パスワード認証

2015 年 8 月 3 日

@twyujiro15

YK Software

プロフィール

加藤 裕次郎

1982.03.03 生まれ（うお座）

左利き（お箸は右）

twitter : @twyujiro15

プログラミング経験

Excel VBA
MATLAB、MATX
C
VC++ (Windows SDK)
VC++ (MFC)
WPF + C#

本職は製造業の開発業務

- 2009 年 4 月に入社
  - ・組み込みソフトウェア開発で初めてまともに C 言語
  - ・デバッグソフトで Visual C++（MFC、Windows SDK）
  - ・Excel 大好きマンだったので VBA も使用

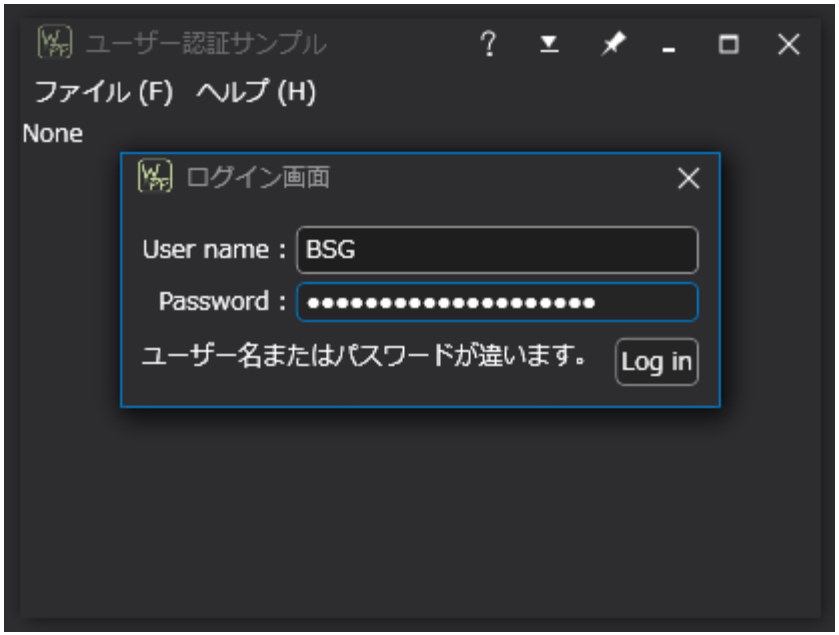
VBA でソケット通信したときは感動した

- 2013 年 10 月
  - ・あるサンプルが "WPF" なるものでできていることを知る

独自調査から @okazuki さんの MVVM 入門四則演算にたどり着く

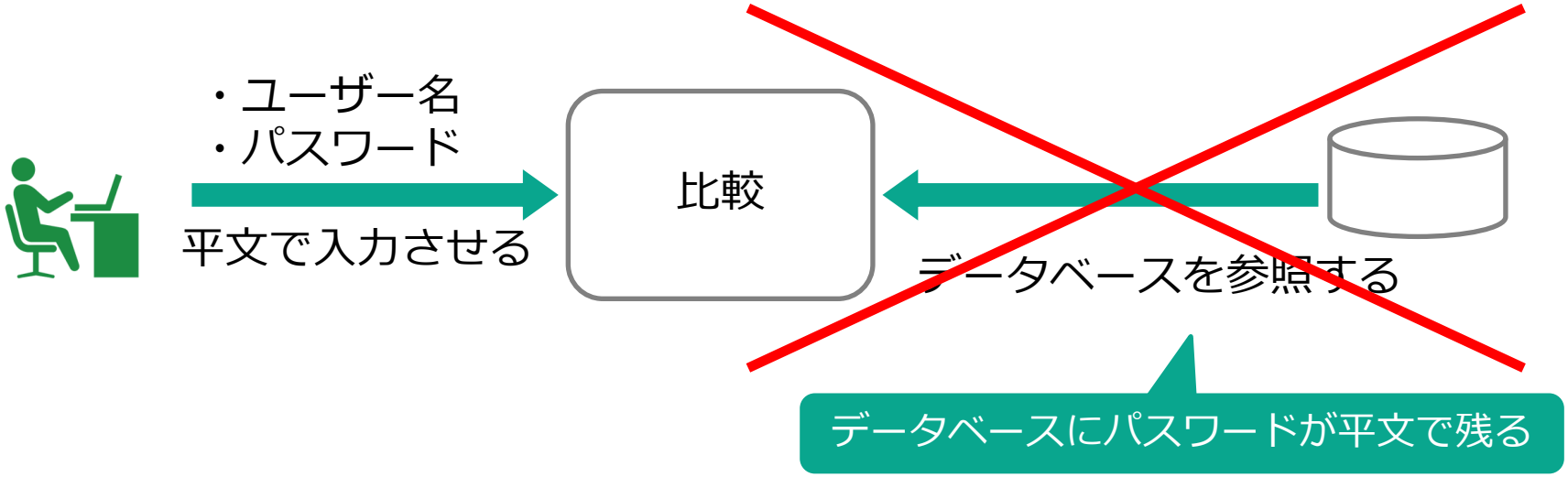
今回のゴール

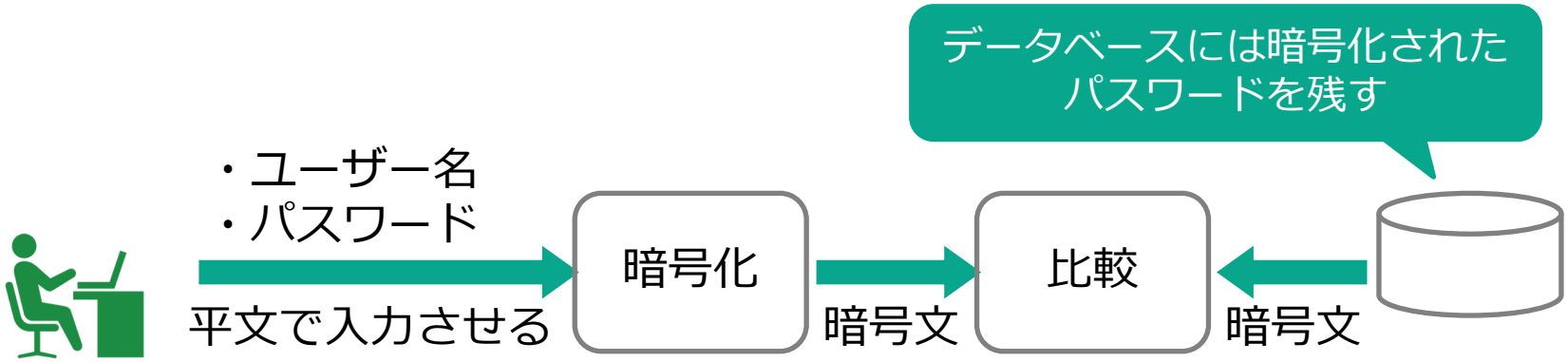
パスワード認証できるようにする



- パスワードを暗号化する
- SecureString 型を知る

パスワード認証の落とし穴（素人編）





どんな暗号化をすればいいの？

ハッシュ関数

よく使われる暗号化手法です

- 可変長入力に対して固定長出力である
- 入力から出力を求めることは容易
- 出力から入力を求めることは計算量的に困難
- 出力が同一となる入力の一つを求めることも計算量的に困難

データベースにはハッシュ値を保存しておきましょう

でもちょっと待って！これって本当に安全なの？

頑張れば求められてしまう

基本的にパスワードは人間が入力するもの

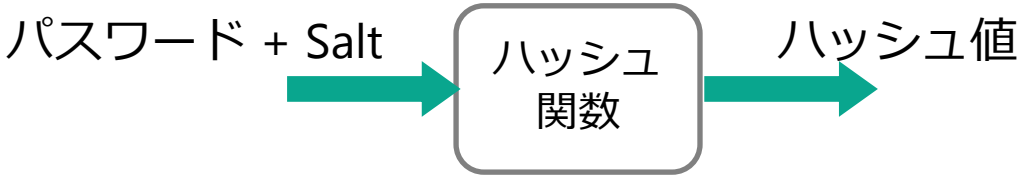
- せいぜい長くても 10 文字前後 → パターンが限られてしまう
- パスワードになりそうな文字のハッシュ値をあらかじめ用意

→ 総当たりで検索されてしまう

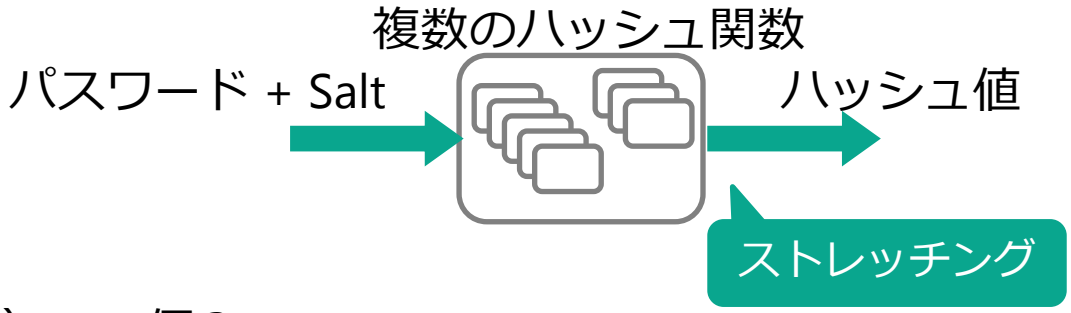
総当たり攻撃を想定しておく

・考え方が甘い

・詰めが甘い



・より安全



Salt（ソルト）って何？

・機械的にランダムなデータを付加することでパスワードの推測を回避

SHA256 のハッシュ値を算出する

```
/// <summary>
/// Salt + パスワードのハッシュ値を算出します。
/// </summary>
/// <param name="userId">Salt に用いる ID を指定します。</param>
/// <param name="password">パスワードを指定します。</param>
/// <returns>ハッシュ値を返します。</returns>
public static string GetSaltedPassword(string userId, string password)
{
    var salt = GetSha256(userId);
    return GetSha256(salt + password);
}

/// <summary>
/// 文字列から SHA256 のハッシュ値を算出します。
/// </summary>
/// <param name="target">変換する文字列を指定します。</param>
/// <returns>ハッシュ値を返します。</returns>
private static string GetSha256(string target)
{
    byte[] byteValue = Encoding.UTF8.GetBytes(target);
    var sha256 = SHA256Managed.Create();
    byte[] hash = sha256.ComputeHash(byteArray);
    var buf = new StringBuilder();
    foreach (var s in hash)
    {
        buf.AppendFormat("{0:x2}", s);
    }
    return buf.ToString();
}
```

複数回ハッシュでストレッチング

```
/// <summary>
/// Salt + ストレッチングしたパスワードのハッシュ値を算出します。
/// </summary>
/// <param name="userId">Salt に用いる ID を指定します。</param>
/// <param name="password">パスワードを指定します。</param>
/// <returns>ハッシュ値を返します。</returns>
public static string GetStretchedPassword(string userId, string password)
{
    var salt = GetSha256(userId);
    var hash = string.Empty;
    for (var i = 0; i < StretchCount; i++)
    {
        hash = GetSha256(hash + salt + password);
    }
    return hash;
}
```

やり方はいろいろあるんだと思う

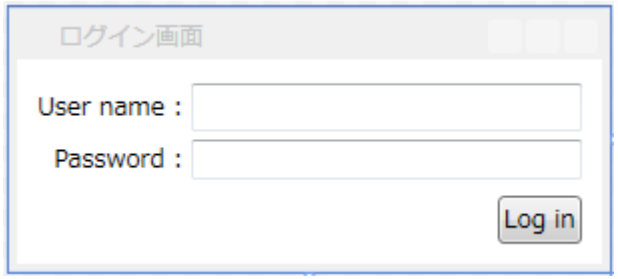
例えばこんなハッシュ値が出力されます

1e15e40501a8e2c9e58d7d8972bfb0412d471739f4e8b215362f6eb2491e4e90

よし、次は UI の実装だ。

UI の実装

まあシンプルに PasswordBox コントロールを使いましょう。




```
<Grid FocusManager.FocusedElement="{Binding ElementName=textBox}" Margin="10">
    <Grid.RowDefinitions>
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto" />
        <RowDefinition Height="Auto
    </Grid.RowDefinitions>
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width=  o" />
        <ColumnDefinition />
    </Grid.ColumnDefinitio
    <TextBlock Grid.Row  0" Grid.Column="0" Text="User name : " />
    <TextBlock Grid.R w="1" Grid.Column="0" Text="Password : " />
    <TextBox Grid Row="0" Grid.Column="1" Text="{Binding UserName}" />
    <PasswordBox Grid.Row="1" Grid.Column="1" />
    <Button Grid.Row="2" Grid.Column="1" Content="Log in" Command="{Binding LogInCommand}" />
    <TextBlock Text="{Binding Message}" Grid.Row="2" Grid.ColumnSpan="2" />
</Grid>
```

PasswordBox ってどうやってバインドするの？

SecurePassword プロパティ

PasswordBox.Password プロパティは依存関係プロパティではない！

- 依存関係プロパティにするとマネジド領域にパスワードが平文で残る
- マネジド領域は GC のタイミングによってメモリ解放される

セキュリティ上よろしくない

SecureString 型 の PasswordBox.SecurePassword プロパティを使いましょう。

SecureString 型？

- string と同様に文字列を扱う型
- 文字列が自動的に暗号化される
- IDisposable インターフェースを実装 → 任意タイミングで破棄できる
- スレッドの中断などでも確実に破棄されることがランタイムで保証される

でも PasswordBox.SecureString プロパティも依存関係プロパティじゃないんだけど…

ビヘイビアでバインドできるようにする

長くなるので概略だけ説明

```
using System.Security;
using System.Windows;
using System.Windows.Controls;

public class SecurePasswordBindingBehavior
{
    IsEnabled 添付プロパティ

    SecurePassword 添付プロパティ

    PasswordChanged イベントハンドラ
}
```

このプロパティにバインドする

- PasswordBox コントロールに添付することが前提
- IsEnabled 添付プロパティが true のときにイベントハンドラを登録
- PasswordBox の PasswordChanged イベントハンドラで処理をおこなう
  - PasswordBox.SecurePassword を SecurePassword 添付プロパティにコピー

本当は SecureString さえも依存関係プロパティにしないほうがいいのかも

→ このビヘイビア内で暗号化してそのデータをバインドしたほうがいいかも

参考 URL

[1] パスワードとハッシュとユーザー認証

http://csharp.keicode.com/topics/password-hashing.php

[2] 安全なパスワードの保存方法（ASP.NET, C#, VB.NET編）

http://www.websec-room.com/2013/02/27/239

[3] 安全なパスワードの保存方法

http://www.websec-room.com/2013/02/27/237

[4] SecureString クラス

http://smdn.jp/programming/netfx/string/6_securestring/

[5] C# + WPFで、GenericPrincipalを使った認証と承認を行い、画面の表示を制限する

http://thinkami.hatenablog.com/entry/2014/08/26/055052

ご清聴ありがとうございました

YK Software