,” *Cloudflare.* [Online]. Available: https://www.cloudflare.com/learning/ssl/what-happens-in-a-tls-handshake/. [Accessed: 29-Nov-2021].

[12] “What is Client-Server? Definition and FAQs,” *OmniSci.* [Online]. Available: https://www.omnisci.com/technical-glossary/client-server. [Accessed: 29-Nov-2021].

[13] “What is transport layer security? | TLS protocol,” *Cloudflare.* [Online]. Available: https://www.cloudflare.com/learning/ssl/transport-layer-security-tls/. [Accessed: 29-Nov-2021].

[14] “Windows networking reference - win32 apps,” *Win32 apps | Microsoft Docs.* [Online]. Available: https://docs.microsoft.com/en-us/windows/win32/wnet/windows-networking-reference. [Accessed: 21-Nov-2021].