Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина: Информационные сети. Основы безопасности

ОТЧЕТ
к лабораторной работе №4
на тему

РАЗРАБОТКА ЗАЩИЩЕННЫХ ПРИЛОЖЕНИЙ

Выполнил:
студент гр. 153504
Тимофеев К.А.

Проверил:
Лещенко Е.А.

Минск 2024

СОДЕРЖАНИЕ

Введение 3
1 Краткие теоретические сведения 4
2 Результат выполнения программы 7
Приложение А 11

ВВЕДЕНИЕ

Цель данной лабораторной работы заключается в ознакомлении с концепцией ролевого управления доступом и способами защиты программного обеспечения от существующих угроз. Научиться разрабатывать приложения, которые используют ролевое управление доступом для разграничения полномочий пользователей. Получить навыки защиты разработанной программы от несанкционированного копирования и других угроз, которым может подвергаться программное обеспечение.

Необходимо реализовать приложение с графическим интерфейсом, удовлетворяющее следующим требованиям: приложение проводит аутентификацию пользователя; каждый пользователь программы должен относиться к какой-нибудь группе пользователей (роли), членам которой доступны различные функциональные возможности программы; программа должна принимать от пользователя некоторые данные и, возможно, после некоторой обработки, отображать их. Также должна осуществляться защита от четырех типов возможных атак на приложение: переполнение буфера; атака «XSS»; принцип минимизации привилегий; Dos-атака.

Необходимо протестировать правильность работы системы защиты разработанного приложения посредством тестовых атак. Также необходимо реализовать приложение-инсталлятор, позволяющее установить на компьютер пользователя приложение, реализованное в данной лабораторной работе.

1 КРАТКИЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

SQL-инъекция (SQLi) – это уязвимость веб-безопасности, которая позволяет злоумышленнику вмешиваться в запросы, которые приложение делает к своей базе данных. Как правило, это позволяет просматривать данные, которые он обычно не может получить. Это могут быть других пользователей, или любые другие данные, доступ к которым имеет само приложение. Во многих случаях злоумышленник может изменять или удалять эти данные, вызывая постоянные изменения в содержимом или поведении приложения.

Успешная атака SQL-инъекции может привести к несанкционированному доступу к конфиденциальным данным, таким как пароли, данные кредитных карт или личная информация пользователей. Многие громкие утечки данных в последние годы стали результатом атак с использованием SQL-инъекций, что привело к ухудшению репутации и штрафам со стороны регулирующих органов. В некоторых случаях злоумышленник может получить постоянный черный ход в системы организации, что приводит к долгосрочной угрозе, которая может оставаться незамеченной в течение длительного периода времени.

Например, приложение для покупок, которое отображает товары в различных категориях. Когда пользователь нажимает на категорию "Подарки", его браузер запрашивает URL-адрес: https://uhahatbltv.com/?category=Gifts. Тогда сервер выбирает продукты данной категории и которые уже выпущены. Но если мы заменим конец запроса на «Gifts’--», то сервер продемонстрирует и невыпущенные товары, т.к. вторая часть условия будет восприниматься как комментарий.

XSS (англ. Cross-Site Scripting — «межсайтовый скриптинг») – довольно распространенная уязвимость, которую можно обнаружить на множестве веб-приложений. Ее суть довольно проста, злоумышленнику удается внедрить на страницу JavaScript-код, который не был предусмотрен разработчиками. Этот код будет выполняться каждый раз, когда жертвы (обычные пользователи) будут заходить на страницу приложения, куда этот код был добавлен.

Можно добавить JavaScript-код в поле ввода, текст из которого сохраняется и в дальнейшем отображается на странице для всех пользователей. Это может быть поле для ввода информации о себе на странице профиля социальной сети или комментарии на форуме. Злоумышленник вводит текст (и за одно вредоносный код), который сохраняется на странице. Когда другие пользователи зайдут на эту же страницу, вместе с текстом они загрузят и JavaScript-код злоумышленника. Именно в момент загрузки этот код

отработает. Конечно, уязвимость сработает, только если текст при сохранении не будет обезопасен.

DoS (от англ. Denial of Service: отказ в обслуживании) – атака на вычислительную систему, целью которой является отказ системы в обслуживании, то есть создание таких условий, при которых легальные пользователи системы не могут получить доступ к системным ресурсам либо этот доступ существенно затруднен.

Целью атак является блокировка возможности сервиса адекватно и своевременно реагировать на запросы настоящих, легитимных клиентов, вплоть до полной невозможности работы с сервисом. Причинами атаки могут быть: недобросовестная конкуренция; хактивизм – когда атакующий в большей степени мотивирован идеологическими разногласиями; вымогательство; шпионаж; политические мотивы в качестве самовыражения и т.д., а также маскировка работ по проникновению в инфраструктуру (отвлечение внимания на факт атаки, чтобы скрыть какие-то иные шумные «работы» по проникновению).

Атака ошибки канонизации (Canonicalization Attack) является одним из методов атаки на веб-приложения. Она основана на том, что веб-сервер может интерпретировать URL-адреса с разными способами в зависимости от того, как они были представлены.

Суть атаки заключается в том, чтобы использовать различные формы представления URL-адреса (например, использование символов, экранирование, обход фильтров) для обхода проверок безопасности и получения доступа к защищённым ресурсам или выполнения нежелательных операций. Это может привести к возможности выполнения инъекций, перехвата данных, повышения привилегий и другим серьезным угрозам безопасности.

Примером атаки ошибки канонизации может быть использование комбинации символов, таких как «..» для обхода ограничений доступа к файлам на сервере или использование нестандартных кодировок URL-адресов для обмана системы фильтрации.

Переполнение буфера – это тип уязвимости, возникающий, когда программа пытается сохранить в буфере (области временного хранения) больше данных, чем он может вместить. Это может привести к переполнению буфера, перезаписи смежной памяти и потенциально позволить злоумышленнику выполнить произвольный код, завершить работу программы или вызвать атаку типа "отказ в обслуживании" (DoS). Уязвимости переполнения буфера обычно встречаются в программах на C и Python, но

могут встречаться и в других языках. Аббревиатура переполнения буфера – BOF.

Основная идея принципа минимальных привилегий состоит в том, что доступ к ресурсам в системе должен быть организован таким образом, чтобы любая сущность внутри этой системы имела доступ только к тем ресурсам, которые минимально необходимы для успешного выполнения рабочей цели этой сущности, – и ни к каким другим.

На практике речь может идти о разных системах и разных сущностях внутри системы. Но с точки зрения применения принципа минимальных привилегий для обеспечения безопасности бизнеса это можно переформулировать так: любой пользователь информационной инфраструктуры организации должен иметь право доступа к тем и только к тем данным, которые необходимы для выполнения его рабочих задач.

Если какому-то пользователю для решения его задач необходим доступ к той информации, к которой у него нет доступа, то его права можно повысить. Постоянно, если это предполагает его роль, или временно, если это необходимо для разового выполнения отдельного проекта или задачи. Принцип минимальных привилегий помогает улучшить управление доступом к ресурсам внутри организации и в целом повысить безопасность информационной инфраструктуры компании.

2 РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ ПРОГРАММЫ

В результате разработки программы было создано приложение с графическим интерфейсом, удовлетворяющее следующим требованиям: приложение проводит аутентификацию пользователя; каждый пользователь программы должен относиться к какой-нибудь группе пользователей (роли), членам которой доступны различные функциональные возможности программы; программа должна принимать от пользователя некоторые данные и, возможно, после некоторой обработки, отображать их. Также должна осуществляться защита от четырех типов возможных атак на приложение: переполнение буфера; атака «XSS»; принцип минимизации привилегий; Dos-атака. На рисунке 2.1 представлен принцип работы приложения без каких-либо атак.

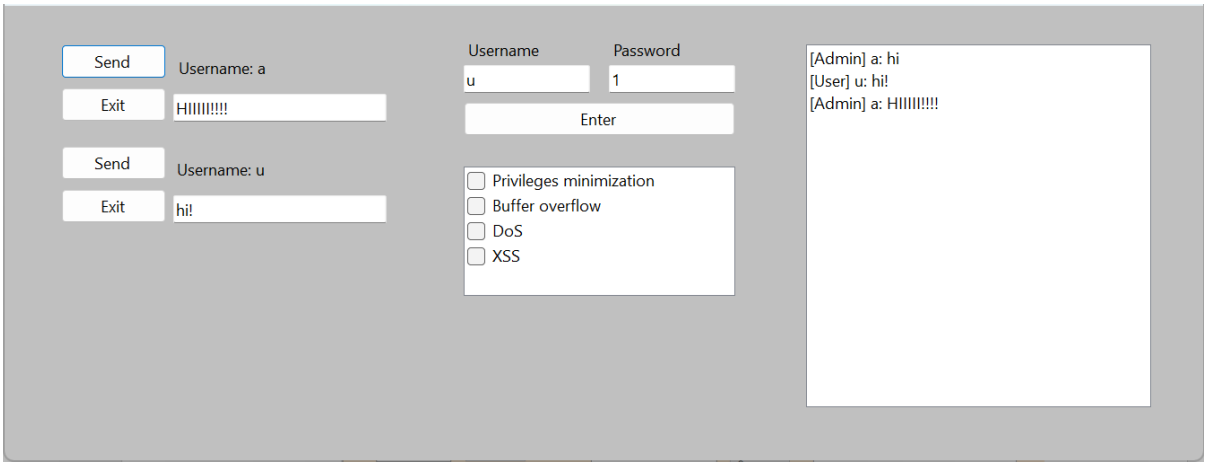


Рисунок 2.1 – Работа программы без атак

На рисунке 2.2 представлена результат работы при минимизации привилегий. Приложении присутствуют 3 роли: админ, пользователь и гость. При активации минимизации привилегий гость не имеет возможность отправлять сообщения в приложении.

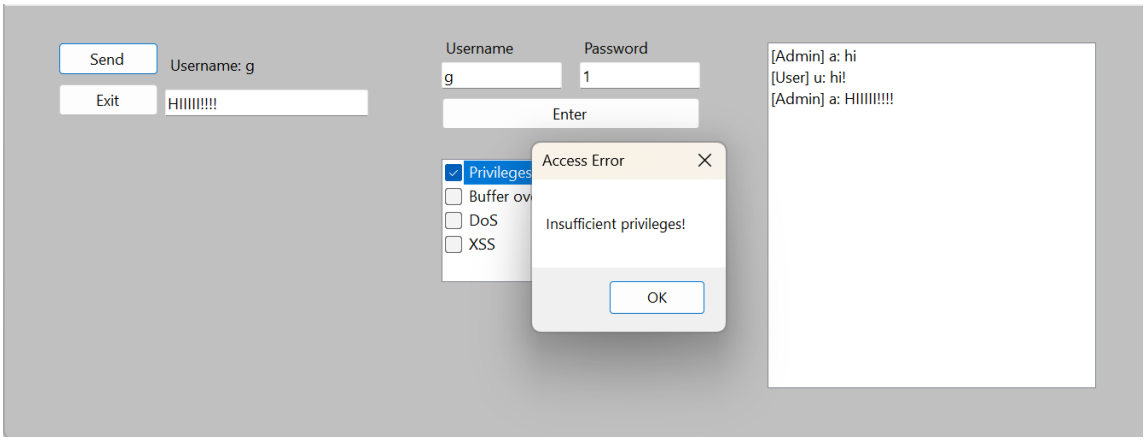


Рисунок 2.2 – Результат минимизации привилегий

На рисунке 2.3 представлена результат работы при использовании переполнения буфера. В таком случае, при отправки клиентом сообщения, оно помещается в буфер размером в 10 байтов. В таком случае, если клиент попытается отправить сообщение по количеству символов больше, чем буфер, то он попытается положить лишние данные в область других переменных.

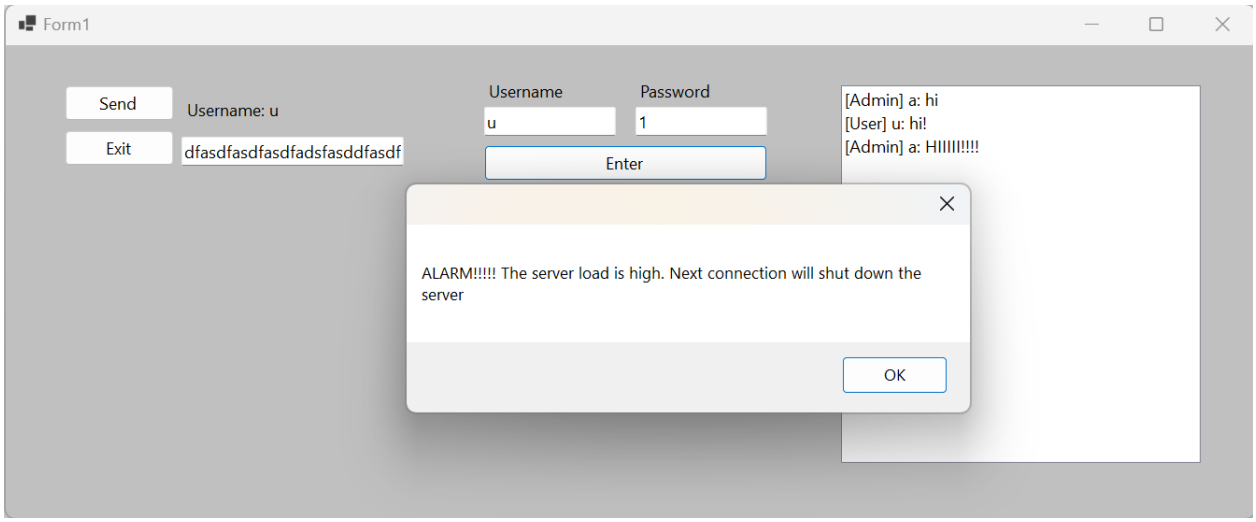


Рисунок 2.3 – Результат переполнения буфера

На рисунке 2.4 представлена результат работы при DOS-атаке. В таком случае, если уже будет авторизировано 2 человека, то следующим придется ожидать, пока не освободится очередь. Без данного пункта, если к серверу подключено больше трех человек, сервер аварийно завершает свою работу.

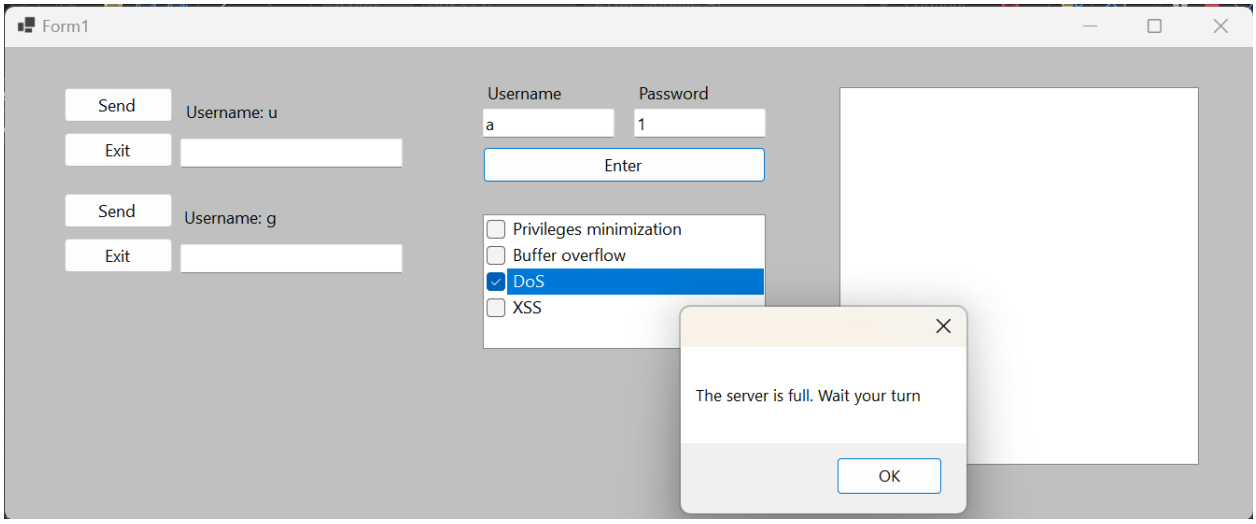
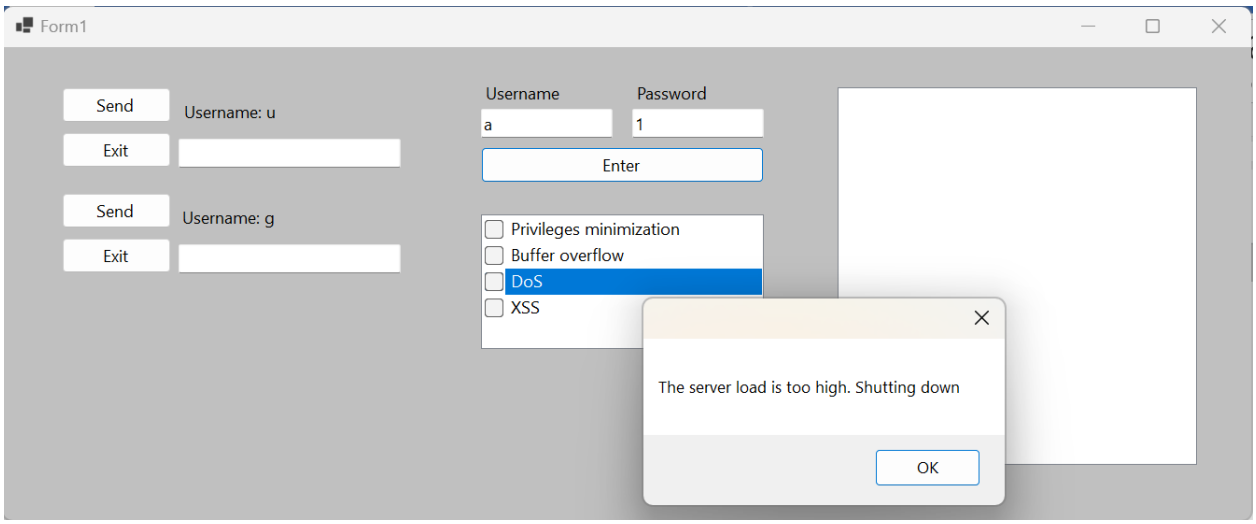


Рисунок 2.4 – Результат работы DOS-атаки

На рисунке 2.5 представлена результат работы DOS-атаки без включенной защиты.



На рисунке 2.6 представлена результат работы XSS-атаки. При активированном пункте сервер позволяет вводить сообщения только на английском и с использованием цифр.

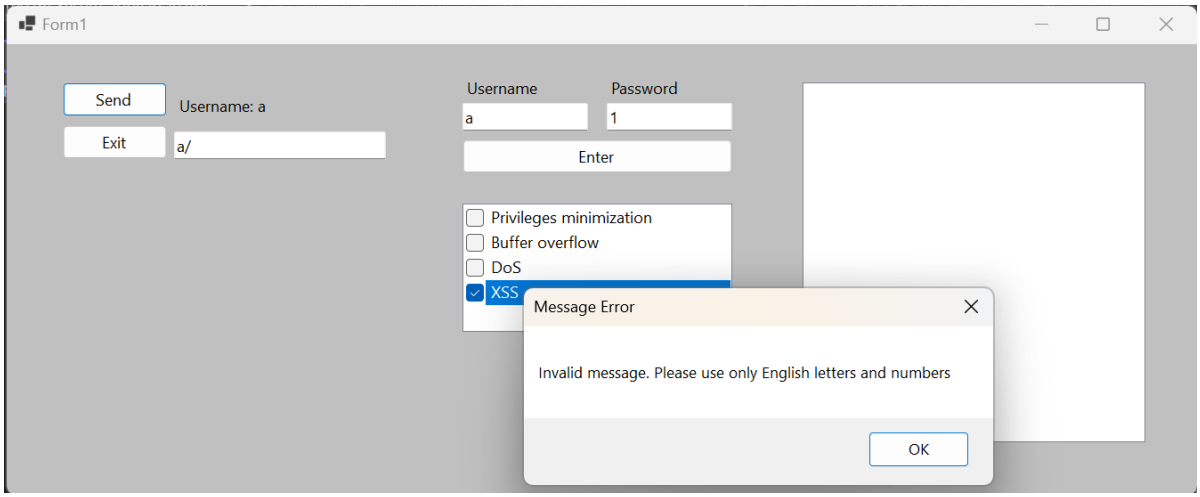


Рисунок 2.6 – Результат работы XSS-атаки

ПРИЛОЖЕНИЕ А
(обязательное)

Исходный код программы

Form1.cs

```
 using System.Text;
using System.Security.Cryptography;
using System.Windows.Forms;
using System.Linq;
using System.Threading.Tasks;
using System.Text.RegularExpressions;
using Microsoft.VisualBasic.ApplicationServices;

namespace Secure_app_lab_4_
{
    public partial class Form1 : Form
    {
        private enum Roles
        {
            Admin,
            User,
            Guest
        }

        private class DefenseSystemSwitcher
        {
            public bool PrivilegeMinimization { get; set; }
            public bool Buffer { get; set; }
            public bool Xss { get; set; }
            public bool Dos { get; set; }
        }

        private class User
        {
            public string Login { get; set; }
            public string Password { get; set; }
            public Roles Role { get; set; }
            public bool IsSignedIn { get; set; }
            public int SlotIndex { get; set; } = -1;

            public User(string login, string password)
            {
                Login = login;
                Password = password;
            }

            public override bool Equals(object obj)
            {
                return obj is User user && Login == user.Login;
            }
        }

        private List<User> usersData = new();

        private int limit = 2;

        public Form1()
```

```
        {
            InitializeComponent();

            InitializeUsersData();
            AssignControlsToUsers();
        }

        private void InitializeUsersData()
        {
            usersData.Add(new User("a", "1"));
            usersData.Add(new User("u", "1"));
            usersData.Add(new User("g", "1"));

            // Assign roles
            usersData[0].Role = Roles.Admin;
            usersData[1].Role = Roles.User;
            usersData[2].Role = Roles.Guest;
            // Other users will default to Client as it's the last in the
enum list
        }

        private void AssignControlsToUsers()
        {
            for (int i = 1; i <= usersData.Count; i++)
            {
                messageFields.Add(this.Controls[$"Message{i}"] as TextBox);
                signOutButtons.Add(this.Controls[$"SignOut{i}"] as Button);
                sendButtons.Add(this.Controls[$"Send{i}"] as Button);
                usernameLabels.Add(this.Controls[$"Username{i}"] as Label);
            }
        }

        private int GetIndex(Control sender) =>
int.Parse(sender.Name.Last().ToString()) - 1;

        private int GetLogInIndex()
        {
            return int.Parse(messageFields.Where(m => m.Visible ==
false).First().Name.Last().ToString()) - 1;
        }


        private bool IsServerLagging() => usersData.Count(user =>
user.IsSignedIn) >= limit;
        private bool IsServerAlmostLagging() => usersData.Count(user =>
user.IsSignedIn) == limit - 1;

        private DefenseSystemSwitcher GetDefenseSystemSwitcher()
        {
            return new DefenseSystemSwitcher
            {
                PrivilegeMinimization =
AttackDefensesCheckedListBox.CheckedIndices.Contains(0),
                Buffer =
AttackDefensesCheckedListBox.CheckedIndices.Contains(1),
                Dos =
AttackDefensesCheckedListBox.CheckedIndices.Contains(2),
                Xss = AttackDefensesCheckedListBox.CheckedIndices.Contains(3)
            };
        }
```

```
        // SignIn_Click, Send_Click, SignOut_Click methods remain the same,
with one change:
        // Replace IsServerLagging with IsServerLagging

        private void SignIn_Click(object sender, EventArgs e)
        {
            int index = GetLogInIndex();
            string login = Login1.Text;
            string password = Password1.Text;
            User userByName = usersData.Find(user => user.Login == login);
            DefenseSystemSwitcher defense = GetDefenseSystemSwitcher();


            if (index < 0 || index >= usersData.Count)
            {
                return;
            }

            if (IsServerAlmostLagging() && !defense.Dos)
            {
                MessageBox.Show("ALARM!!!!! The server load is high. Next
connection will shut down the server");
            }

            if (IsServerLagging())
            {
                if (defense.Dos)
                {
                    MessageBox.Show("The server is full. Wait your turn");
                    return;
                }
                else
                {
                    MessageBox.Show("The server load is too high. Shutting
down");
                    Application.Exit();
                }
            }



            if (userByName is null)
            {
                MessageBox.Show($"User {login} doesn't exists",
"Authorization error");
                return;
            }


            if (userByName is not null && userByName.IsSignedIn)
            {
                MessageBox.Show($"User {login} already authorized in
{userByName.SlotIndex + 1} slot", "Authorization error");
                return;
            }


            if (userByName != null && userByName.Password == password)
            {
                userByName.IsSignedIn = true;
                userByName.SlotIndex = index;
```

```
                messageFields[index].Visible = true;
                signOutButtons[index].Visible = true;
                sendButtons[index].Visible = true;
                usernameLabels[index].Text = "Username: " + userByName.Login;
            }
        }

        private void Send_Click(object sender, EventArgs e)
        {
            int index = GetIndex(sender as Button);
            string messageText = messageFields[index].Text;
            string answer;
            var userByIndex = usersData.Find(user => user.SlotIndex ==
index);
            DefenseSystemSwitcher defense = GetDefenseSystemSwitcher();

            if (0 > index || index >= usersData.Count)
            {
                return;
            }

            // If the user at the specified index is not found (null),
display an error message.
            if (userByIndex == null)
            {
                MessageBox.Show("The user must be authorized to send
messages", "User Error");
                return;
            }

            // If Cross-Site Scripting (XSS) defenses are active, check the
message text for invalid characters.
            if (defense.Xss)
            {
                // If the message text contains characters other than English
letters and numbers, display an error message.
                if (!Regex.IsMatch(messageText, @"^[A-z0-9]*$"))
                {
                    MessageBox.Show("Invalid message. Please use only English
letters and numbers", "Message Error");
                    return;
                }
            }

            // If Privilege Minimization defense is active and the user's
role is Client,
            // they do not have sufficient privileges to proceed.
            if (defense.PrivilegeMinimization && userByIndex.Role ==
Roles.Guest)
            {
                MessageBox.Show("Insufficient privileges!", "Access Error");
                return;
            }

            if (defense.Buffer)
            {
                int size = 10;
                char[] buf = new char[size];

                try
                {
                    messageText.CopyTo(0, buf, 0, messageText.Length);
```

```
                    messageText = string.Join("", buf);

                }
                catch (ArgumentOutOfRangeException)
                {
                    MessageBox.Show("The message exceeded the buffer. Part of
the data was written to adjacent memory", "Buffer Overflow");
                }

                messageText = string.Join("", buf);
            }

            messageText = messageText.Insert(0, $"[{userByIndex.Role}]
{userByIndex.Login}: ");

            MessagesListBox.Items.Add(messageText);
        }

        private void SignOut_Click(object sender, EventArgs e)
        {
            int index = GetIndex(sender as Button);
            var userByIndex = usersData.Find(user => user.SlotIndex ==
index);


            if (userByIndex == null)
            {
                MessageBox.Show("This slot is empty because the user is not
authorized", "Sign Out Error");
                return;
            }

            if (userByIndex != null)
            {
                userByIndex.SlotIndex = -1;
                userByIndex.IsSignedIn = false;

                messageFields[index].Visible = false;
                signOutButtons[index].Visible = false;
                sendButtons[index].Visible = false;
                usernameLabels[index].Text = "";
            }
        }
    }
}
```