Project Report
Secure Login Demonstrator

Nandan Thareja & James Lunsford

Project Idea

Secure Login is a foundation of the modern internet. Services such as e-commerce, online banking, email, electronic medical records rely on users being both Authorized and Authenticated to perform whatever actions they request of the service. The normal method of verifying authorization and authentication is an implementation of secure login procedures and protocols. Our project implements secure login on a basic webpage as a concept demonstrator. The purpose of this project is to both teach us the concepts of secure login and be a lesson on the issues that can arise when you transition form concept to application. In the implementation of our project idea we will be applying policies learned in Information Security One using protocols learned in Information Security Two to create a working demonstrator of a critical part of today's internet experience. Specifically our project implements UAC, password rules, CSRF prevention and hashing methods from Information Security One with encryption technics and hashing techniques from Information Security Two.

Plan and Design

Plan

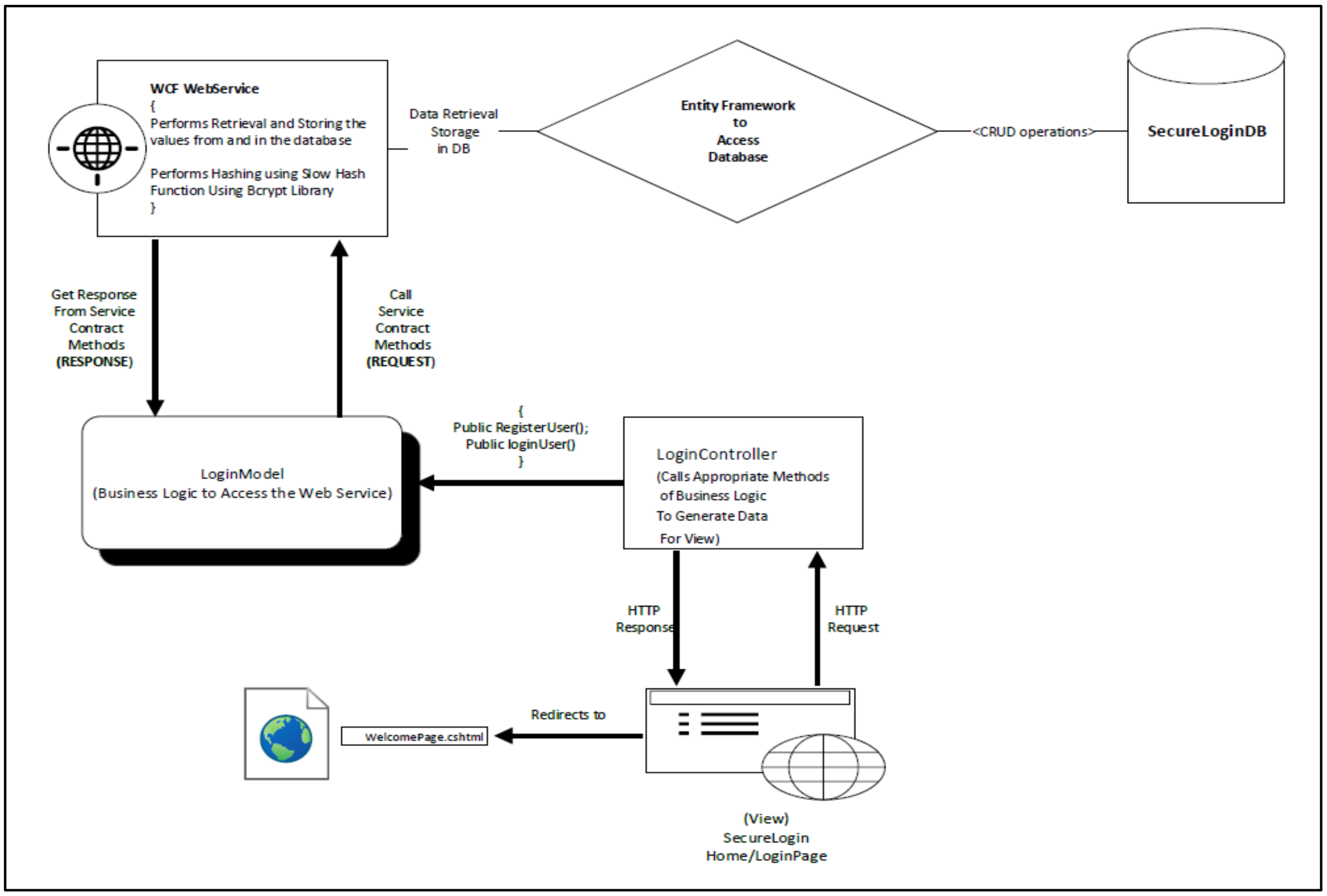
In planning our secure login demonstrator we quickly realized that we would need many interconnected parts to get a working demo and that we might have underestimated the difficulty in getting a seemingly simple concept applied to a real system. After analyzing the problem we were able to identify five discrete parts that we needed to make the demonstration work and plan out our implementation.

The five pieces to our secure login puzzle were identified as follows: website, application, server, database, and logging framework. However these parts do not make a secure whole unless they are implemented and connected according to best practices in security. To make sure that our demonstrator is secure we had to find a verifiable way to follow best practices and security principles. To do this we based our security policies and password rules on industry accepted standards published by SANS and our design supports the implementation of these policies. We decided that for a fully secure demonstrator we would comply with the following three SANS policies: Web Application Security Policy, Password Protection Policy, and Password Guidelines

Obviously this is a rather large scope so the next exercise in planning was for us to try and reduce the amount of effort without sacrificing quality. It was apparent from our plan that the website UI and the back end logging functions could be rudimentary as they are less essential to the security of the login process. By spending less time on web development through the use of template web pages and doing our logging with the log4net library we were able to focus our effort on the security critical components of our project.

Once we had the framework of our plan to guide us we began designing the individual parts of our system and how they would work together to keep our website and user data secure during the login process. The details of the completed design are described below.

Design



WCF WebService
{
Performs Retrieval and Storing the values from and in the database
Performs Hashing using Slow Hash Function Using Bcrypt Library
}
Data Retrieval Storage in DB
Entity Framework to Access Database
<CRUD operations>
SecureLoginDB
Get Response From Service Contract Methods (RESPONSE)
Call Service Contract Methods (REQUEST)
{
Public RegisterUser();
Public loginUser()
}
LoginModel
(Business Logic to Access the Web Service)
LoginController
(Calls Appropriate Methods of Business Logic To Generate Data For View)
HTTP Response
HTTP Request
Redirects to
WelcomePage.cshtml
(View)
SecureLogin
Home/LoginPage

Figure 1.0: Secure Login Application Design and Architecture

In figure 1.0 the overall architecture of the application has been demonstrated.

To implement Secure Login, the team was focused on three major tasks, Secure Access to the Application using SSL/TLS, Hashing of Passwords using Slow Hash functions and Logging using a logging framework which helped us to determine the vulnerabilities in the login system.

The First and Major task was to provide a service that could access the database in a secure manner without letting the user know the type, location and nature of the database. To accomplish this task the team implemented a Web Service using Windows Communication Foundation (WCF). This web service performed the major task of the application including the data validation, data sanitization and hashing of the passwords and other sensitive data. The web Application makes secure calls to the service contract methods of the web service which makes it difficult to the intruder to determine the techniques or algorithms used in the sensitive data hashing. All the relevant business logic that has been used in the web service contract methods can be seen in the file

'SecureLoginService.svc.cs '.

The second major task of the project is to hash passwords using slow has functions. The reason behind hashing passwords is they turn any amount of data into a fixed-length "fingerprint" that cannot be reversed. They also have the property that if the input changes by even a tiny bit, the resulting hash is completely different (see the example above). This is great for protecting passwords, because we want to store passwords in a form that protects them even if the password file itself is compromised, but at the same time, we need to be able to verify that a user's password is correct. To accomplish hashing of passwords we used BCrypt library in C#. Below are two Reasons why the team chose this library for salted hashing of the passwords:

- It is slow, and slow is good because it thwarts brute-force attacks (read more here [1]).
- The output from BCrypt is a Base-64 alphabet which means there are no characters that are tricky to store in a simple character field

There is an appropriate way of implementing hashing properly that the team figured out:

To Store a Password

- Generate a long random salt using a CSPRNG.
- Prepend the salt to the password and hash it with a **standard** cryptographic hash function such as SHA256.
- Save both the salt and the hash in the user's database record.

To Validate a Password

- Retrieve the user's salt and hash from the database.
- Prepend the salt to the given password and hash it using the same hash function.
- Compare the hash of the given password with the hash from the database. If they match, the password is correct. Otherwise, the password is incorrect.

The third most important part of the application was the log. A log is an important part of any application regardless of wat kind of application it is. For logging of the user and server activities the team found a suitable open source framework called log4net, which has been used. The Apache log4net library is a tool that helps the programmer output log statements to a variety of output targets. It is an open source library that allows .NET applications to log output.

Website

Our website is a simple site with only enough pages to support rudimentary login functionality. The website must only support, login/logout, and login errors and writing to the logging framework as necessary. To support the needed functionality we designed two webpages and a bootstrap alert box for all the relevant error messages or success messages returned by the application controller

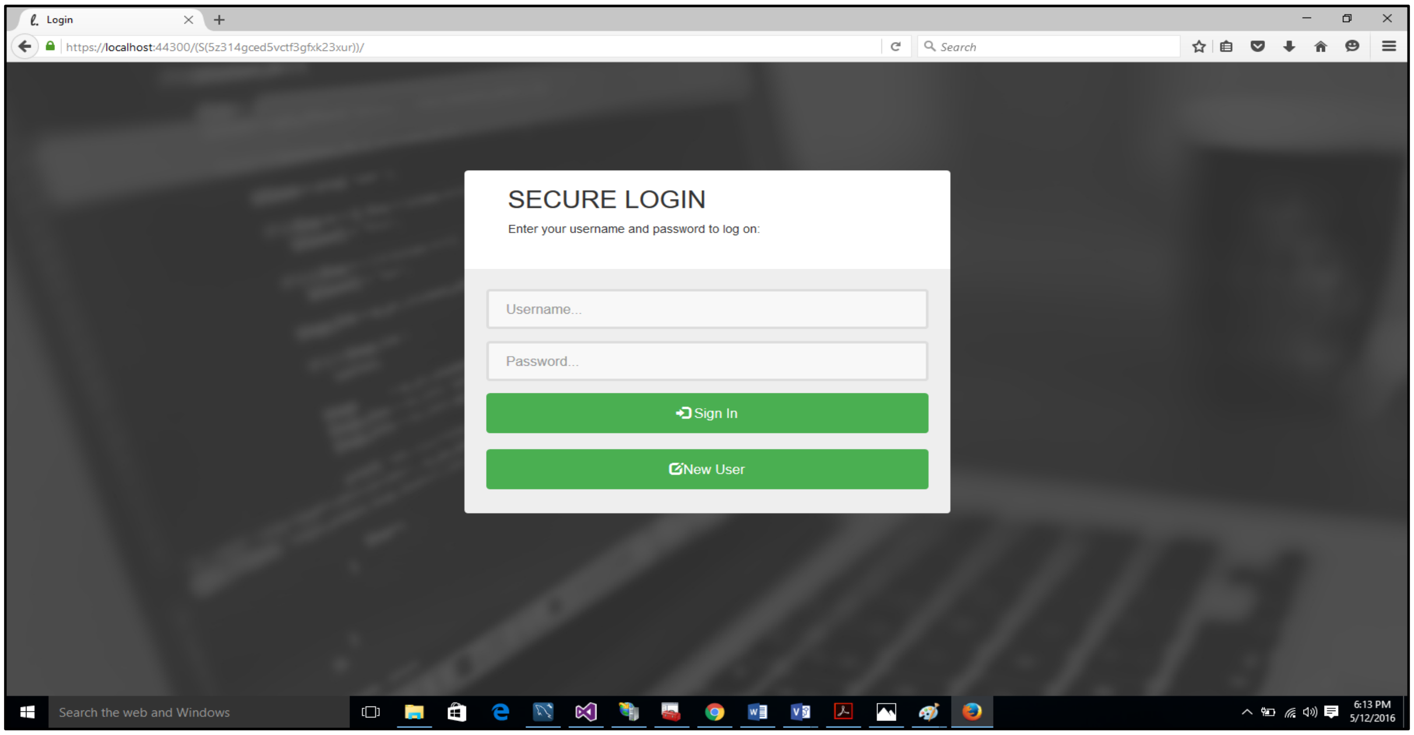
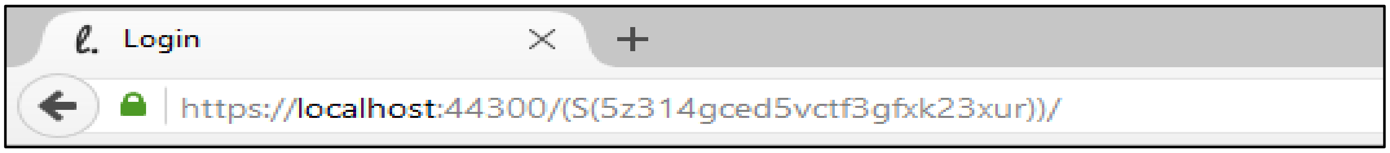

Login
https://localhost:44300/(S(5z314gced5vctf3gfxk23xur))/
Search
SECURE LOGIN
Enter your username and password to log on:
Username...
Password...
Sign In
New User
Search the web and Windows
6:13 PM
5/12/2016

Figure 2.0: Login Page

As shown in Figure 2.0, the login page is very simple with only two text boxes for user name and password. The security features build into these fields are described in the application section since the application contains the logic for them. The Login page also has a bootstrap alert box DOM element to show the relevant error and success messages. The client will observe a lock sign in the address bar as shown below.



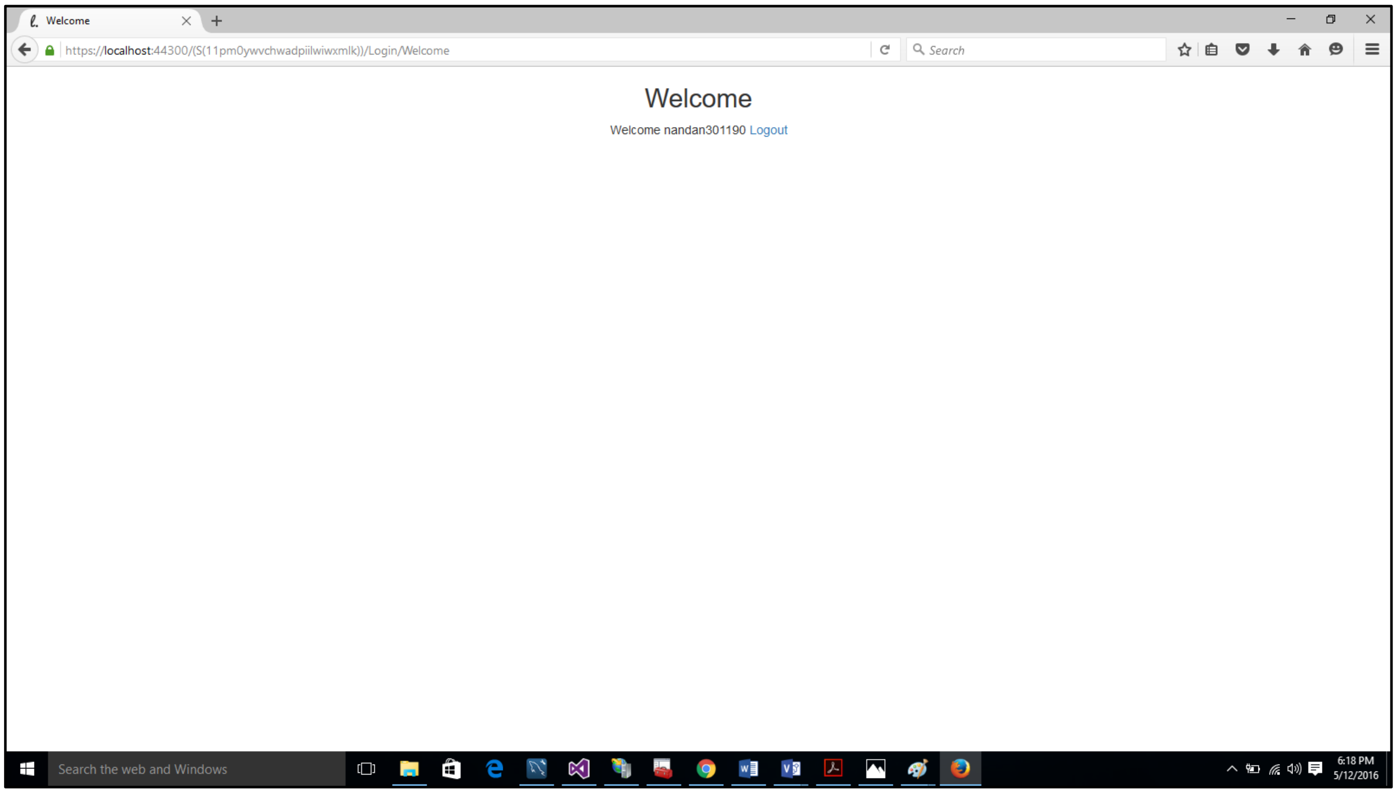
https://localhost:44300/(S(5z314gced5vctf3gfxk23xur))/

The Lock sign is a symbol of secure communication over transport layer also known as Transport Layer Security (TLS).

TLS is composed of two layers: the TLS Record Protocol and the TLS Handshake Protocol. The TLS Record Protocol provides connection security with some encryption method such as the Data Encryption Standard (DES) or any other encryption algorithms. The TLS Record Protocol can also be used without encryption. The TLS Handshake Protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before data is exchanged.

The team implemented this by generating self-signed certificates using private key using IIS 7.0. And importing the certificates in the Trusted Root Certification Authorities. The self-signed certificate generated using IIS 7.0 using SHA-1 as the thumbprint algorithm by default, which has been depreciated in most of the web browsers. To fix this the thumbprint algorithm can be migrated or upgraded to SHA256 by using certsrv.exe tools in windows server.

Below in figure 3.0, welcome page is shown that is redirected to if the user login is successful.



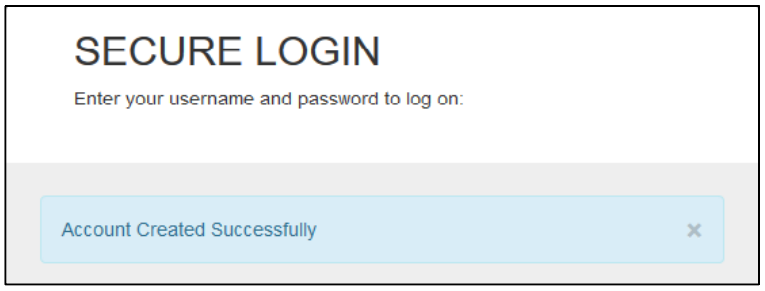
https://localhost:44300/(S(11pm0ywvchwadpiilwiwxmlk))/Login/Welcome

Welcome

Welcome nandan301190 Logout

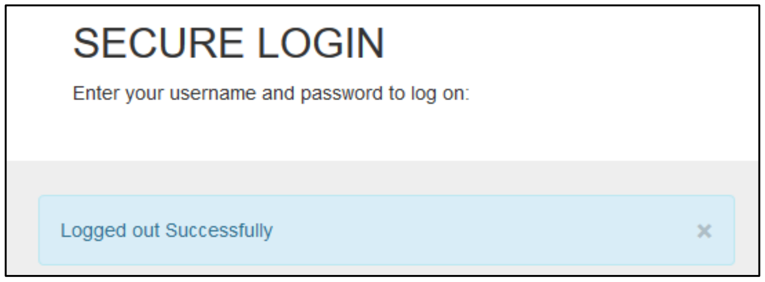
Figure 3.0: Welcome Page

The welcome page is just as simple as the login page, containing only a welcome message and a logout button. Since our project exists solely to demonstrate secure login, any further functionality on the welcome page would be extraneous.


SECURE LOGIN

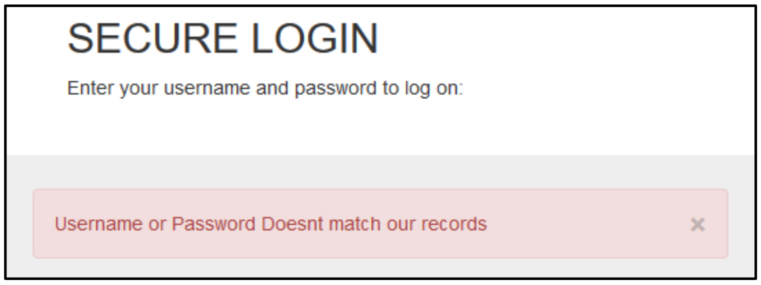
Enter your username and password to log on:

Account Created Successfully


SECURE LOGIN

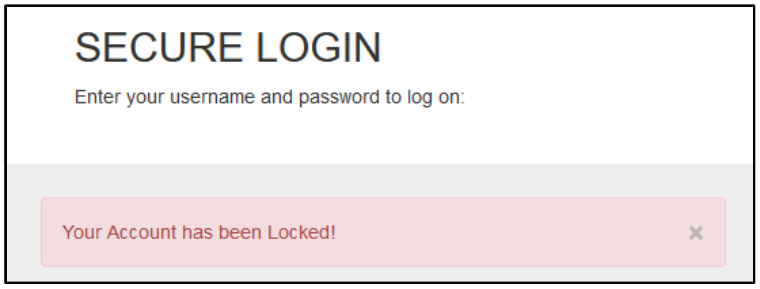
Enter your username and password to log on:

Logged out Successfully


SECURE LOGIN

Enter your username and password to log on:

Username or Password Doesnt match our records


SECURE LOGIN

Enter your username and password to log on:

Your Account has been Locked!

Figure 4.0: Error and Notification Messages

In figure 4.0, various error or notification messages has been shown which the user can encounter at various stages of access of the application. For instances messages has been shown if the user is successfully registered, or has been logged successfully, whereas error messages are shown if the user enters an incorrect username or password, and error message if the user account if locked after three unsuccessful login attempts.

Application

Our web application contains all of the functionality for our demonstrator. The application is built with **.net MVC 4 Framework**, and contains several components as needed to support our design. The application must accept user input from the web form, parse and sanitize the user input, send the attributes as a request to web service which in turn applies the hash function on the provided password in the request, query the database with the hashed password, evaluate if the password is correct, and return the result to the main component of the website called the model. Additionally, the application incorporates policies from the SANS Web Application Security Policy and creates entries to be stored in the logging framework. Two of these functions are critical to our demonstrator and the rest support the demonstrator's functionality.

The two major secure login demonstrator functions handled by the application are validating user input and applying the hash algorithm. With any web form validating and parsing user input is very important. This is the easiest place for a misuser of the system to inject code or attack the system. To prevent the (Cross Site Request Forgery) CSRF attack, Anti forgery token has been used which is provided by the development framework which in turn sanitize user input by stripping out escape characters and other malicious combinations which is validated in the application controller. Finally the hashing algorithm is applied after the password is sent to the web service the user input is hashed and compared to our stored hash values in the database using BCrypt library to look for a match.

The hashed password stored in the database separate from the web application in an effort to increase security this makes several techniques for accessing much harder than if the tables were simply some sort of collection in the application. We are using a salted hash implemented as described in the referenced "Password Hashing" website listed as reference four below. Although user input sanitation and hashing are the most important functions of our application they are not the only uses.

Additionally the application features a counter to lock out users after 3 attempts in an effort to prevent DOS attacks on our server. When any given IP address originates three incorrect login attempts that IP address will be blocked from making additional attempts for 2 minutes. This will be an effective buffer to prevent DOS attacks.

Database

For the database we decided to use **MySQL** as we need only two tables to be referenced by one client at a time. This means that any more robust implementation would likely be overkill and a waste of development effort. Additionally the hashed password is stored according to best practices and consequently is as secure as we can make it. The hashed password gets stored in the database by the Web service implemented using the windows communication foundation (WCF).The database has two entities which are shown below:

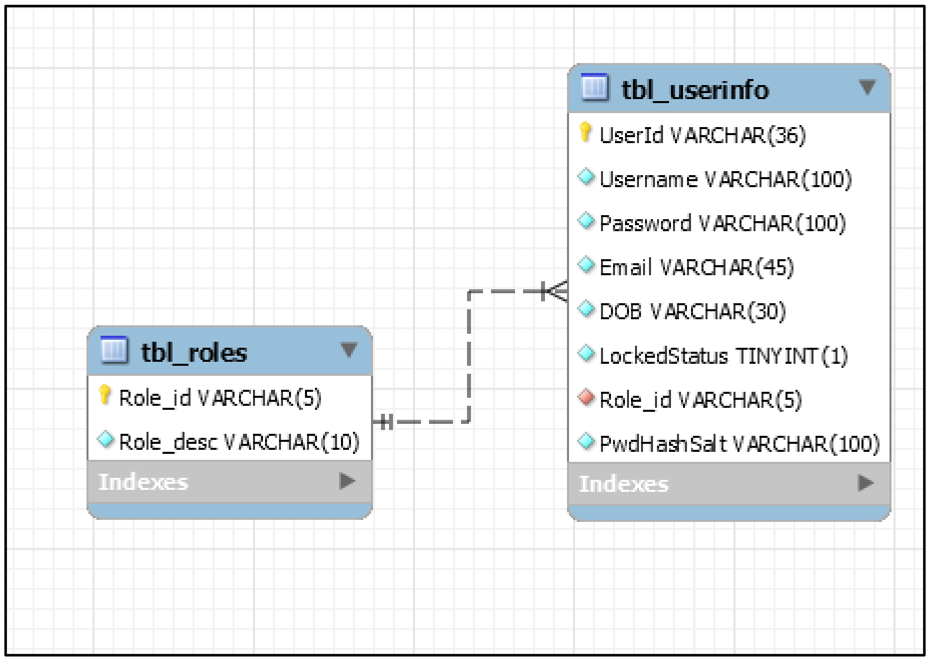
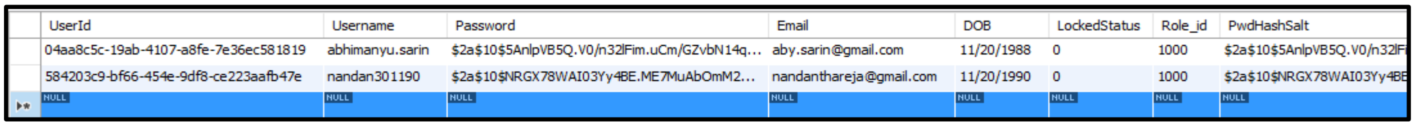


Figure 5.0 SecureLoginDB Model

The hashing is performed using BCrypt library in C#. Below code performs hashing of the passwords using a slow hash function:

```
//Generating Salt for Pwd
string pwdSalt = BCrypt.Net.BCrypt.GenerateSalt();
//Hashing pwd
 hashed_pwd = BCrypt.Net.BCrypt.HashPassword(Password, pwdSalt);
```

The hashed password stored in the database looks as following:



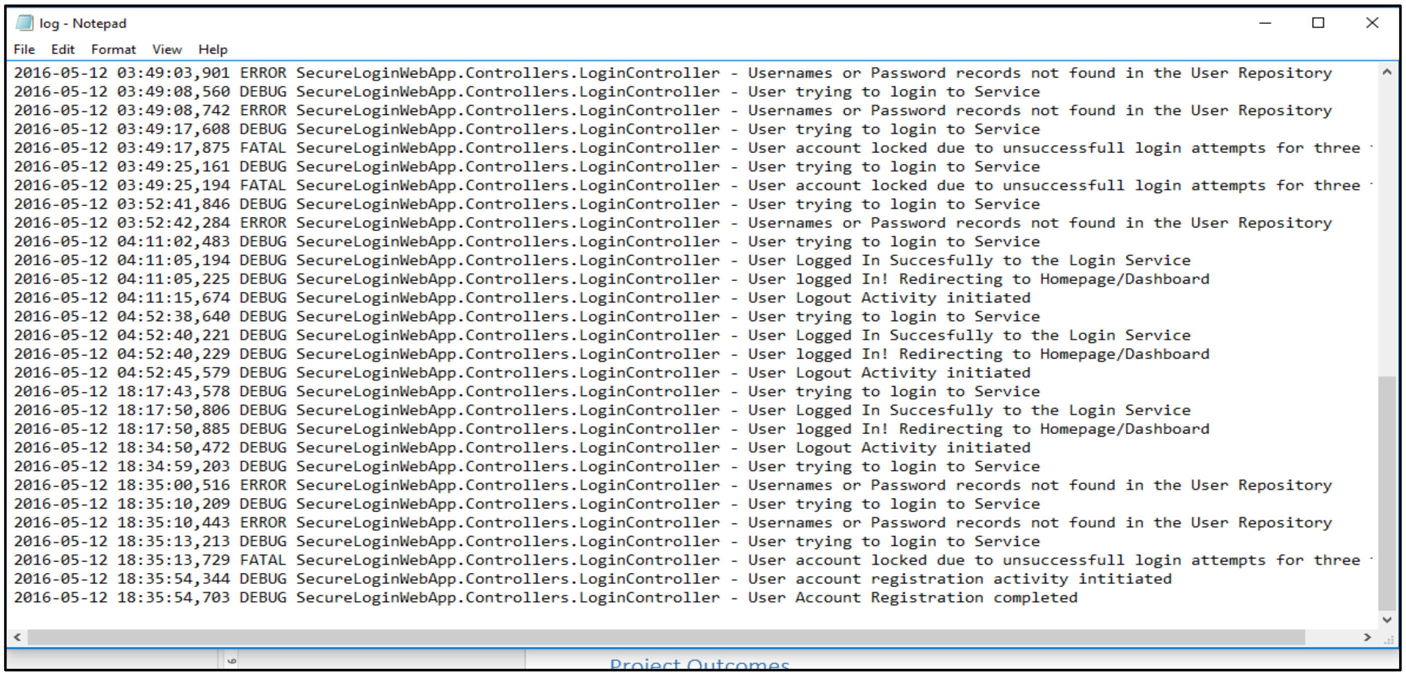
	UserId	Username	Password	Email	DOB	LockedStatus	Role_id	PwdHashSalt
	04aa8c5c-19ab-4107-a8fe-7e36ec581819	abhimanyu.sarin	$2a$10$5AnlpVB5Q.V0/n32lFim.uCm/GZvbN14q...	aby.sarin@gmail.com	11/20/1988	0	1000	$2a$10$5AnlpVB5Q.V0/n32lF
	584203c9-bf66-454e-9df8-ce223aafb47e	nandan301190	$2a$10$NRGX78WAI03Yy4BE.ME7MuAbOmM2...	nandanthareja@gmail.com	11/20/1990	0	1000	$2a$10$NRGX78WAI03Yy4BE
▶*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The comparison of the hashed password and user provided password is performed by the Bcrypt library to validate the provided password. This can be done using the following code as shown below:

```
//Verify password
bool success = BCrypt.Net.BCrypt.Verify(Password, usrHashedPwd); Logging
```

Logging

For the logging functionality we chose **Log4net** since it is compatible with our development environment and easily implemented. Log4 J logging framework allows us to set logging statements at several different levels and use a configuration file to declare the logging level without editing our source code. This was very beneficial as we used "Debug" logs to help during development and were able to increase the log level to "Error" upon deployment. The application log appends a file called '***log.txt***', which appears to be as shown below in figure:



```
2016-05-12 03:49:03,901 ERROR SecureLoginWebApp.Controllers.LoginController - Usernames or Password records not found in the User Repository
2016-05-12 03:49:08,560 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 03:49:08,742 ERROR SecureLoginWebApp.Controllers.LoginController - Usernames or Password records not found in the User Repository
2016-05-12 03:49:17,608 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 03:49:17,875 FATAL SecureLoginWebApp.Controllers.LoginController - User account locked due to unsuccessfull login attempts for three
2016-05-12 03:49:25,161 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 03:49:25,194 FATAL SecureLoginWebApp.Controllers.LoginController - User account locked due to unsuccessfull login attempts for three
2016-05-12 03:52:41,846 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 03:52:42,284 ERROR SecureLoginWebApp.Controllers.LoginController - Usernames or Password records not found in the User Repository
2016-05-12 04:11:02,483 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 04:11:05,194 DEBUG SecureLoginWebApp.Controllers.LoginController - User Logged In Succesfully to the Login Service
2016-05-12 04:11:05,225 DEBUG SecureLoginWebApp.Controllers.LoginController - User logged In! Redirecting to Homepage/Dashboard
2016-05-12 04:11:15,674 DEBUG SecureLoginWebApp.Controllers.LoginController - User Logout Activity initiated
2016-05-12 04:52:38,640 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 04:52:40,221 DEBUG SecureLoginWebApp.Controllers.LoginController - User Logged In Succesfully to the Login Service
2016-05-12 04:52:40,229 DEBUG SecureLoginWebApp.Controllers.LoginController - User logged In! Redirecting to Homepage/Dashboard
2016-05-12 04:52:45,579 DEBUG SecureLoginWebApp.Controllers.LoginController - User Logout Activity initiated
2016-05-12 18:17:43,578 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 18:17:50,806 DEBUG SecureLoginWebApp.Controllers.LoginController - User Logged In Succesfully to the Login Service
2016-05-12 18:17:50,885 DEBUG SecureLoginWebApp.Controllers.LoginController - User logged In! Redirecting to Homepage/Dashboard
2016-05-12 18:34:50,472 DEBUG SecureLoginWebApp.Controllers.LoginController - User Logout Activity initiated
2016-05-12 18:34:59,203 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 18:35:00,516 ERROR SecureLoginWebApp.Controllers.LoginController - Usernames or Password records not found in the User Repository
2016-05-12 18:35:10,209 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 18:35:10,443 ERROR SecureLoginWebApp.Controllers.LoginController - Usernames or Password records not found in the User Repository
2016-05-12 18:35:13,213 DEBUG SecureLoginWebApp.Controllers.LoginController - User trying to login to Service
2016-05-12 18:35:13,729 FATAL SecureLoginWebApp.Controllers.LoginController - User account locked due to unsuccessfull login attempts for three
2016-05-12 18:35:54,344 DEBUG SecureLoginWebApp.Controllers.LoginController - User account registration activity intitiated
2016-05-12 18:35:54,703 DEBUG SecureLoginWebApp.Controllers.LoginController - User Account Registration completed
```

Figure 5.0: Log.txt File View

Project Outcomes

Implementation of Secure Login

Our implementation was successful and we were able to create a secure demonstrator of an internet based login. The demonstrator is built according to the described plan and by following the above design. This demonstrator includes a correct, secure hashing scheme, database, web application, framework for secure DB transactions, Implementation of Transport Layer Security(TLS) and built in accordance with the referenced SANS standards. Additionally the user password rules policy is enforced by the web application. Although the system turned out well and we consider it a success there were some missteps and difficulties along the way. These stumbling blocks are described in the lessons learned below which are as important as the overall result to the team's learning process throughout this project.

Lessons Learned

- Leave time for testing: We did not penetration test our project. Penetration testing could have potentially discovered unknown flaws or weaknesses in our design and no project can truly be deemed secure until it has been evaluated using thorough penetration testing.
- Make use of templates: Using website templates saved a lot of time during development and provided professional design services without the associated fees.
- Use libraries for encryption and hashing: We did not try to implement hash or encryption ourselves due to great risk in doing so.

References

- Password Guidelines:
  https://www.sans.org/security-resources/policies/general/pdf/password-construction-guidelines
- Password Protection Policy:
  https://www.sans.org/security-resources/policies/general/pdf/password-protection-policy
- Web Application Security Policy:
  https://www.sans.org/security-resources/policies/application-security/pdf/web-application-security-policy
- Password Hashing:
  https://crackstation.net/hashing-security.htm
- BCrypt Library C# Advantages:
  http://security.stackexchange.com/questions/4781/do-any-security-experts-recommend-bcrypt-for-password-storage