Table of Contents

Introduction 2
Password Security 2
Combination Requirement 2
Strength 2
Blocklist 4
Validating the password 5
Multi-Factor Authentication 5
Copy/paste Usages 6
AES Encryption 6
Reset Password 8
Password Repetition Prevention 8
Captcha Feature 9

Introduction

The prototype system for registration process was made with Windows Forms .Net Framework in C#. The prototype has been designed by considering a set of password policies and procedures that includes acceptation of only certain combination of password, determination of strength, encryption of password, blocking the use of certain passwords and multi factor authentication to login. Also, to make sure that the registration request is made by a human user, the number captcha function has been implemented.

Password Security

Combination Requirement

Enforcing specific combination prevents users from using common words as their password which would make them vulnerable and weak. In this prototype, for a user to be able to register their account successfully, they must create a password with the following features. (Saranga Komanduri, 2011)

- At least 8 characters long alphanumeric
- At least 1 uppercase letter
- At least 1 lowercase letter
- At least 1 digit
- At least 1 special character

Anything that does not meet this combination will be considered invalid and not be accepted by the prototype.

```
if (Regex.IsMatch(txtpassword.Text, "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}$"))
{
    strength = lower + upper * 2 + digit + symbol * 2;
    panelwhy.Hide();
}
```

This is a part of the code which checks if the entered password meets the above mention requirement. Else it will not accept the password. More about the code has been describes below in the strength section.

Strength

Determination of password strength helps users to choose more crack-resilient passwords. In this prototype, a strength meter has been implemented in order to aware the user about how strong or vulnerable their password is. (Richard Shay, 2016) Here is how the strength is measured:

Each character of the password has a score point. So, each character is checked to see how many character belongs to each category of requirement.

```
private void txtpassword_TextChanged(object sender, EventArgs e)
{
    txtmeter.Show();
    int len = txtpassword.Text.Length;

    int lower = 0, upper = 0, digit = 0, symbol = 0;

    for (int x = 0; x < len; x++)
    {
        if (char.IsLetter(txtpassword.Text[x]) & char.IsLower(txtpassword.Text[x]))
        {
            lower++;
        }
        if (char.IsLetter(txtpassword.Text[x]) & char.IsUpper(txtpassword.Text[x]))
        {
            upper++;
        }
        if (char.IsDigit(txtpassword.Text[x]))
        {
            digit++;
        }
        if (char.IsSymbol(txtpassword.Text[x]) | char.IsPunctuation(txtpassword.Text[x]))
        {
            symbol++;
        }

    }

    if (Regex.IsMatch(txtpassword.Text, "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}$"))
```

```
if (Regex.IsMatch(txtpassword.Text, "^(?=.*?[A-Z])(?=.*?[a-z])(?=.*?[0-9])(?=.*?[#?!@$%^&*-]).{8,}$"))
{
    strength = lower + upper * 2 + digit + symbol * 2;
    panelwhy.Hide();
}

else
{
    strength = 8;
    panelwhy.Show();

}
```

Now, if the password entered satisfies the combination requirement, uppercase is 2 points, lowercase 1 point digit 1 point and special character 2 points else the point is set to 8. As per the score calculated, the color of strength meter is set.

Strength has been categorized into 3 levels; weak, average and strong. A password meeting the combination requirement with least scoring characters scores 10 points and with maximum scoring characters scores 14 points. Hence, if score is less than 10 the strength is weak, if it's more than 14 the score is strong and everything else falls on the average.

Strong password is represented by green color, average by yellow and weak by red. Password falling in weak level/red color is not accepted by the system.

```
if (strength < 10 || len < 8)
{
    txtmeter.BackColor = Color.Red;
}

else if (len > 8 && strength > 15)
{
    txtmeter.BackColor = Color.Green;
}
else
{
    txtmeter.BackColor = Color.Yellow;
}
blocklist();

if (txtmeter.BackColor == Color.Red)
{
    lxconfirm.Show();
}
else
{
    lxconfirm.Hide();

}
```

Strength determination can be modified to be much stricter by keeping 14 as the range for average. This could result in much stronger and complex password but it could make user difficult to remember. Therefore, I consider this approach to be better for implementation.

Blocklist

There are people who use very common phrases such as 'password' itself by modifying the case and filling the requirement with other common digit and symbols for password. E.g., Password@123, pa$$Word123, etc. Such password would be easier to guessed or breached into. Adding in a password blocklist can provide additional security by preventing the attackers from getting past a simple password login flow in the first place. Hence, I have created an array of such possible passwords which if matches to the entered password turns the strength meter color red.

```
public void blocklist ()
{
    string[] myArray = { "password@123", "p@ssword123","pa$$word123","pa$sword@123", "qwerty.123", "qwerty@123"};

    foreach (string ch in myArray)
    {
        if (txtpassword.Text.ToLower()== ch)
        {
            txtmeter.BackColor = Color.Red;
            MessageBox.Show("Please enter a proper password!");
        }
    }
}
```

Validating the password

The confirm password field checks if it is same as the text in password field. It is to reassure that the password typed and thought to be typed by the user is same. If it matches, the password is accepted else a message is displayed to aware the user.

```
private void txtconfirmpass_TextChanged(object sender, EventArgs e)
{
    string pass = txtpassword.Text;
    string config = txtconfirmpass.Text;
    if (pass.Equals(config))
    {
        lxconfirm.Hide();
    }
    else
    {
        lxconfirm.Show();
    }
}
```

Multi-Factor Authentication

Enforcing the use of an MFA enhances the security by requiring users to identify themselves by more than just username and password because they are vulnerable to the brute force attacks and can be stolen by the third parties. I have implemented MFA that works by requiring the most common verification factor, 6-digit one-time passwords (OTP) received via email. (Fäl, 2020)

```
private void sendcode()
{
    string email, from, pass, messagebody;
    con.Open();
    SqlCommand cmd = new SqlCommand("select * from users where username='" + txtusername.Text + "'", con);
    SqlDataReader sdr = cmd.ExecuteReader();
    if (sdr.Read())
    {
        email = (string)sdr[1];
        Random ran = new Random();
        randomcode = (ran.Next(999999)).ToString();
        MailMessage message = new MailMessage();
        to = email;
        from = "buyin.mailyou@gmail.com";
        pass = "ampbjrennekunojc";
        messagebody = "Your authentication code is " + randomcode;
        message.To.Add(to);
        message.From=new MailAddress(from);
        message.Body = messagebody;
        message.Subject = "Authentication";
        SmtpClient smtp = new SmtpClient("smtp.gmail.com");
        smtp.EnableSsl = true;
        smtp.Port = 587;
        smtp.DeliveryMethod = SmtpDeliveryMethod.Network;
        smtp.Credentials = new NetworkCredential(from, pass);
        try
        {
            smtp.Send(message);
            MessageBox.Show("Please enter the code sent in your email");
            txtusername.Hide();
            txtpassword.Hide();
            label1.Hide();
            label2.Hide();
            lusername.Hide();
            lpassword.Hide();
            panelcaptcha.Hide();
            btnlogin.Hide();
            txtcode.Show();
            btnsubmit.Show();

        }
        catch(Exception ex)
        {
            MessageBox.Show("Something went wrong! Please make sure the username and password are correct");
        }
```

```
1 reference
private void btnsubmit_Click(object sender, EventArgs e)
{
    if (randomcode == (txtcode.Text).ToString())
    {
        submission();

    }
    else
    {
        MessageBox.Show("Code not a match");
    }
}
```

Once the user attempts to login with their username and password, OTP is sent to their email and the UI prompts the user to enter the code. If entered code matches the send code, user's attempt to login becomes successful.

Copy/paste Usages

Password is hidden when we type. So, while registering or resetting the password, user is not able to view the spelling mistake. Now if they copy and paste the same password into the Confirm password field without realizing that the password they had in mind was not the password they actually created for their account and end up being locked out of their account the next time they try to log in. To prevent this from happening I have disabled the feature of copy/paste in password and confirm password field.

```
1 reference
private void txtpassword_Click(object sender, EventArgs e)
{
    lpassword.Hide();
    txtpassword.ShortcutsEnabled = false;

}

1 reference
private void txtconfirmpass_Click(object sender, EventArgs e)
{
    lconfirmpassword.Hide();
    txtpassword.ShortcutsEnabled = false;

}
```

AES Encryption

Encryption protects sensitive data like password and enhances the communication security between client app and server. I have chosen the AES encryption to store password in database as it has longer keys than other making it highly secure. So, even if the database server is breached, without the key only cipher text is accessed.

Here I have used the secret key: `0ram@1234xxxxxxxxxxttttttuuuuuiiiiio`

Using this key, I have implemented encryption and decryption of password.

```
2 references
public static string Encrypt(string encryptString)
{
    string EncryptionKey = "0ram@1234xxxxxxxxxttttttuuuuuiiiiio";
    byte[] clearBytes = Encoding.Unicode.GetBytes(encryptString);
    using (Aes encryptor = Aes.Create())
    {
        Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[]
        {0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76});
        encryptor.Key = pdb.GetBytes(32);
        encryptor.IV = pdb.GetBytes(16);
        using (MemoryStream ms = new MemoryStream())
        {
            using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateEncryptor(), CryptoStreamMode.Write))
            {
                cs.Write(clearBytes, 0, clearBytes.Length);
                cs.Close();
            }
            encryptString = Convert.ToBase64String(ms.ToArray());
        }
    }
    return encryptString;
}
```

```
2 references
public static string Decrypt(string cipherText)
{
    string EncryptionKey = "0ram@1234xxxxxxxxxttttttuuuuuiiiiio";
    cipherText = cipherText.Replace(" ", "+");
    byte[] cipherBytes = Convert.FromBase64String(cipherText);
    using (Aes encryptor = Aes.Create())
    {
        Rfc2898DeriveBytes pdb = new Rfc2898DeriveBytes(EncryptionKey, new byte[]
        { 0x49, 0x76, 0x61, 0x6e, 0x20, 0x4d, 0x65, 0x64, 0x76, 0x65, 0x64, 0x65, 0x76 });
        encryptor.Key = pdb.GetBytes(32);
        encryptor.IV = pdb.GetBytes(16);
        using (MemoryStream ms = new MemoryStream())
        {
            using (CryptoStream cs = new CryptoStream(ms, encryptor.CreateDecryptor(), CryptoStreamMode.Write))
            {
                cs.Write(cipherBytes, 0, cipherBytes.Length);
                cs.Close();
            }
            cipherText = Encoding.Unicode.GetString(ms.ToArray());
        }
    }
    return cipherText;
}
```

Only the cipher text is stored in the database not the password itself.

```
private void submission()
{
    if (txtmeter.BackColor != Color.Red && lxemail.Visible == false && lxusername.Visible == false && lxpassword.Visible == false && l
    {
        try
        {
            string password = cryptography.Encrypt(txtpassword.Text.ToString());
            con.Open();
            panelwhy.Hide();
            SqlCommand cmd1 = new SqlCommand("Insert into users(email, username, password) values( '" + txtemail.Text + "'," +
           " '" + txtusername.Text + "', '" + password + "')", con);
            cmd1.ExecuteNonQuery();
            MessageBox.Show("Account Registered! Now login with your details to continue.", "Congratulation!", MessageBoxButtons.OK, M
            con.Close();
            this.Hide();
```

When logging in the cipher text is decrypted and compared with the password.

```
private void submission()
{
    string Password = "";
    bool IsExist = false;
    con.Open();
    SqlCommand cmd = new SqlCommand("select * from users where username='" + txtusername.Text + "'", con);
    SqlDataReader sdr = cmd.ExecuteReader();
    if (sdr.Read())
    {
        Password = sdr.GetString(2);
        IsExist = true;
    }
    if (IsExist)
    {
        if (cryptography.Decrypt(Password).Equals(txtpassword.Text))
        {
            MessageBox.Show("Login Success", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Hide();
             Welcome welcome = new Welcome();
             welcome.Show();
        }
```

Reset Password

Forgot password feature allows users who have forgotten their password to reset it by sending them an e-mail with an OTP. The working phenomenon of OTP is same as that of the MFA. Just the OTP verification is used to identity the user and change the password.

```
private void btnreset_Click(object sender, EventArgs e)
{


        if (txtmeter.BackColor != Color.Red && lxpassword.Visible == false && lxconfirm.Visible == false)
        {
            try
            {
                string password = cryptography.Encrypt(txtpassword.Text.ToString());
                con.Open();
                panelwhy.Hide();
                SqlCommand cmd1 = new SqlCommand("update users set password='" + password + "' where email ='" + txtemail.Text + "'", con);

                cmd1.ExecuteNonQuery();
                MessageBox.Show("Password reset sucessfull.", "Congratulation!", MessageBoxButtons.OK, MessageBoxIcon.Information);
                con.Close();
                this.Hide();
                Login login = new Login();
                login.Show();

            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
        else
        {
            MessageBox.Show("Please check if all detail are vaild!");
        }
```

Here, if the password satisfies the requirement and strength features, the new entered password encrypted and stored in the database.

Password Repetition Prevention

After resetting the password, user is not allowed to use their old password because it increases the chance to determine the password through brute force attacks and the effectiveness of a good password policy is greatly reduced.

```
string Password = "";
bool IsExist = false;
con.Open();
SqlCommand cmd = new SqlCommand("select * from users where email='" + txtemail.Text + "'", con);
SqlDataReader sdr = cmd.ExecuteReader();
if (sdr.Read())
{
    Password = sdr.GetString(2);
    IsExist = true;
}
if (IsExist)
{
    if (cryptography.Decrypt(Password).Equals(txtpassword.Text))
    {
        MessageBox.Show("You cannot use old password again");
        txtpassword.Clear(); txtconfirmpass.Clear();
        txtmeter.BackColor = Color.Red;
    }
```

Here it checks if the new password is same as old password and shows error message if it's a match.

Captcha Feature

Captcha feature is implemented in the prototype to restrict bots from faking to be users. Bots lack the sophistication that human minds have of picking up on patterns in everything they see; pareidolia. While some can be programmed to recognize letters and numbers. But when they are obscured or distorted too much, they cannot recognize it.

There are various types of captchas; text, audio, image based. (Shivani Deosatwar, 2020)

```
private void loadCaptchaImage()
{

    Random r1 = new Random();
    number = r1.Next(100, 10000);
    var image = new Bitmap(this.pictureBox1.Width, this.pictureBox1.Height);
    var font = new Font("Fancy", 26, FontStyle.Bold | FontStyle.Strikeout|FontStyle.Italic, GraphicsUnit.Pixel);
    var graphics = Graphics.FromImage(image);
    graphics.DrawString(number.ToString(), font, Brushes.Yellow, new Point(10, 0));
    pictureBox1.Image = image;

}
```

Here I have designed and enforced a text-based number captcha where the number has been Strikeout. Making it difficult for bots to recognize but not the humans.

References

Anon., 2001. ADVANCED ENCRYPTION STANDARD (AES). In: s.l.:Processing Standards Publication 197.

Fäl, M., 2020. Multi-factor Authentication. p. 73.

Richard Shay, S. K. L. D. P. (. H. M. L. M. S. M. S. B., 2016. Designing Password Policies for Strength and Usability. *ACM Transactions on Information and System Security*, Volume V, p. 30.

Saranga Komanduri, R. S. P. G. K. M. L. M. L. B. N. C. L. F. C. S. E., 2011. *Of Passwords and People: Measuring the Effect of Password-Composition Policies.* Vanacouver, s.n.

Shivani Deosatwar, S. D. V. D. R. S. L. K., 2020. An overview of Various Types of CAPTCHA.