**Object-Oriented Design II**

SSE 554

Dr. MacNeil

Spring 2017

**Project III**

Brent Bitler

Matthew Robison

Table of Contents

[Table of Contents 2](#_Toc477417771)

[Table of Figures 3](#_Toc477417772)

[Introduction 4](#_Toc477417773)

[Purpose 4](#_Toc477417774)

[ASP.net MVC Breakdown 5](#_Toc477417775)

[Overview 5](#_Toc477417776)

[Appendix 6](#_Toc477417777)

[Program.cs 6](#_Toc477417778)

[Bibliography 7](#_Toc477417779)

[Activity Log - Bitler 8](#_Toc477417780)

[Activity Log - Robison 9](#_Toc477417781)

Table of Figures

No table of figures entries found.

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 . 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.

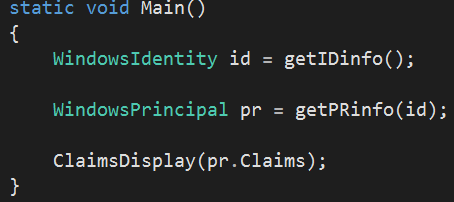


Figure . Main method, getIDinfo invoked

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.

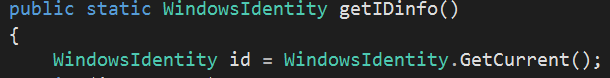


Figure . Invokation of GetCurrent withing getIDinfo method

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

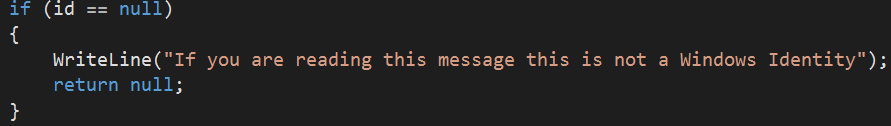


Figure . ID check

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

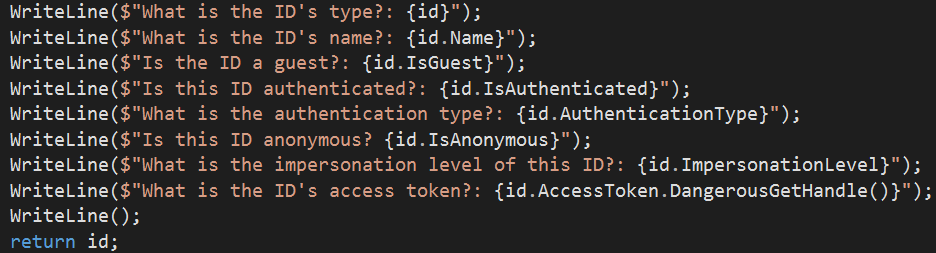


Figure . Identity information written to console

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

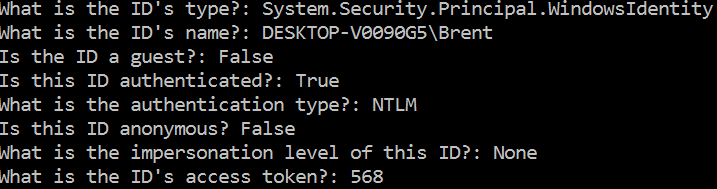


Figure . Identity console output

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.

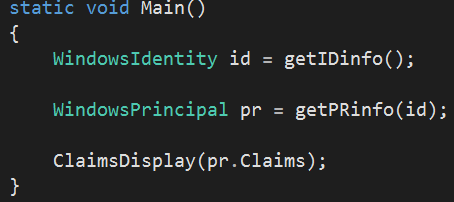


Figure . getPRinfo method invocation

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.

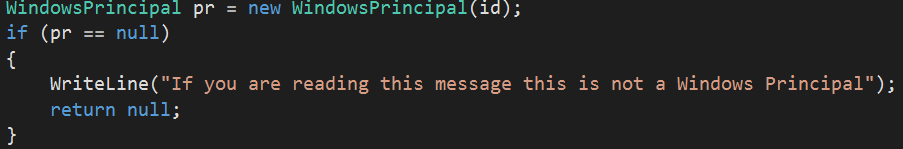


Figure . Principal conditional check

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.

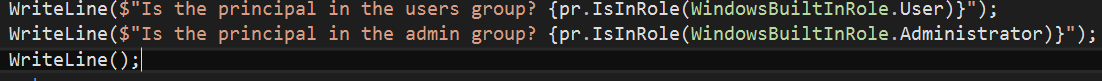


Figure . User/Admin role authentication

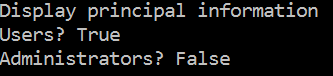


Figure . User/Admin role output

Encryption

## Overview

ASP.net

File Access Control

## Overview

ASP.net

Code Access Permissions

## Overview

ASP.net

Testing

Appendix

## Auth.cs

using static System.Console;

using System.Collections.Generic;

using System.Security.Claims;

using System.Security.Principal;

namespace WindowsPrincipalSample

{

class Program

{

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();

}

}

}

}

## Code2.cs

## Code3.cs

## Code4.cs

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 |  |  |
| 2/27/17 |  |  |
| 2/28/17 |  |  |
| 3/1/17 |  |  |
| 3/2/17 |  |  |
| 3/3/17 |  |  |
| 3/4/17 |  |  |
| 3/5/17 |  |  |
| 3/6/17 |  |  |
| 3/7/17 |  |  |
| 3/8/17 |  |  |
| 3/9/17 |  |  |
| 3/10/17 |  |  |
| 3/11/17 |  |  |
| 3/12/17 |  |  |
| 3/13/17 |  |  |
| 3/14/17 |  |  |
| 3/15/17 |  |  |
| 3/16/17 |  |  |
| 3/17/17 |  |  |
| 3/18/17 |  |  |
| 3/19/17 |  |  |
| 3/20/17 |  |  |
|  |  |  |
| TOTAL 1 | 1725 |  |
| TOTAL 2 | 1700 |  |
| TOTAL 3 |  |  |
| RUNNING TOTAL | 3425 |  |

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 |  |  |
| FEB 26, 2017 |  |  |
| FEB 27, 2017 |  |  |
| FEB 28, 2017 |  |  |
| MAR 1, 2017 |  |  |
| MAR 2, 2017 |  |  |
| MAR 3, 2017 |  |  |
| MAR 4, 2017 |  |  |
| MAR 5, 2017 |  |  |
| MAR 6, 2017 |  |  |
| MAR 7, 2017 |  |  |
| MAR 8, 2017 |  |  |
| MAR 9, 2017 |  |  |
| MAR 10, 2017 |  |  |
| MAR 11, 2017 |  |  |
| MAR 12, 2017 |  |  |
| MAR 13, 2017 |  |  |
| MAR 14, 2017 |  |  |
| MAR 15, 2017 |  |  |
| MAR 16, 2017 |  |  |
| MAR 17, 2017 |  |  |
| MAR 18, 2017 |  |  |
| MAR 19, 2017 |  |  |
| MAR 20, 2017 |  |  |
|  |  |  |
| Project I Total | 1710 |
| PROJECT 2 TOTAL | 1690 |
| PROJECT 3 TOTAL |  |
| Running total | 3400 |