**Object-Oriented Design II**

SSE 554

Dr. MacNeil

Spring 2017

**Project III**

Brent Bitler

Matthew Robison

Table of Contents

[Table of Contents 2](#_Toc477729218)

[Table of Figures 3](#_Toc477729219)

[Project Logistics 4](#_Toc477729220)

[Purpose 4](#_Toc477729221)

[Authentication 5](#_Toc477729222)

[Overview 5](#_Toc477729223)

[Unit Testing 12](#_Toc477729224)

[Encryption 13](#_Toc477729225)

[Overview 13](#_Toc477729226)

[Implementation 13](#_Toc477729227)

[Unit Testing 19](#_Toc477729228)

[File Access 20](#_Toc477729229)

[Overview 20](#_Toc477729230)

[Unit Testing 21](#_Toc477729231)

[Appendix 22](#_Toc477729232)

[Auth.cs [Authentication] 22](#_Toc477729233)

[ProgramTest.cs [Authentication] 24](#_Toc477729234)

[Program.cs [Encryption] 25](#_Toc477729235)

[CryptorEngine.cs [Encryption] 26](#_Toc477729236)

[frmMain.cs [Encryption] 28](#_Toc477729237)

[frmMain.Designer.cs [Encryption] 30](#_Toc477729238)

[frmMainTest.cs[Encryption] 34](#_Toc477729239)

[CryptorEngineTests.cs [Encryption] 35](#_Toc477729240)

[FileAccess.cs [File Access] 36](#_Toc477729241)

[FileAccessTest.cs [File Access] 37](#_Toc477729242)

[Bibliography 38](#_Toc477729243)

[Activity Log - Bitler 39](#_Toc477729244)

[Activity Log - Robison 41](#_Toc477729245)

Table of Figures

Figure 1. Main method, getIDinfo invoked 6

Figure 2. Invokation of GetCurrent withing getIDinfo method 6

Figure 3. ID check 6

Figure 4. Identity information written to console 7

Figure 5. Identity console output 7

Figure 6. getPRinfo method invocation 8

Figure 7. Principal conditional check 8

Figure 8. User/Admin role authentication 9

Figure 9. User/Admin role output 9

Figure 10. Claims method 9

Figure 11. Claims console output 10

Figure 12. Foreach loop to print the property key and value 10

Figure 13. Claim section output 11

Figure 14. Authorization unit test 12

Figure 15. EnCryptDecrypt entry point 14

Figure 16. EnCryptDecrypt Form 14

Figure 17. btnEncrypt\_Click method 15

Figure 18. Form error/input validation 15

Figure 19. Encrypt method 16

Figure 20. Output after user input and press of encrypt button 17

Figure 21. btnDecrypt\_Click method 17

Figure 22. Decrypt method 18

Figure 23. Output after decrypt button pressed. 19

Figure 24. Unit test of EnCryptDecrypt methods 19

Figure 25. FileAccess main 20

Figure 26. FileAccess console output 21

Figure 27. FileAccess unit test. 21

Project Logistics

## Purpose

For Project III, the purpose was to choose a topic new to us from the text, demonstrate our capabilities through software design, coding, and finally unit testing while using some form of source control. For Project III, we have decided to learn about Security from the text. The features of the project requirements are outlined in the table below.

Table 1. Project II Composition

|  |  |
| --- | --- |
| Team Composition | Brent Bitler, Matthew Robison |
| Topic | Security |
| Distributed Version Control System | GitHub |
| Programming Language | C# |
| IDE | Visual Studio 2015 |
| TDD Tool | Built-in VS unit test tool |

Authentication

## Overview

The first foray into program security is to find out information about the user to see if they are first authorized to utilize an application. Several questions can be asked to begin investigating this very issue: Is the user who they claim to be? Is the user being impersonated by someone with malicious intent? How do we decide whether to trust and accept the user’s request to access the program or deny them? This is the first in a two-part process of securing a program known as Authentication: the process of identifying the user [1].

C# has a few integrated classes a developer is able to utilize to derive information about a user. This has been increasingly useful since Windows Live accounts are becoming almost required with later versions of Windows. Some of the basic information provided at this level is the user’s name, it can determine if an authentication exists, and if so, what type of authentication the user contains. The WindowsIdentity class contains several constructors, properties, and methods one can utilize as a developer to extra meaningful data about the user’s Windows account for security purposes.

We decided to implement a simple console application to illustrate such basic but important behavior before getting into some of the more robust topics such as encryption and data access control and more further along in the report. This application delves down into the underpinnings of the user’s Windows account. To do so, two namespaces were imported:

* System.Security.Claims
* System.Security.Principal

As seen in figure 1, the program’s Main method invokes the getIDinfo to start.

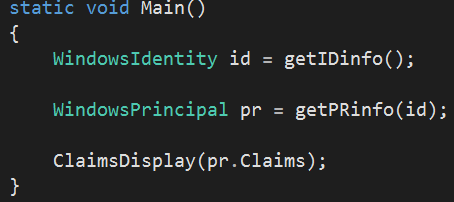


Figure 1. Main method, getIDinfo invoked

From here, a WindowsIdentity object is created by utilizing the GetCurrent() built-in method from the WindowsIdentity class. This method call can be seen in figure 2. The GetCurrent() method returns a System.Security.Principal.WindowsIdentity object representing the current Windows user.

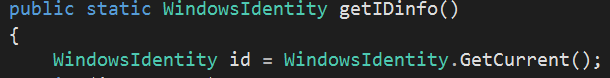


Figure 2. Invokation of GetCurrent withing getIDinfo method

Furthermore, a quick check if the GetCurrent method is unable to return an id is run through the conditional shown in figure 3.

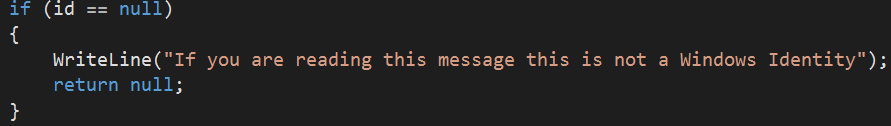


Figure 3. ID check

Utilizing several of the built-in methods as seen in figure 4, the program is able to write to the console information such as the identity’s:

* Type
* Name
* Guest verification
* Authentication verification
* Authentication type
* Anonymous verification
* Impersonation level
* Access token

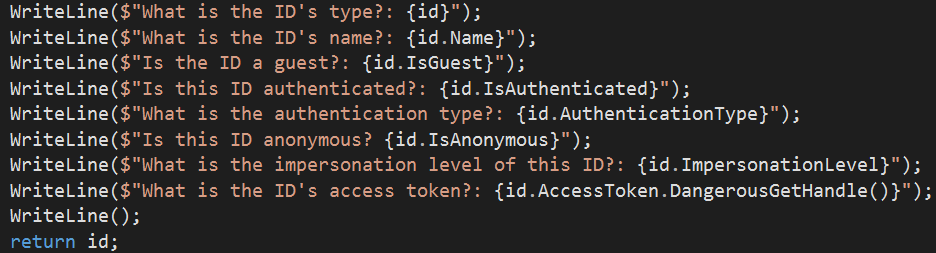


Figure 4. Identity information written to console

The output from this section of code execution yields the console output shown in figure 5.

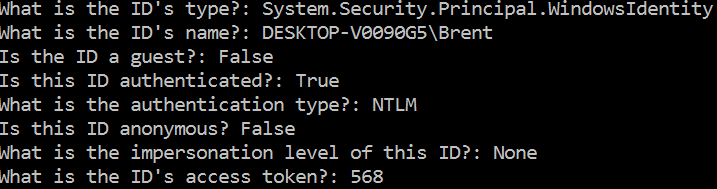


Figure 5. Identity console output

The ID type is as expected with name being the logon to the Windows machine in use. This account is the sole user on the bootcamped version of the MacBook in use and therefore is not a guest account. Furthermore, the account has been authenticated with type NT Lan Manager (NTLM), which is a challenge-response authentication protocol whereupon three messages are used to authenticate a client. The client sets up a network path with the server, first sending a NEGOTIATE\_MESSAGE qualifying its abilities. The server will follow up with a CHALLENGE\_MESSAGE used to demonstrate the client’s identity. Last, the client will then respond with an AUTHENTICATE\_MESSAGE [2]. More information about this authentication type can be gleaned from the Wikipedia article in the bibliography.

The identity is not anonymous and no impersonation level is set because the account is not impersonating. Finally, the access token for the session is 568. An access token enables applications to acquire user information. For more robust applications, this token can be stored in a server session and can be refreshed in a database. This sort of process will ensure the application can access the user’s information until an allotted time period expires. The next step in the process is to look at the invocation of the getPRinfo() method in Main. This is the second of three method calls in Main as seen in figure 6.

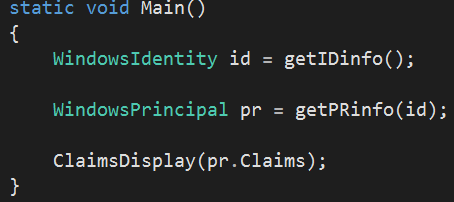


Figure 6. getPRinfo method invocation

The getPRinfo method takes the identity object returned by the getIDinfo method to retrieve the principal information requested by the console application. Moreover, a principal holds an identity and even more information about the user; more specifically, the roles associated with the user. Like within the getIDinfo method, the beginning of the getPRinfo method utilizes a conditional check to see if principal is null or not as seen in figure 7 before attemption to delve deeper and retrieve information about it.

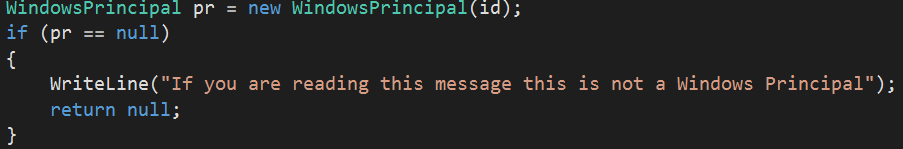


Figure 7. Principal conditional check

This section is crucial to find out more information about the user before providing them access to certain restricted resources or files. The getPRinfo method checks the user’s account rule by utilizing the IsInRole() method to determine if the account has a role as a user and/or an administrator. This code snippet can be seen in figure 8 with the IsInRole() method being overloaded with an enumeration value of the WindowsBuiltInRole enumeration. Both methods return a Boolean to validate the check at hand. The output of this code snippet is shown in figure 9, with the account being involved in the users group, but is not designated as an administrator because that setting was not chosen upon account creation.

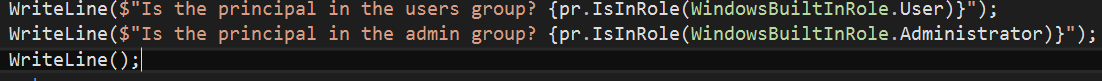


Figure 8. User/Admin role authentication

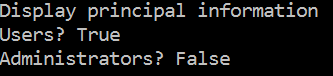


Figure 9. User/Admin role output

The last 33% of this application deals with claims, or more specifically statements about a given identity from an authority. This can be seen as a deeper layer of verification from roles. The authority can be the Active Directory (AD) or the Microsoft Live account authentication service are responsible for making claims about the user such as their name, groups they exist in, etc. [1]. The ClaimsDisplay method utilizes a collection of claims to produce the subject, issuer, type, or value as seen in figure 10 with resulting output in figure 11.

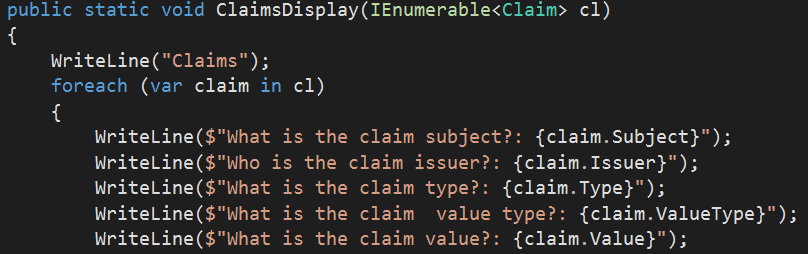


Figure 10. Claims method

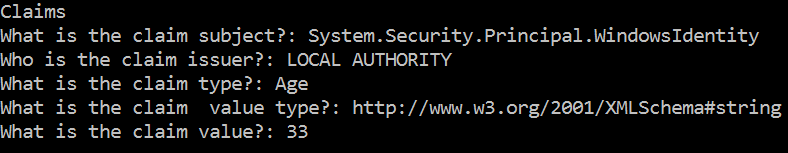


Figure 11. Claims console output

This is a simple output for a single claim added in the getIDinfo method. Furthermore, a simple for loop (figure 12) produces laundry list of outputs is shown in figure 13 in similar fashion.

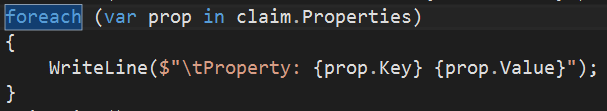


Figure 12. Foreach loop to print the property key and value

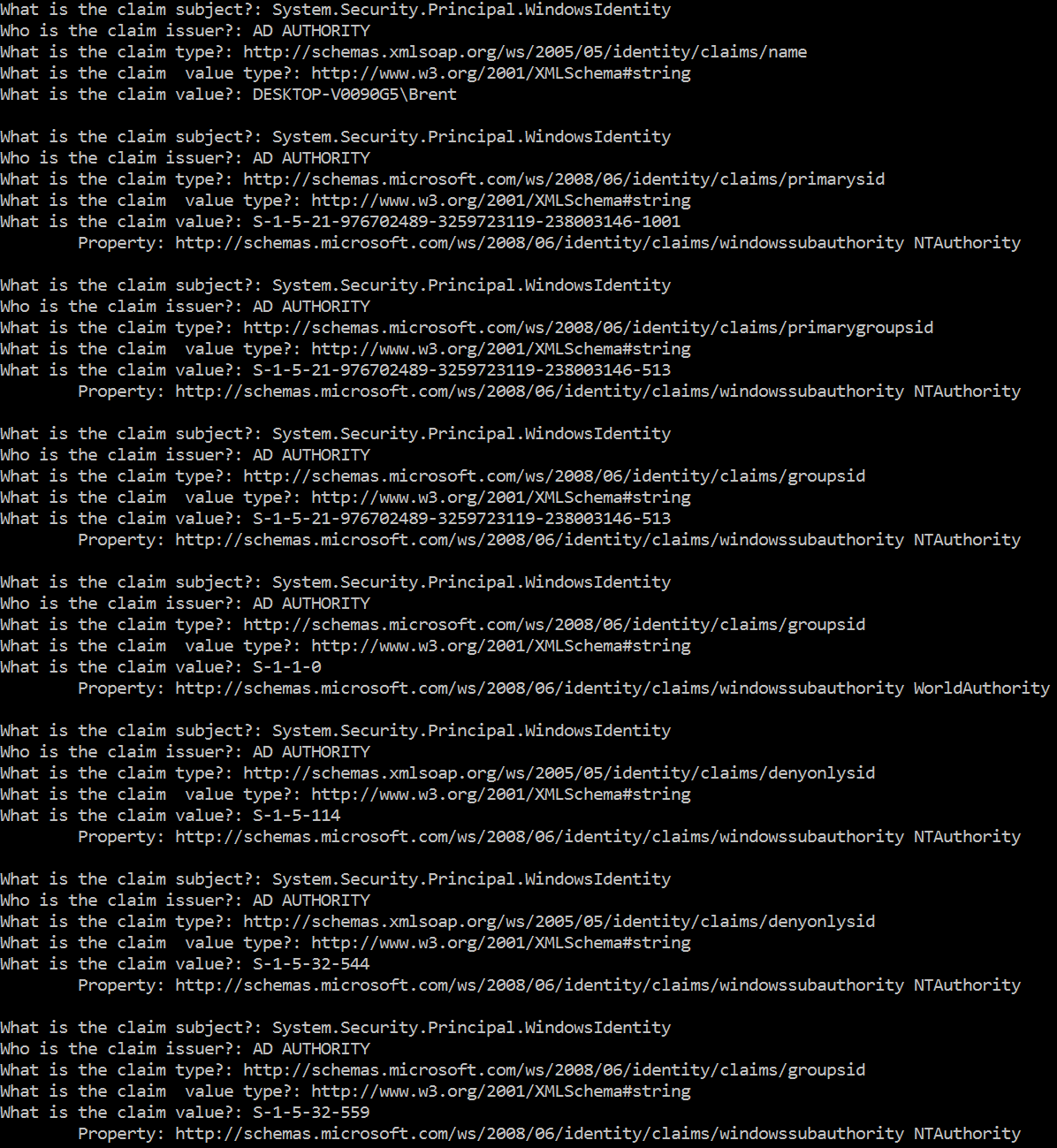


Figure 13. Claim section output

## Unit Testing

We followed test driven development process in the implementation of this program. Figure 14 illustrates successful unit test execution for our methods. The unit test code can be found in the appendix.



Figure 14. Authorization unit test

Encryption

## Overview

The next stint into program security is encryption. Encryption is the process of encoding data in such a manner as only people that are authorized are able to access it. Encryption itself doesn’t prevent interference of data, but denies any malicious interceptor of useful data. Some prime examples of where encryption is useful are online transactions, online banking, and e-filing tax returns. In the process of encryption, we take data and run it thru an encryption algorithm which returns to use a cyphertext that can only be read if decrypted. The encryption process uses keys to restrict decryption of data. There are two methods of using keys public key and private key. In the public key method for the encryption side the key is published so anyone can use it to encrypt data, but the recipient of the data has a private key so only they can decrypt the data. In the private key method both the encryption side and decryption side have the same private key.

C# provides classes via the .Net framework to developers for utilization in encryption of data. The classes that are provided are located in the System.Security.Cryptography namespace. Some notable classes in this namespace are SHA256 class this is the same hashing algorithm utilized by most banking organization in their encryption process. C# also provides a SHA512 class banks undoubtedly will be moving to this in the future to increase difficulty of stealing data. There are many other hashing classes that C# provides in its .Net framework. Another notable class in this name space is the TripleDESCryptoServiceProvider class. This class provides a wrapper object that we can use to access the cryptographic service provider version of the TripleDES algorithm. The TripleDES algorithm is were the process of encryption takes place.

## Implementation

We decided to implement a Graphical User Interface(GUI) application to illustrate the process of encryption. The program takes and string input from a user. Once the user clicks the encrypt button the program encrypts the string, and returns the encrypted version of the string back to the user via a text box. The program also has a decryption button that takes the encrypted string and decrypts it returns the value to a text box. Below in figure 15 we can see the entry point for our GUI application. In this section of the code we are setting up the visual style of the form and evoking run on the form.

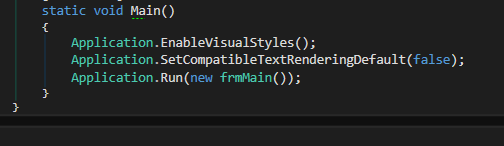


Figure 15. EnCryptDecrypt entry point

After the Application.Run(new frmMain()); line of text is executed the form is displayed to the screen. Below in figure 16 we can see the form.

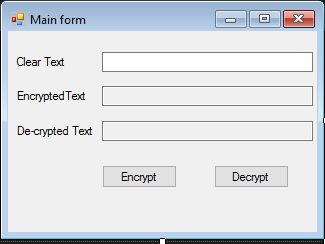


Figure 16. EnCryptDecrypt Form

Once this form is on the screen the user can type text into the clear text entry field. After the user, has entered their text the Encrypt button is pressed. Below in figure 17 we can see the code that is executed on button press of the encrypt button.

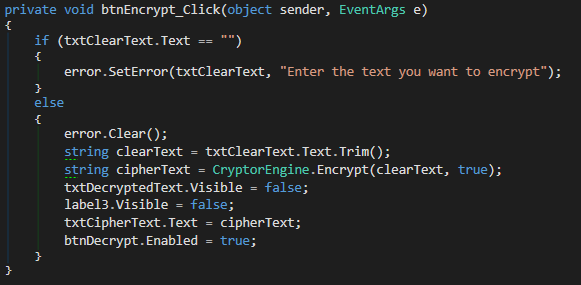


Figure 17. btnEncrypt\_Click method

Inside of the btnEncrypt\_Click method we first use an if else structure to validate the input from the user. If the text box is empty, we will set an error. Below in figure 18 we can see what this error looks like in the GUI.

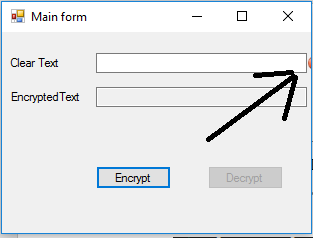


Figure 18. Form error/input validation

In the else block of the code we first clear the error just in case it had been thrown earlier. We then set a string equal to the input received by the user and declare a variable cipherText. cipherText is set to the value returned by the CryptorEngine.Encrypt methods returning value. Below in figure 19 we can see the encrypt methods code. The first thing the encrypt method does declare and define the variables that will be needed by the TripleDESCryptoServiceProvider object later. Next, we check if the bool useHashing is set to true if it is we will be hashing the key with the md5 hashing algorithm and storing it inside of the key\_array variable. If the useHashing bool is false we will just store the original key into the key\_array variable. After this we need to build a TripleDESCryptoServiceProvider object to encrypt the data. The process of is implemented by creating the crypto\_transform variable and defining it with the output of the TripleDESCryptoServiceProvider CreateEncryptor method. After we have created the crypto\_transform variable we can use this objects method TransformFinalBlock to encrypt are data. Finally, we return our encrypted string.

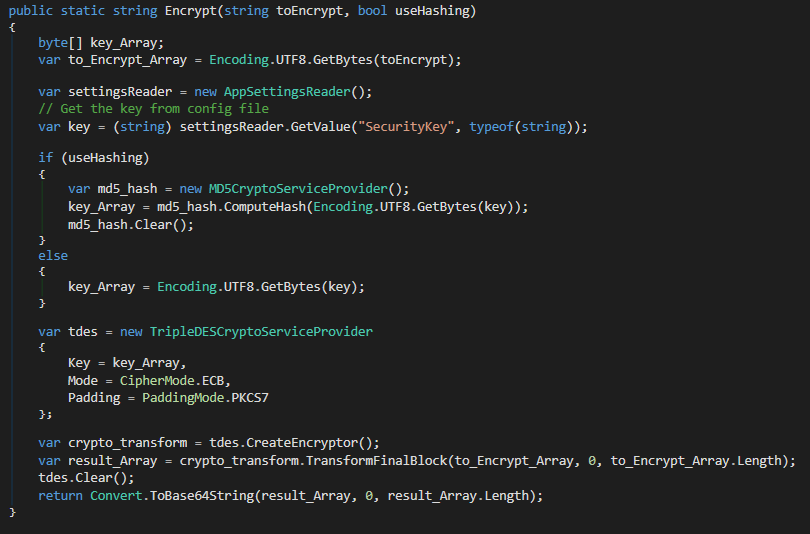


Figure 19. Encrypt method

Below in figure 20 we can see what the form looks like after user input has been given and the encrypt button has been pressed.

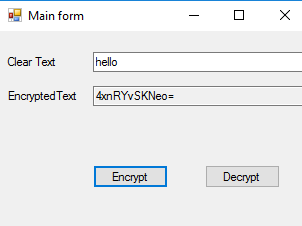


Figure 20. Output after user input and press of encrypt button

After the program, has reached this point the decrypt button because clickable. Below in figure 21 we can see the btnDecrypt\_Click method. In this method we store the cipherText variable from the Encrypted Text entry field and make a call to CryptorEngine.Decrypt method to set the value of decryptedText. Once this value is set we set the text of the txtDecryptedText text box to the value of decryptedText and set the visibility of the label and the text box to true.

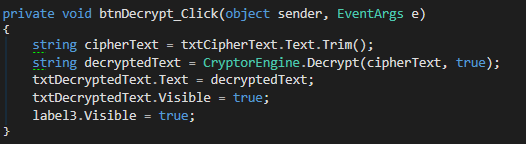


Figure 21. btnDecrypt\_Click method

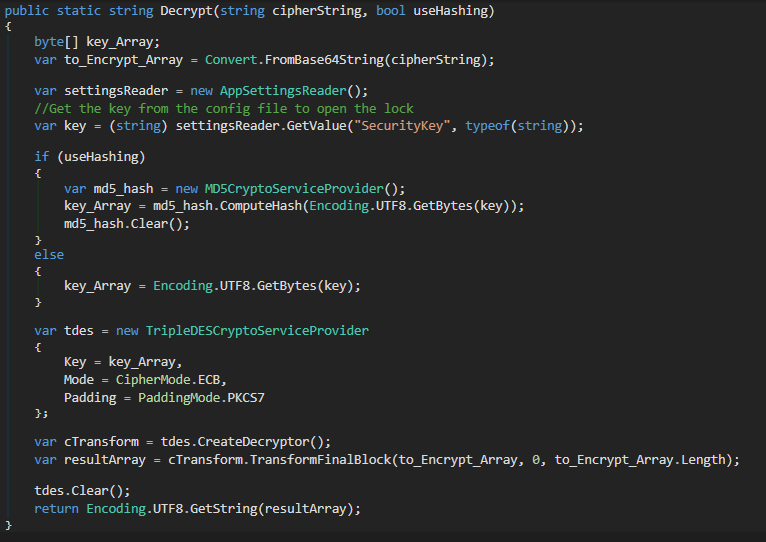
Below in figure 22 we can see the decrypt method. The decrypt method starts out by declaring and defining key\_array, to\_Encrypt\_Array, settingsReader and key. These variables will be needed later by the TripleDESCryptoServiceProvider object to create the decryptor. Next, we determine if the useHashing boolean is set to true or false. If useHashing is true, the key will be hashed before being placed into the key\_Array variable. If false, then no hashing will be done on the key. Then the decrypt method creates the TripleDESCryptoServiceProvider object tdes. This object is used to create a decryptor which we will use to decrypt the data. Finally, the method decrypts the data and returns the unencrypted data.

Figure 22. Decrypt method

Below in figure 23 we can see the output of the GUI application after the decrypt button has been pressed.

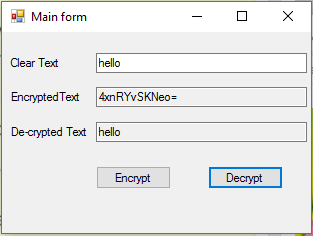


Figure 23. Output after decrypt button pressed.

## Unit Testing

We followed test driven development process in the implementation of this program. Figure 23 illustrates successful unit test execution for our methods. The unit test code can be found in the appendix.

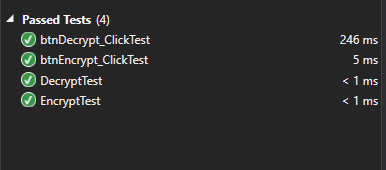


Figure 24. Unit test of EnCryptDecrypt methods

File Access

## Overview

The last third of our report encompasses access given a user’s identity. The System.Security.AccessControl namespace will be the new player to help us out. The FileStream predefined class contains the definition for the GetAccessControl method, where the return type is a FileSecurity object. The FileSecurity class contains the GetAccessRules() method which allows the user to dictate if inherited access rules and whether only access rules defined with the object can be utilized [1]. The third parameter passed is the security ID type, which in our case is NTAccount. This helps locate the security object based on its name. Due to the nature of the example, a file is accessed and therefore the AuthorizationRule is cast to a FileSystemAccessRule. The code discussed above is located in figure 25, where a FileAccessText.txt file is accessed by the FileStream and the ruleset is printed to the console, containing the user’s identity, the access type, and the rights.

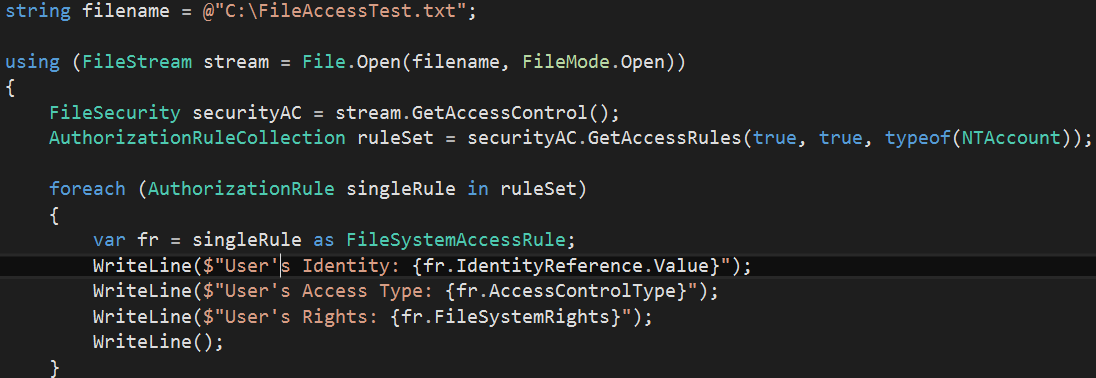


Figure 25. FileAccess main

The output is shown in figure 26, whereupon based on the type of identity, every user is allowed access to the text file. However, only the System and Admin have full control. A regular user simply has the capability to read and execute/sync the file, while an authenticated user can modify the file or synchronize. Furthermore, a breakdown of the enumeration of file access rights can be found at the following URL: <https://msdn.microsoft.com/en-us/library/system.security.accesscontrol.filesystemrights(v=vs.110).aspx>

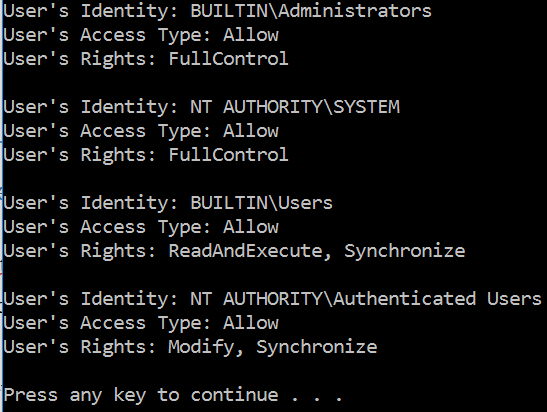


Figure 26. FileAccess console output

## Unit Testing

We followed test driven development process in the implementation of this program. Figure 27 illustrates successful unit test execution for our methods. The unit test code can be found in the appendix.



Figure 27. FileAccess unit test.

## 

Appendix

## Auth.cs [Authentication]

using static System.Console;

using System.Collections.Generic;

using System.Security.Claims;

using System.Security.Principal;

namespace WindowsPrincipalSample

{

public class Program

{

public static void Main()

{

WindowsIdentity id = getIDinfo();

WindowsPrincipal pr = getPRinfo(id);

ClaimsDisplay(pr.Claims);

}

public static WindowsIdentity getIDinfo()

{

WindowsIdentity id = WindowsIdentity.GetCurrent();

if (id == null)

{

WriteLine("If you are reading this message this is not a Windows Identity");

return null;

}

id.AddClaim(new Claim("Age", "33"));

WriteLine($"What is the ID's type?: {id}");

WriteLine($"What is the ID's name?: {id.Name}");

WriteLine($"Is the ID a guest?: {id.IsGuest}");

WriteLine($"Is this ID authenticated?: {id.IsAuthenticated}");

WriteLine($"What is the authentication type?: {id.AuthenticationType}");

WriteLine($"Is this ID anonymous? {id.IsAnonymous}");

WriteLine($"What is the impersonation level of this ID?: {id.ImpersonationLevel}");

WriteLine($"What is the ID's access token?: {id.AccessToken.DangerousGetHandle()}");

WriteLine();

return id;

}

public static WindowsPrincipal getPRinfo(WindowsIdentity id)

{

WriteLine("Display principal information");

WindowsPrincipal pr = new WindowsPrincipal(id);

if (pr == null)

{

WriteLine("If you are reading this message this is not a Windows Principal");

return null;

}

WriteLine($"Is the principal in the users group? {pr.IsInRole(WindowsBuiltInRole.User)}");

WriteLine($"Is the principal in the admin group? {pr.IsInRole(WindowsBuiltInRole.Administrator)}");

WriteLine();

return pr;

}

public static void ClaimsDisplay(IEnumerable<Claim> cl)

{

WriteLine("Claims");

foreach (var claim in cl)

{

WriteLine($"What is the claim subject?: {claim.Subject}");

WriteLine($"Who is the claim issuer?: {claim.Issuer}");

WriteLine($"What is the claim type?: {claim.Type}");

WriteLine($"What is the claim value type?: {claim.ValueType}");

WriteLine($"What is the claim value?: {claim.Value}");

foreach (var prop in claim.Properties)

{

WriteLine($"\tProperty: {prop.Key} {prop.Value}");

}

WriteLine();

}

}

}

}

## ProgramTest.cs [Authentication]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using WindowsPrincipalSample;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.Principal;

using System.Text;

using System.Threading.Tasks;

namespace WindowsPrincipalSample.Tests

{

[TestClass()]

public class ProgramTests

{

[TestMethod()]

public void GetIDinfoTest()

{

WindowsIdentity id = Program.getIDinfo();

Assert.IsInstanceOfType(id, typeof(WindowsIdentity));

}

[TestMethod()]

public void GetPRinfoTest()

{

WindowsIdentity id = Program.getIDinfo();

WindowsPrincipal pr = Program.getPRinfo(id);

Assert.IsInstanceOfType(pr, typeof(WindowsIdentity));

}

[TestMethod()]

public void ClaimsDisplayTest()

{

WindowsIdentity id = Program.getIDinfo();

WindowsPrincipal pr = Program.getPRinfo(id);

Program.ClaimsDisplay(pr.Claims);

Assert.Fail();

}

}

}

## Program.cs [Encryption]

using System;

using System.Collections.Generic;

using System.Windows.Forms;

namespace EnCryptDecrypt

{

static class Program

{

/// The main entry point for the application.

[STAThread]

public static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new frmMain());

}

}

}

## CryptorEngine.cs [Encryption]

using System;

using System.Configuration;

using System.Diagnostics.CodeAnalysis;

using System.Security.Cryptography;

using System.Text;

namespace EnCryptDecrypt

{

[SuppressMessage("ReSharper", "InconsistentNaming")]

public class CryptorEngine

{

public static string Encrypt(string toEncrypt, bool useHashing)

{

byte[] key\_Array;

var to\_Encrypt\_Array = Encoding.UTF8.GetBytes(toEncrypt);

var settingsReader = new AppSettingsReader();

// Get the key from config file

var key = (string) settingsReader.GetValue("SecurityKey", typeof(string));

if (useHashing)

{

var md5\_hash = new MD5CryptoServiceProvider();

key\_Array = md5\_hash.ComputeHash(Encoding.UTF8.GetBytes(key));

md5\_hash.Clear();

}

else

{

key\_Array = Encoding.UTF8.GetBytes(key);

}

var tdes = new TripleDESCryptoServiceProvider

{

Key = key\_Array,

Mode = CipherMode.ECB,

Padding = PaddingMode.PKCS7

};

var crypto\_transform = tdes.CreateEncryptor();

var result\_Array = crypto\_transform.TransformFinalBlock(to\_Encrypt\_Array, 0, to\_Encrypt\_Array.Length);

tdes.Clear();

return Convert.ToBase64String(result\_Array, 0, result\_Array.Length);

}

public static string Decrypt(string cipherString, bool useHashing)

{

byte[] key\_Array;

var to\_Encrypt\_Array = Convert.FromBase64String(cipherString);

var settingsReader = new AppSettingsReader();

//Get the key from the config file to open the lock

var key = (string) settingsReader.GetValue("SecurityKey", typeof(string));

if (useHashing)

{

var md5\_hash = new MD5CryptoServiceProvider();

key\_Array = md5\_hash.ComputeHash(Encoding.UTF8.GetBytes(key));

md5\_hash.Clear();

}

else

{

key\_Array = Encoding.UTF8.GetBytes(key);

}

var tdes = new TripleDESCryptoServiceProvider

{

Key = key\_Array,

Mode = CipherMode.ECB,

Padding = PaddingMode.PKCS7

};

var cTransform = tdes.CreateDecryptor();

var resultArray = cTransform.TransformFinalBlock(to\_Encrypt\_Array, 0, to\_Encrypt\_Array.Length);

tdes.Clear();

return Encoding.UTF8.GetString(resultArray);

}

}

}

|  |
| --- |
| frmMain.cs [Encryption] using System;  using System.Collections.Generic;  using System.ComponentModel;  using System.Data;  using System.Drawing;  using System.Text;  using System.Windows.Forms;  namespace EnCryptDecrypt  {  public partial class frmMain : Form  {  public frmMain()  {  InitializeComponent();  }  public void btnEncrypt\_Click(object sender, EventArgs e)  {  if (txtClearText.Text == "")  {  error.SetError(txtClearText, "Enter the text you want to encrypt");  }  else  {  error.Clear();  string clearText = txtClearText.Text.Trim();  string cipherText = CryptorEngine.Encrypt(clearText, true);  txtDecryptedText.Visible = false;  label3.Visible = false;  txtCipherText.Text = cipherText;  btnDecrypt.Enabled = true;  }  }  public void btnDecrypt\_Click(object sender, EventArgs e)  {  string cipherText = txtCipherText.Text.Trim();  string decryptedText = CryptorEngine.Decrypt(cipherText, true);  txtDecryptedText.Text = decryptedText;  txtDecryptedText.Visible = true;  label3.Visible = true;  }  }  } |

## frmMain.Designer.cs [Encryption]

namespace EnCryptDecrypt

{

partial class frmMain

{

private System.ComponentModel.IContainer components = null;

protected override void Dispose(bool disposing)

{

if (disposing && (components != null))

{

components.Dispose();

}

base.Dispose(disposing);

}

/// <summary>

/// Required method for Designer support - do not modify

/// the contents of this method with the code editor.

/// </summary>

private void InitializeComponent()

{

this.components = new System.ComponentModel.Container();

this.btnEncrypt = new System.Windows.Forms.Button();

this.btnDecrypt = new System.Windows.Forms.Button();

this.txtClearText = new System.Windows.Forms.TextBox();

this.txtCipherText = new System.Windows.Forms.TextBox();

this.label1 = new System.Windows.Forms.Label();

this.label2 = new System.Windows.Forms.Label();

this.label3 = new System.Windows.Forms.Label();

this.txtDecryptedText = new System.Windows.Forms.TextBox();

this.error = new System.Windows.Forms.ErrorProvider(this.components);

((System.ComponentModel.ISupportInitialize)(this.error)).BeginInit();

this.SuspendLayout();

//

// btnEncrypt

//

this.btnEncrypt.Location = new System.Drawing.Point(94, 134);

this.btnEncrypt.Name = "btnEncrypt";

this.btnEncrypt.Size = new System.Drawing.Size(75, 23);

this.btnEncrypt.TabIndex = 0;

this.btnEncrypt.Text = "Encrypt";

this.btnEncrypt.UseVisualStyleBackColor = true;

this.btnEncrypt.Click += new System.EventHandler(this.btnEncrypt\_Click);

//

// btnDecrypt

//

this.btnDecrypt.Enabled = false;

this.btnDecrypt.Location = new System.Drawing.Point(206, 134);

this.btnDecrypt.Name = "btnDecrypt";

this.btnDecrypt.Size = new System.Drawing.Size(75, 23);

this.btnDecrypt.TabIndex = 1;

this.btnDecrypt.Text = "Decrypt";

this.btnDecrypt.UseVisualStyleBackColor = true;

this.btnDecrypt.Click += new System.EventHandler(this.btnDecrypt\_Click);

//

// txtClearText

//

this.txtClearText.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Left)

| System.Windows.Forms.AnchorStyles.Right)));

this.txtClearText.Location = new System.Drawing.Point(94, 21);

this.txtClearText.Name = "txtClearText";

this.txtClearText.Size = new System.Drawing.Size(211, 20);

this.txtClearText.TabIndex = 2;

//

// txtCipherText

//

this.txtCipherText.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Left)

| System.Windows.Forms.AnchorStyles.Right)));

this.txtCipherText.Location = new System.Drawing.Point(94, 55);

this.txtCipherText.Name = "txtCipherText";

this.txtCipherText.ReadOnly = true;

this.txtCipherText.Size = new System.Drawing.Size(211, 20);

this.txtCipherText.TabIndex = 3;

this.txtCipherText.TabStop = false;

//

// label1

//

this.label1.AutoSize = true;

this.label1.Location = new System.Drawing.Point(6, 24);

this.label1.Name = "label1";

this.label1.Size = new System.Drawing.Size(55, 13);

this.label1.TabIndex = 4;

this.label1.Text = "Clear Text";

//

// label2

//

this.label2.AutoSize = true;

this.label2.Location = new System.Drawing.Point(6, 58);

this.label2.Name = "label2";

this.label2.Size = new System.Drawing.Size(76, 13);

this.label2.TabIndex = 5;

this.label2.Text = "EncryptedText";

//

// label3

//

this.label3.AutoSize = true;

this.label3.Location = new System.Drawing.Point(6, 93);

this.label3.Name = "label3";

this.label3.Size = new System.Drawing.Size(83, 13);

this.label3.TabIndex = 6;

this.label3.Text = "De-crypted Text";

this.label3.Visible = false;

//

// txtDecryptedText

//

this.txtDecryptedText.Anchor = ((System.Windows.Forms.AnchorStyles)(((System.Windows.Forms.AnchorStyles.Top | System.Windows.Forms.AnchorStyles.Left)

| System.Windows.Forms.AnchorStyles.Right)));

this.txtDecryptedText.Location = new System.Drawing.Point(94, 90);

this.txtDecryptedText.Name = "txtDecryptedText";

this.txtDecryptedText.ReadOnly = true;

this.txtDecryptedText.Size = new System.Drawing.Size(211, 20);

this.txtDecryptedText.TabIndex = 7;

this.txtDecryptedText.TabStop = false;

this.txtDecryptedText.Visible = false;

//

// error

//

this.error.ContainerControl = this;

//

// frmMain

//

this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);

this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;

this.ClientSize = new System.Drawing.Size(309, 201);

this.Controls.Add(this.txtDecryptedText);

this.Controls.Add(this.label3);

this.Controls.Add(this.label2);

this.Controls.Add(this.label1);

this.Controls.Add(this.txtCipherText);

this.Controls.Add(this.txtClearText);

this.Controls.Add(this.btnDecrypt);

this.Controls.Add(this.btnEncrypt);

this.Name = "frmMain";

this.Text = "Main form";

((System.ComponentModel.ISupportInitialize)(this.error)).EndInit();

this.ResumeLayout(false);

this.PerformLayout();

}

private System.Windows.Forms.Button btnEncrypt;

private System.Windows.Forms.Button btnDecrypt;

private System.Windows.Forms.TextBox txtClearText;

private System.Windows.Forms.TextBox txtCipherText;

private System.Windows.Forms.Label label1;

private System.Windows.Forms.Label label2;

private System.Windows.Forms.Label label3;

private System.Windows.Forms.TextBox txtDecryptedText;

private System.Windows.Forms.ErrorProvider error;

}

}

## frmMainTest.cs[Encryption]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using EnCryptDecrypt;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EnCryptDecrypt.Tests

{

[TestClass()]

public class frmMainTests

{

[TestMethod()]

public void btnEncrypt\_ClickTest()

{

frmMain frm = new frmMain();

object one = new object();

EventArgs two = new EventArgs();

frm.btnEncrypt\_Click(one,two);

Assert.Fail();

}

[TestMethod()]

public void btnDecrypt\_ClickTest()

{

frmMain frm = new frmMain();

object one = new object();

EventArgs two = new EventArgs();

frm.btnDecrypt\_Click(one, two);

Assert.Fail();

}

}

}

## CryptorEngineTests.cs [Encryption]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using EnCryptDecrypt;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace EnCryptDecrypt.Tests

{

[TestClass()]

public class CryptorEngineTests

{

[TestMethod()]

public void EncryptTest()

{

string input = "hello";

String output = CryptorEngine.Encrypt(input, true);

Assert.AreEqual(output, "4xnRYvSKNeo=");

}

[TestMethod()]

public void DecryptTest()

{

string input = "4xnRYvSKNeo=";

String output = CryptorEngine.Decrypt(input, true);

Assert.AreEqual(output, "hello");

}

}

}

## FileAccess.cs [File Access]

using static System.Console;

using System.IO;

using System.Security.AccessControl;

using System.Security.Principal;

namespace AccessRights

{

public class Program

{

static void Main(string[] args)

{

string filename = @"C:\FileAccessTest.txt";

AuthorizationRuleCollection ruleset = CreateRuleSet(filename);

PrintRules(ruleset);

}

public static AuthorizationRuleCollection CreateRuleSet(string filename)

{

using (FileStream stream = File.Open(filename, FileMode.Open))

{

FileSecurity securityAC = stream.GetAccessControl();

AuthorizationRuleCollection ruleSet = securityAC.GetAccessRules(true, true, typeof(NTAccount));

return ruleSet;

}

}

public static void PrintRules(AuthorizationRuleCollection ruleSet)

{

foreach (AuthorizationRule singleRule in ruleSet)

{

var fr = singleRule as FileSystemAccessRule;

WriteLine($"User's Identity: {fr.IdentityReference.Value}");

WriteLine($"User's Access Type: {fr.AccessControlType}");

WriteLine($"User's Rights: {fr.FileSystemRights}");

WriteLine();

}

}

}

}

## FileAccessTest.cs [File Access]

using Microsoft.VisualStudio.TestTools.UnitTesting;

using AccessRights;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Security.AccessControl;

using System.Text;

namespace AccessRights.Tests

{

[TestClass()]

public class ProgramTests

{

[TestMethod()]

public void CreateRuleSetTest()

{

string filename = @"C:\FileAccessTest.txt";

AuthorizationRuleCollection ruleset = Program.CreateRuleSet(filename);

Assert.IsInstanceOfType(ruleset, typeof(AuthorizationRuleCollection));

}

[TestMethod()]

public void PrintRulesTest()

{

string filename = @"C:\FileAccessTest.txt";

AuthorizationRuleCollection ruleset = Program.CreateRuleSet(filename);

Program.PrintRules(ruleset);

Assert.Fail();

}

}

}

Bibliography

[1] Nagel, Christian. *Professional C# 6 and .NET Core 1.0, (1st ed.)*. WROX, 2016.

[2] "NT LAN Manager". *En.wikipedia.org*. N.p., 2017. Web. 19 Mar. 2017.

Activity Log - Bitler

|  |  |  |
| --- | --- | --- |
| Date | Time (mins) | Description |
| 1/9/17 | 150 | Looked through text for topics |
| 1/10/17 | 60 | Researched online for topics |
| 1/11/17 |  |  |
| 1/12/17 |  |  |
| 1/13/17 | 120 | Began investigating sockets in earnest |
| 1/14/17 | 90 | Class structure for chat program using sockets |
| 1/15/17 | 60 | Begin formatting paper outline |
| 1/16/17 | 60 | Begin server side code |
| 1/17/17 |  |  |
| 1/18/17 |  |  |
| 1/19/17 | 45 | Continue server side code |
| 1/20/17 | 60 | Start client side code |
| 1/21/17 |  |  |
| 1/22/17 | 240 | Through TDD, continue developing client code while cleaning up server side |
| 1/23/17 | 180 | Debug and continue working through client issues |
| 1/24/17 |  |  |
| 1/25/17 |  |  |
| 1/26/17 | 90 | Conclude coding, begin test case development |
| 1/27/17 | 60 | Screenshots/graphs of program |
| 1/28/17 | 60 | Fix bugs found in testing |
| 1/29/17 | 45 | Adding testing fixes to report |
| 1/30/17 | 120 | Client and server report sections |
| 2/1/17 |  |  |
| 2/2/17 | 120 | Conclude paper |
| 2/3/17 | 30 | Run final tests for verification |
| 2/4/17 |  |  |
| 2/5/17 | 120 | Final review of paper |
| 2/6/17 | 15 | Submit paper |
| 2/7/17 | 200 | Read MVC chapters |
| 2/8/17 | 60 | Begin drafting class structure for MVC apps |
| 2/9/17 |  |  |
| 2/10/17 |  |  |
| 2/11/17 | 240 | Work through controllers, selectors, views code |
| 2/12/17 |  |  |
| 2/13/17 | 300 | Working through Example app HTML |
| 2/14/17 |  |  |
| 2/15/17 | 180 | Database design |
| 2/16/17 |  |  |
| 2/17/17 | 100 | Unit Test design |
| 2/18/17 | 200 | Work on paper |
| 2/19/17 | 280 | Work on paper |
| 2/20/17 |  |  |
| 2/21/17 | 100 | Continue working paper |
| 2/22/17 |  |  |
| 2/23/17 |  |  |
| 2/24/17 | 100 | Add screenshots |
| 2/25/17 |  |  |
| 2/26/17 | 120 | Research topics for project 3 |
| 2/27/17 | 60 | Begin reviewing Security section of book for project ideas |
| 2/28/17 |  |  |
| 3/1/17 |  |  |
| 3/2/17 | 120 | Read/take notes on User Verification |
| 3/3/17 |  |  |
| 3/4/17 | 120 | Read/take notes on encryption |
| 3/5/17 |  |  |
| 3/6/17 | 240 | Read take notes on Certificates and file access |
| 3/7/17 |  |  |
| 3/8/17 |  |  |
| 3/9/17 | 200 | Work on user verification app |
| 3/10/17 | 180 | Conclude user verification app, begin encryption app |
| 3/11/17 |  |  |
| 3/12/17 | 240 | Conclude encryption app |
| 3/13/17 | 120 | Work and conclude file access app |
| 3/14/17 |  |  |
| 3/15/17 | 300 | Begin paper |
| 3/16/17 | 120 | Gather screenshots |
| 3/17/17 | 100 | Continue paper |
| 3/18/17 |  |  |
| 3/19/17 | 120 | Conclude and review paper |
| 3/20/17 |  |  |
|  |  |  |
| TOTAL 1 | 1725 |  |
| TOTAL 2 | 1700 |  |
| TOTAL 3 | 2040 |  |
| RUNNING TOTAL | 5465 |  |

Activity Log - Robison

|  |  |  |
| --- | --- | --- |
| Time/TaskLog | Activity Type: Indirect | |
| Date | Duration(mins) | Activities |
| Jan 10, 2017 | 30  120 | * Look through book for topics to use for project * Read Chapter 25 in text book. |
| Jan 11, 2017 | 90 | * Read more online about tcp/udp/http |
| Jan 13, 2017 | 120 | * Start Development of TCP Server |
| Jan 14, 2017 | 90 | * Start Development of TCP Client |
| Jan 16, 2017 | 120 | * Finish Development of TCP Server and Client |
| Jan 17, 2017 | 90 | * Research into how to use Http Request and Response with API |
| Jan 18, 2017 | 180 | * Start Development of API JSON Parser |
| Jan 22, 2017 | 180 | * Finish Development of API JSON Parser |
| Jan 24, 2017 | 90 | * Set up and experiment with GitHub |
| Jan 25, 2017 | 120 | * Start development of report |
| Feb 1, 2017 | 180 | * Continue development of report |
| Feb 3, 2017 | 120 | * Continue development of report |
| Feb 5, 2017 | 90 | * Continue development of report |
| Feb 6, 2017 | 90 | * Format Report and Upload to ftp |
| FEB 7, 2017 | 200 | * Read MVC text chapters |
| FEB 8, 2017 | 60 | * Class design for MVC app |
| FEB 9, 2017 |  |  |
| FEB 10, 2017 |  |  |
| FEB 11, 2017 |  |  |
| FEB 12, 2017 | 300 | * Review/update selectors, views, controllers |
| FEB 13, 2017 | 150 | * Work on HTML code for webpage |
| FEB 14, 2017 | 300 | * Continue HTML dev |
| FEB 15, 2017 | 240 | * Database implementation |
| FEB 16, 2017 |  |  |
| FEB 17, 2017 |  |  |
| FEB 18, 2017 | 180 | * Begin working paper |
| FEB 19, 2017 |  |  |
| FEB 20, 2017 |  |  |
| FEB 21, 2017 | 200 | * Continue paper |
| FEB 22, 2017 |  |  |
| FEB 23, 2017 | 60 | * Review paper before submission |
| FEB 24, 2017 |  |  |
| FEB 25, 2017 | 180 | * Research ideas for project 3 |
| FEB 26, 2017 | 100 | * Start reading security chapter of text |
| FEB 27, 2017 |  |  |
| FEB 28, 2017 |  |  |
| MAR 1, 2017 | 90 | * User Verification section |
| MAR 2, 2017 | 90 | * Encryption section |
| MAR 3, 2017 | 180 | * Certificates and file access sections |
| MAR 4, 2017 |  |  |
| MAR 5, 2017 |  |  |
| MAR 6, 2017 |  |  |
| MAR 7, 2017 | 100 | * Start user verification app |
| MAR 8, 2017 |  |  |
| MAR 9, 2017 | 150 | * Help finish user verification app |
| MAR 10, 2017 | 100 | * Start encryption app |
| MAR 11, 2017 | 180 | * Help finish encryption app |
| MAR 12, 2017 | 100 | * Help finish file access app |
| MAR 13, 2017 |  |  |
| MAR 14, 2017 | 200 | * Work on paper |
| MAR 15, 2017 | 180 | * Take screenshots, work on testing |
| MAR 16, 2017 |  |  |
| MAR 17, 2017 | 180 | * Continue paper |
| MAR 18, 2017 |  |  |
| MAR 19, 2017 | 150 | * Edit and finish paper |
| MAR 20, 2017 |  |  |
|  |  |  |
| Project I Total | 1710 |
| PROJECT 2 TOTAL | 1690 |
| PROJECT 3 TOTAL | 1980 |
| Running total | 5380 |