PROJECT 5: TEST PLAN IMPLEMENTATION

Tintswalo Chauke, 577220

Matthew Damstra, 578374

Ozias Mulenda, 577222

Mutale Mwananshiku, 574374

Ruan van Reenen, 577071

SEN3x1

Alfred Mazorodze

17 August 2023

Table of Contents

1. Introduction

Test cases and test suites form the core of software testing. A test case is a specific condition or scenario that testers employ to verify and determine if a software function is working correctly, with a grouping of multiple thereof based on features, functionality, or other criteria results, being a test suite. The utilization of test cases and test suites enables a comprehensive assessment of overall system functionalities and performance, especially when applied in the context of specific software testing methodologies, such as Blackbox and Whitebox testing.

Black Box testing focuses on software outputs returned in response to specific inputs, specifically in scenarios where the tester has no knowledge regarding internal system mechanisms. The tester interacts with the software user interface, provides inputs, and observes the outputs, ensuring the software meets its external functional and non-functional requirements. One key advantage of this methodology is its generic nature, allowing this testing method to be applied regardless of the platform or language.

White Box testing, in contrast, delves into internal software structures. Testers have full access to the source code, evaluating logic, loops, conditions, and branches. This method ensures thorough test coverage, but it also demands a comprehensive understanding of the software's programming language and overall architecture.

The following is a test suite, which consists of two test cases executed/applied in context of Blackbox testing and two executed/applied in context of Whitebox testing, for our proposed Premier Service Solutions software system.

2. Blackbox Testing

2.1 Test Case 1: User Role-based Access Control

Objective: Verify that a user belonging to a specific category can only access system parts related to that category. Verify that admins can access every category.

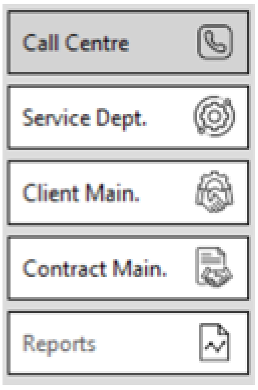
Input: Log in with a user account of a specific category. Log in as an admin.

Expected Outcome: The user should only be able to view and interact with the system sections related to their category and should be restricted from accessing other parts. Admin has access to

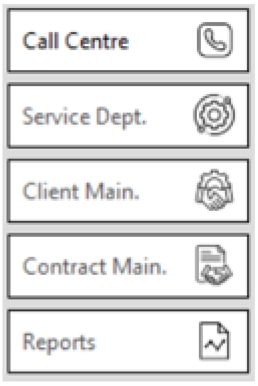
Outcome: The expected outcome was achieved

Result: Pass

All functional areas enabled with admin login (as indicated by the black text).



Only call centre area enabled with call centre user login (as indicated by the grey text). This proves true for all other categories.



2.2 Test Case 2: Log Client Problem

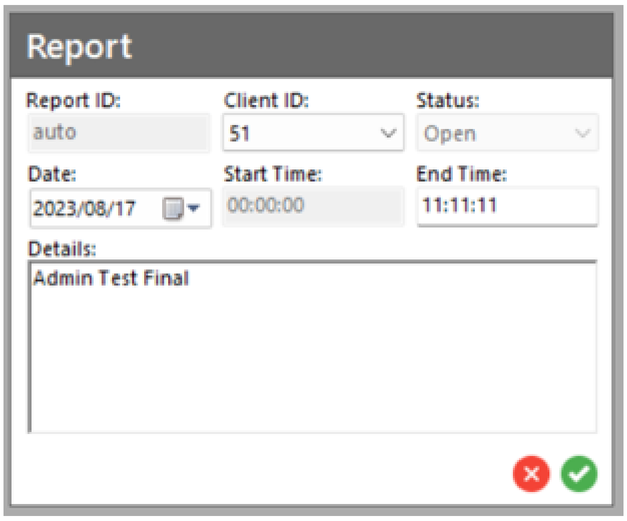
Objective: Ensure that the system can successfully log a client proble. Only admins and users related to the call centre role should be able to fulfil this function.

Input: A valid client ID, a detailed problem description.

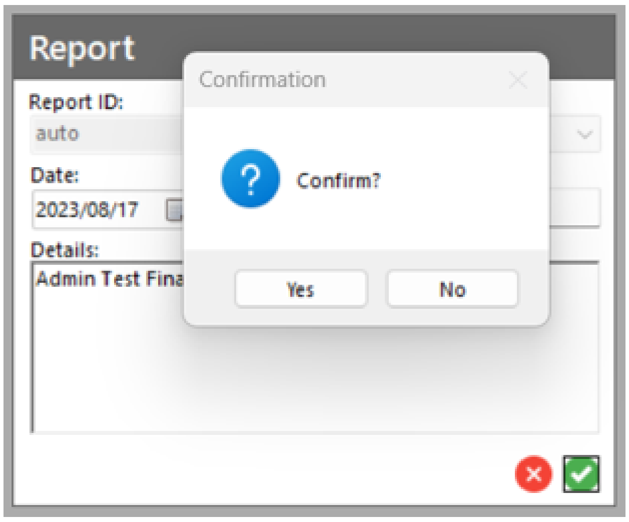
Expected Outcome: The system logs the problem and associates it with the client's record. A unique incident ID is generated.

Outcome: The expected outcome was achieved.

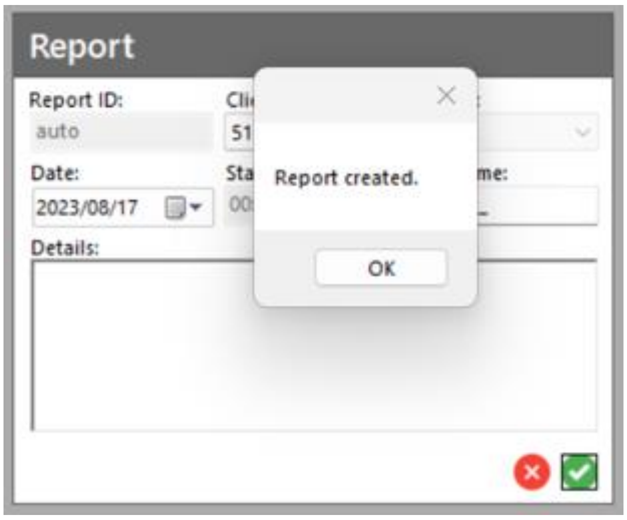
Result: Pass



Report ID: auto
Client ID: 51
Status: Open
Date: 2023/08/17
Start Time: 00:00:00
End Time: 11:11:11
Details: Admin Test Final



Report ID: auto
Date: 2023/08/17
Details: Admin Test Fina
Confirmation
Confirm?



Report ID: auto
Cli
51
Date: 2023/08/17
Sta
00
me:
Details:
Report created.

3. Whitebox Testing

3.1 Test Case 3: Internal Logic for Capturing and Maintaining Technician Details

Objective: Ensure that internal logic correctly captures, processes, and stores technician details into the database. Only users related to the service department role must be able to achieve this, as well as admins.

Prerequisites: Access to the system code and database.

Steps:

1. Examine the code responsible for capturing technician details in the service department section. Understand the expected logic paths, especially around data validation, transformation, and storage.
2. Insert breakpoints at:
   - Data validation checks (e.g. ensure required fields are not empty or invalid).
   - Any logic that transforms or processes the data before saving (e.g. converting all text to uppercase, parsing addresses into different database fields etc.)
   - Data commitment to database.
3. Input valid technician data and observe the code execution through the breakpoints and ensure:
   - All validation checks pass as expected.
   - The final data committed to the database is correctly structured and matches the input.
   - Check the SQL queries/commands used to input and retrieve technician data.

Expected Outcome: The internal logic operates correctly, all validation checks are passed, transformations are done as designed, and the data is committed to the database without issues.

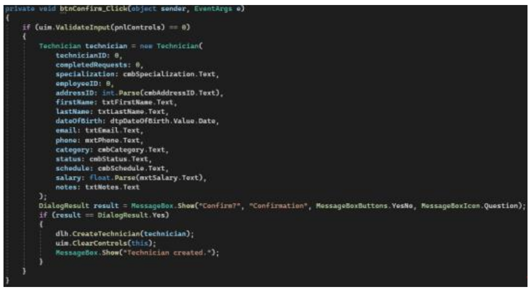
Outcome: The expected outcome was achieved.

Result: Pass

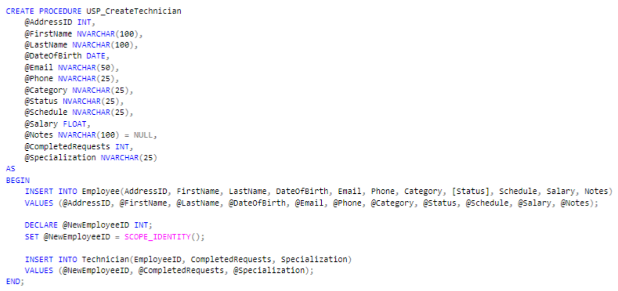
3.1.1 Screenshots

# Technician structure and capturing technician details.

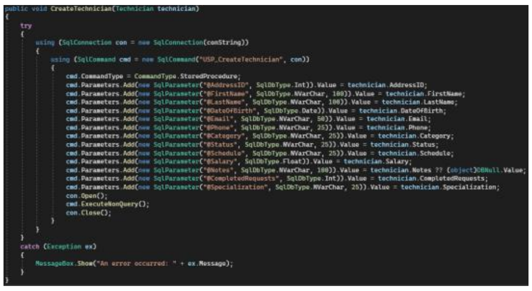
Method(s) and stored procedure(s) utilized for capturing technician details from the frmTechnician inputs and the creation of the technician object.



```
private void btnConfirm_Click(object sender, EventArgs e)
{
    if (uim.ValidateInput(pnlControls) == 0)
    {
        Technician technician = new Technician(
            technicianID: 0,
            completedRequests: 0,
            specialization: cmbSpecialization.Text,
            employeeID: 0,
            addressID: int.Parse(cmbAddressID.Text),
            firstName: txtFirstName.Text,
            lastName: txtLastName.Text,
            dateOfBirth: dtpDateOfBirth.Value.Date,
            email: txtEmail.Text,
            phone: mxtPhone.Text,
            category: cmbCategory.Text,
            status: cmbStatus.Text,
            schedule: cmbSchedule.Text,
            salary: float.Parse(mxtSalary.Text),
            notes: txtNotes.Text
        );
        DialogResult result = MessageBox.Show("Confirm?", "Confirmation", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
        if (result == DialogResult.Yes)
        {
            dlh.CreateTechnician(technician);
            uim.ClearControls(this);
            MessageBox.Show("Technician created.");
        }
    }
}
```



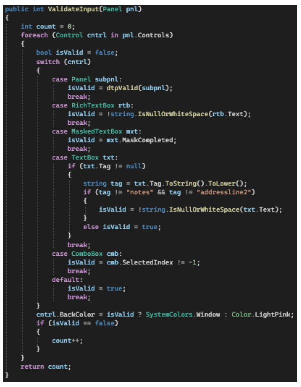
```
CREATE PROCEDURE USP_CreateTechnician
    @AddressID INT,
    @FirstName NVARCHAR(100),
    @LastName NVARCHAR(100),
    @DateOfBirth DATE,
    @Email NVARCHAR(50),
    @Phone NVARCHAR(25),
    @Category NVARCHAR(25),
    @Status NVARCHAR(25),
    @Schedule NVARCHAR(25),
    @Salary FLOAT,
    @Notes NVARCHAR(100) = NULL,
    @CompletedRequests INT,
    @Specialization NVARCHAR(25)
AS
BEGIN
    INSERT INTO Employee(AddressID, FirstName, LastName, DateOfBirth, Email, Phone, Category, [Status], Schedule, Salary, Notes)
    VALUES (@AddressID, @FirstName, @LastName, @DateOfBirth, @Email, @Phone, @Category, @Status, @Schedule, @Salary, @Notes);

    DECLARE @NewEmployeeID INT;
    SET @NewEmployeeID = SCOPE_IDENTITY();

    INSERT INTO Technician(EmployeeID, CompletedRequests, Specialization)
    VALUES (@NewEmployeeID, @CompletedRequests, @Specialization);
END;
```



```
public void CreateTechnician(Technician technician)
{
    try
    {
        using (SqlConnection con = new SqlConnection(conString))
        {
            using (SqlCommand cmd = new SqlCommand("USP_CreateTechnician", con))
            {
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.Add(new SqlParameter("@AddressID", SqlDbType.Int)).Value = technician.AddressID;
                cmd.Parameters.Add(new SqlParameter("@FirstName", SqlDbType.NVarChar, 100)).Value = technician.FirstName;
                cmd.Parameters.Add(new SqlParameter("@LastName", SqlDbType.NVarChar, 100)).Value = technician.LastName;
                cmd.Parameters.Add(new SqlParameter("@DateOfBirth", SqlDbType.Date)).Value = technician.DateOfBirth;
                cmd.Parameters.Add(new SqlParameter("@Email", SqlDbType.NVarChar, 50)).Value = technician.Email;
                cmd.Parameters.Add(new SqlParameter("@Phone", SqlDbType.NVarChar, 25)).Value = technician.Phone;
                cmd.Parameters.Add(new SqlParameter("@Category", SqlDbType.NVarChar, 25)).Value = technician.Category;
                cmd.Parameters.Add(new SqlParameter("@Status", SqlDbType.NVarChar, 25)).Value = technician.Status;
                cmd.Parameters.Add(new SqlParameter("@Schedule", SqlDbType.NVarChar, 25)).Value = technician.Schedule;
                cmd.Parameters.Add(new SqlParameter("@Salary", SqlDbType.Float)).Value = technician.Salary;
                cmd.Parameters.Add(new SqlParameter("@Notes", SqlDbType.NVarChar, 100)).Value = technician.Notes ?? (object)DBNull.Value;
                cmd.Parameters.Add(new SqlParameter("@CompletedRequests", SqlDbType.Int)).Value = technician.CompletedRequests;
                cmd.Parameters.Add(new SqlParameter("@Specialization", SqlDbType.NVarChar, 25)).Value = technician.Specialization;
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show("An error occurred: " + ex.Message);
    }
}
```

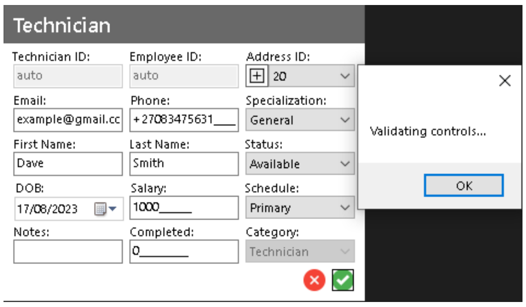
# Validation, inserting new technician, breakpoints.

All form controls are validated, ensuring required fields are not empty.



```
public int ValidateInput(Panel pnl)
{
    int count = 0;
    foreach (Control cntrl in pnl.Controls)
    {
        bool isValid = false;
        switch (cntrl)
        {
            case Panel subpnl:
                isValid = dtpValid(subpnl);
                break;
            case RichTextBox rtb:
                isValid = !string.IsNullOrWhiteSpace(rtb.Text);
                break;
            case MaskedTextBox mxt:
                isValid = mxt.MaskCompleted;
                break;
            case TextBox txt:
                if (txt.Tag != null)
                {
                    string tag = txt.Tag.ToString().ToLower();
                    if (tag != "notes" && tag != "addressline2")
                    {
                        isValid = !string.IsNullOrWhiteSpace(txt.Text);
                    }
                    else isValid = true;
                }
                break;
            case ComboBox cmb:
                isValid = cmb.SelectedIndex != -1;
                break;
            default:
                isValid = true;
                break;
        }
        cntrl.BackColor = isValid ? SystemColors.Window : Color.LightPink;
        if (isValid == false)
        {
            count++;
        }
    }
    return count;
}
```

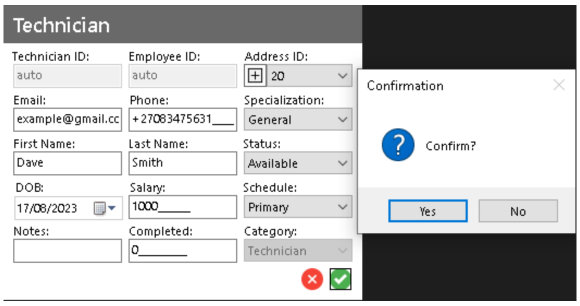
A breakpoint is used to indicate the successful start of this process. And if successful, the tester/user may either confirm or reject submission.



Technician

Technician ID: auto	Employee ID: auto	Address ID: 20
Email: example@gmail.cc	Phone: +27083475631___	Specialization: General
First Name: Dave	Last Name: Smith	Status: Available
DOB: 17/08/2023	Salary: 1000____	Schedule: Primary
Notes:	Completed: 0______	Category: Technician

Validating controls...



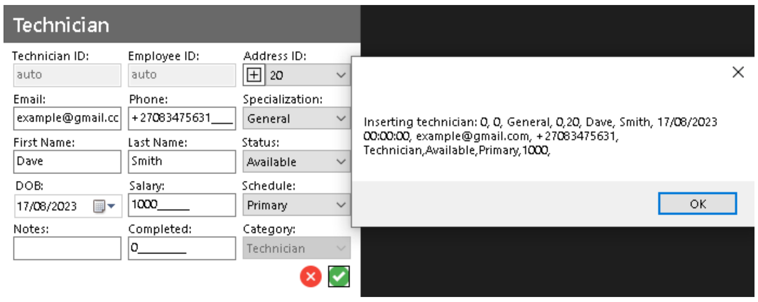
Technician

Technician ID: auto	Employee ID: auto	Address ID: 20
Email: example@gmail.cc	Phone: +27083475631___	Specialization: General
First Name: Dave	Last Name: Smith	Status: Available
DOB: 17/08/2023	Salary: 1000____	Schedule: Primary
Notes:	Completed: 0______	Category: Technician

Confirmation

Confirm?

A breakpoint showing technician details is used at the database insert occurrence within the CreateTechnician(Technician) method, enabling correctness verification.



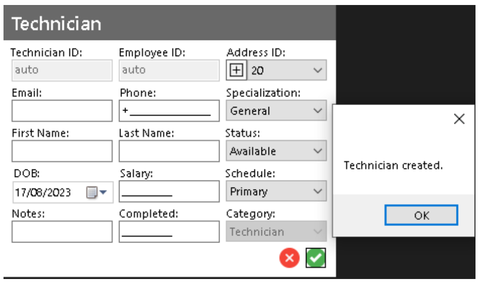
Technician

Technician ID: auto	Employee ID: auto	Address ID: 20
Email: example@gmail.cc	Phone: +27083475631___	Specialization: General
First Name: Dave	Last Name: Smith	Status: Available
DOB: 17/08/2023	Salary: 1000_____	Schedule: Primary
Notes:	Completed: 0_______	Category: Technician

```
Inserting technician: 0, 0, General, 0,20, Dave, Smith, 17/08/2023
00:00:00, example@gmail.com, +27083475631,
Technician,Available,Primary,1000,
```

OK

A message indicating success is displayed if successful.



Technician

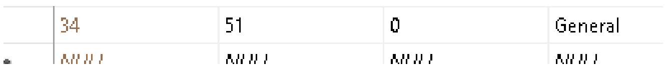
Technician ID: auto	Employee ID: auto	Address ID: 20
Email:	Phone: +__________	Specialization: General
First Name:	Last Name:	Status: Available
DOB: 17/08/2023	Salary: _______	Schedule: Primary
Notes:	Completed: _______	Category: Technician

```
Technician created.
```

OK

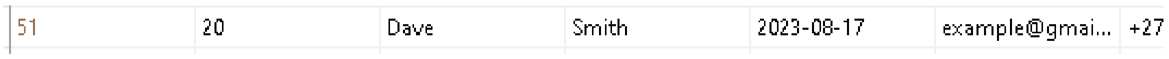
# Database verification.

Technician table:



34	51	0	General
[illegible]	[illegible]	[illegible]	[illegible]

Employee table:



51	20	Dave	Smith	2023-08-17	example@gmai...	+27

3.2 Test Case 4: Internal Logic for Logging in

Objective: Validate that the system's internal mechanisms correctly authenticate users during the login process.

Prerequisites: Access to system code and tools to monitor code execution paths.

Steps:

1. Examine the code responsible for user authentication:
    - Understand the logic of how credentials (username/password) are verified.
    - Identify how passwords are hashed and compared with stored hashed passwords.
    - Examine session management (session creation at login).
2. Insert Breakpoints:
    - The point where user input is collected and validated.
    - The mechanism that hashes the entered password and compares it with the stored hash.
    - Logic that creates a new user session or assigns a session token upon successful login.
    - Any error-handling logic for failed logins (e.g., user doesn't exist, wrong password).
3. Input valid user credentials and observe code execution, ensuring that:
    - Input validation is processed.
    - Correct password hashing algorithm is used.
    - User session/user is correctly initiated.
4. Database Logic Verification:
    - Validate that there's proper encryption or hashing of passwords in the database.

Expected Outcome: The internal logic correctly authenticates user credentials and manages user sessions. Passwords are securely hashed and compared without exposing sensitive data. User sessions are correctly initiated. Database interactions are secure and efficient.

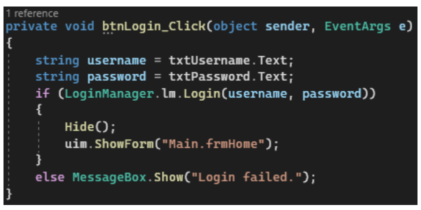
Outcome: The expected outcome is achieved.

Result: Pass

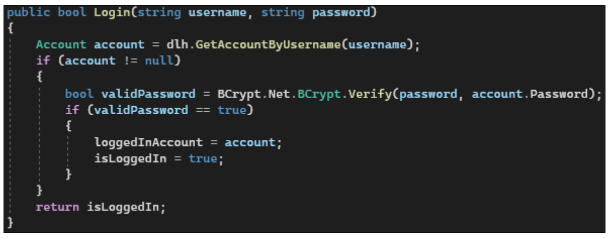
3.2.1 Screenshots

# User Authentication & Database Verification.

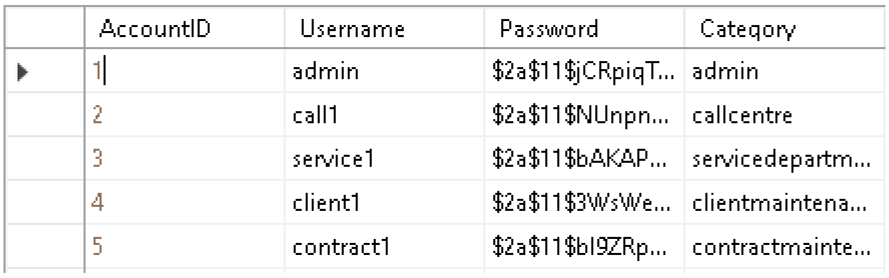
LoginManager instance Login method is utilized to verify entered details against accounts contained within the database, along with BCrypt.Verify. BCrypt is the specific function used for password hashing.



```
1 reference
private void btnLogin_Click(object sender, EventArgs e)
{
    string username = txtUsername.Text;
    string password = txtPassword.Text;
    if (LoginManager.lm.Login(username, password))
    {
        Hide();
        uim.ShowForm("Main.frmHome");
    }
    else MessageBox.Show("Login failed.");
}
```



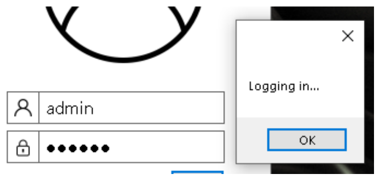
```
public bool Login(string username, string password)
{
    Account account = dlh.GetAccountByUsername(username);
    if (account != null)
    {
        bool validPassword = BCrypt.Net.BCrypt.Verify(password, account.Password);
        if (validPassword == true)
        {
            loggedInAccount = account;
            isLoggedIn = true;
        }
    }
    return isLoggedIn;
}
```



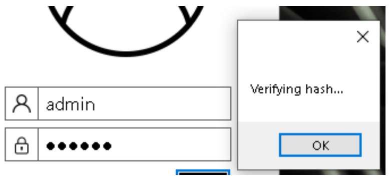
	AccountID	Username	Password	Category
▶	1	admin	$2a$11$jCRpiqT...	admin
	2	call1	$2a$11$NUnpn...	callcentre
	3	service1	$2a$11$bAKAP...	servicedepartm...
	4	client1	$2a$11$3WsWe...	clientmaintena...
	5	contract1	$2a$11$bl9ZRp...	contractmainte...

# Breakpoints

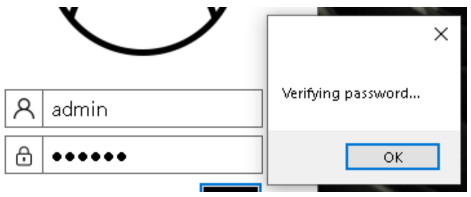
Various breakpoints showing the process of password verification up until successful login.



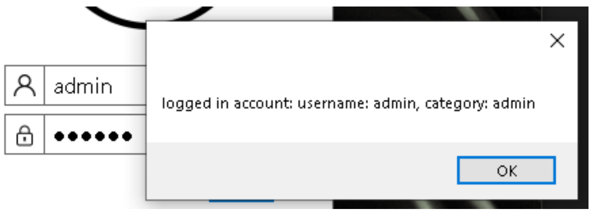
admin
••••••
Logging in...



admin
••••••
Verifying hash...



admin
••••••
Verifying password...



admin
••••••
logged in account: username: admin, category: admin

4. References

Check Point Software Technologies, 2023. *What is Black Box Testing?.* [Online]
Available at: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-penetration-testing/what-is-black-box-testing/
Check Point Software Technologies, 2023. *What is White Box Testing?.* [Online]
Available at: https://www.checkpoint.com/cyber-hub/cyber-security/what-is-white-box-testing/