**Звіт**

**до предмету «Основи інформаційної безпеки»**

Студента Київського Національного Університету ім. Тараса Шевченко

Пруніча Сергія Миколайовича

Факультет інформаційних технологій

Група МІТ-21



Лабораторна робота №5

Тема: «Безпечне зберігання паролів»

3. Розробити клас SaltedHash, що реалізує хешування паролів із додаванням додаткової ентропії. Продемонструвати роботу класу, обчислюючи хеш для заданого пароля та "солі".

4. Розробити клас PBKDF2, що має наступну функціональність: генерує "сіль", задає алгоритм хешування (MD5, SHA1, SHA256, SHA384, SHA512) та обчислює хеш для заданого числа ітерацій. Створити програму, що обчислює час, витрачений на обчислення хешу для різного числа ітерацій (10 значень із кроком 50'000; перше значення = номер варіанта \* 10'000 ). Побудувати графік залежності витраченого часу від числа ітерацій.

5. Написати програму, що реалізує хешування введеного пароля під час реєстрації користувача та зберігає логін, пароль та "сіль" у пам'яті. Реалізувати можливість автентифікації за логіном і паролем. Число ітерацій = номер варіанта \* 10'000.

#1

using System.Security.Cryptography;

using System.Text;

class Salted\_Hash

{

static void Main()

{

Console.WriteLine("Enter your password:");

string password = Console.ReadLine();

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");

byte[] salt = SaltedHash.GenerateSalt();

Console.WriteLine("Password: " + password);

Console.WriteLine("Salt: " + Convert.ToBase64String(salt));

Console.WriteLine("---------------------------------------------------------------");

var hashedPassword = SaltedHash.HashingviaSalt(Encoding.UTF8.GetBytes(password), salt);

Console.WriteLine("Hashed Password: " + Convert.ToBase64String(hashedPassword));

Console.ReadLine();

}

}

public class SaltedHash

{

public static byte[] GenerateSalt()

{

const int salt\_length = 32;

using (var randomNumberGenerator = RandomNumberGenerator.Create())

{

var rnd = new byte[salt\_length];

randomNumberGenerator.GetBytes(rnd);

return rnd;

}

}

private static byte[] Combine(byte[] pas, byte[] salt)

{

var unit = new byte[pas.Length + salt.Length];

Buffer.BlockCopy(pas, 0, unit, 0, pas.Length);

Buffer.BlockCopy(salt, 0, unit, pas.Length, salt.Length);

return unit;

}

public static byte[] HashingviaSalt(byte[] toBeHashed, byte[] salting)

{

using (var sha256 = SHA256.Create())

{

return sha256.ComputeHash(Combine(toBeHashed, salting));

}

}

}

#2

using System.Security.Cryptography;

using System.Text;

using System.Diagnostics;

class PBKDF2\_Hash

{

static void Main()

{

do

{

Console.WriteLine("Want you to run program?" + " YES / NO");

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

string choice = Console.ReadLine();

Console.WriteLine();

if (choice.ToLower()=="yes")

{

Console.WriteLine("Choose which method for hashing you want to use:");

Console.WriteLine("1 - SHA1");

Console.WriteLine("2 - SHA256");

Console.WriteLine("3 - SHA384");

Console.WriteLine("4 - SHA512");

Console.WriteLine("5 - MD5");

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

int option = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

Console.WriteLine("Enter the password you want to hash:");

string password = null;

while (true)

{

var key = Console.ReadKey(true);

if (key.Key == ConsoleKey.Enter)

break;

password += key.KeyChar;

}

int iteration = 40000;

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");

HashAlgorithmName hashAlgorithmName;

switch (option)

{

case 1:

hashAlgorithmName = HashAlgorithmName.SHA1;

for (int i = 0; i < 10; i++)

{

PBKDF2.HashPassword(password, iteration, hashAlgorithmName);

iteration += 50000;

}

break;

case 2:

hashAlgorithmName = HashAlgorithmName.SHA256;

for (int i = 0; i < 10; i++)

{

PBKDF2.HashPassword(password, iteration, hashAlgorithmName);

iteration = iteration + 50000;

}

break;

case 3:

hashAlgorithmName = HashAlgorithmName.SHA384;

for (int i = 0; i < 10; i++)

{

PBKDF2.HashPassword(password, iteration, hashAlgorithmName);

iteration = iteration + 50000;

}

break;

case 4:

hashAlgorithmName = HashAlgorithmName.SHA512;

for (int i = 0; i < 10; i++)

{

PBKDF2.HashPassword(password, iteration, hashAlgorithmName);

iteration = iteration + 50000;

}

break;

case 5:

hashAlgorithmName = HashAlgorithmName.MD5;

for (int i = 0; i < 10; i++)

{

PBKDF2.HashPassword(password, iteration, hashAlgorithmName);

iteration = iteration + 50000;

}

break;

}

}

if (choice.ToLower() == "no")

{

break;

}

}

while (true);

}

}

public class PBKDF2

{

public static byte[] GenerateSalt()

{

const int salt\_length = 32;

using (var randomNumberGenerator = RandomNumberGenerator.Create())

{

var rnd = new byte[salt\_length];

randomNumberGenerator.GetBytes(rnd);

return rnd;

}

}

public static void HashPassword(string passwordToHash, int numOfRounds, HashAlgorithmName hashalgorithmname)

{

var sw = new Stopwatch();

sw.Start();

var hashedPassword = PBKDF2.HashPasswordhash(Encoding.UTF8.GetBytes(passwordToHash), PBKDF2.GenerateSalt(), numOfRounds, hashalgorithmname);

sw.Stop();

Console.WriteLine("Password to hash:" + passwordToHash);

Console.WriteLine("Hashed Password via " + hashalgorithmname + " :" + Convert.ToBase64String(hashedPassword));

Console.WriteLine("Iterations <" + numOfRounds + "> Elapsed Time: " + sw.ElapsedMilliseconds + "ms");

Console.WriteLine();

}

public static byte[] HashPasswordhash(byte[] toBeHashed, byte[] salt, int numOfRounds, HashAlgorithmName hashalgorithm)

{

using (var rfc2898 = new Rfc2898DeriveBytes(toBeHashed, salt, numOfRounds, hashalgorithm))

{

int size = 0;

if (hashalgorithm == HashAlgorithmName.SHA1)

{

size = 20;

}

if (hashalgorithm == HashAlgorithmName.SHA256)

{

size = 32;

}

if (hashalgorithm == HashAlgorithmName.SHA384)

{

size = 48;

}

if (hashalgorithm == HashAlgorithmName.SHA512)

{

size = 64;

}

if (hashalgorithm == HashAlgorithmName.MD5)

{

size = 16;

}

return rfc2898.GetBytes(size);

}

}

}

#3

using System.Security.Cryptography;

using System.Text;

class HashProgram

{

public static void Main()

{

int j = 0;

string[] data\_login = new string[10];

string[] data\_password = new string[10];

string[] data\_salt = new string[10];

do

{

Console.WriteLine("Choose your option:");

Console.WriteLine("1 - Create your personal office");

Console.WriteLine("2 - To verify your data");

Console.WriteLine("3 - Exit");

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");

int option = Convert.ToInt32(Console.ReadLine());

Console.WriteLine("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");

switch (option)

{

case 1:

Console.WriteLine("\n-----------------------------");

Console.WriteLine("Authentication");

Console.WriteLine("-----------------------------\n\n");

Console.WriteLine("Enter your login:");

string login\_input = Console.ReadLine();

byte[] login\_repeat = Encoding.UTF8.GetBytes(login\_input);

Console.WriteLine("Enter your password:");

string pas\_input = null;

while (true)

{

var key = Console.ReadKey(true);

if (key.Key == ConsoleKey.Enter)

break;

pas\_input += key.KeyChar;

}

Console.WriteLine("\n...");

Console.WriteLine("Processing");

Console.WriteLine("...\n");

byte[] login\_byte = Encoding.Unicode.GetBytes(login\_input);

byte[] salt = GenerateSalt();

string SALT = Convert.ToBase64String(salt);

string hashing = "";

int iteration = 40000;

for (int i = 0; i < 10; i++)

{

hashing = HashingPassword(pas\_input, SALT, iteration);

iteration += 50000;

}

data\_login[j] = Convert.ToBase64String(login\_byte);

data\_password[j] = hashing;

data\_salt[j] = SALT;

j++;

Console.WriteLine("\n\n-----------------------------");

Console.WriteLine("Done");

Console.ReadLine();

break;

case 2:

Console.WriteLine("\n\n-----------------------------");

Console.WriteLine("Verification");

Console.WriteLine("\n\n-----------------------------");

Console.WriteLine("Enter your login:");

string login\_verify = Console.ReadLine();

Console.WriteLine("Enter the password:");

string pas\_verify = null;

while (true)

{

var key = Console.ReadKey(true);

if (key.Key == ConsoleKey.Enter)

break;

pas\_verify += key.KeyChar;

}

byte[] login\_verify\_byte = Encoding.Unicode.GetBytes(login\_verify);

int index\_login = Array.IndexOf(data\_login, Convert.ToBase64String(login\_verify\_byte));

string salt\_verify = data\_salt[index\_login];

string hashing\_verify = "";

int iteration\_verify = 40000;

for (int i = 0; i < 10; i++)

{

hashing\_verify = HashingPassword(pas\_verify, salt\_verify, iteration\_verify);

iteration\_verify += 50000;

}

if (data\_password.Contains(hashing\_verify))

{

Console.WriteLine("\n\n-----------------------------");

Console.WriteLine("Such user exists\n\n");

}

else

{

Console.WriteLine("\n\n-----------------------------");

Console.WriteLine("Such user does not exist\n\n");

}

break;

Console.ReadLine();

case 3:

Environment.Exit(0);

break;

default:

Console.WriteLine("Choose an option from the list above");

break;

}

}

while (true);

}

public static byte[] GenerateSalt()

{

const int salt\_length = 32;

using (var randomNumberGenerator = RandomNumberGenerator.Create())

{

var rnd = new byte[salt\_length];

randomNumberGenerator.GetBytes(rnd);

return rnd;

}

}

private static string HashingPassword(string passwordToHash, string salt, int numOfRounds)

{

byte[] salt\_byte = Encoding.UTF8.GetBytes(salt);

var hashedPassword = HashAuthentication.HashPassword(Encoding.UTF8.GetBytes(passwordToHash), salt\_byte, numOfRounds);

return Convert.ToBase64String(hashedPassword);

}

}

public class HashAuthentication

{

public static byte[] HashPassword(byte[] toBeHashed, byte[] salt, int numOfRounds)

{

using (var rfc2898 = new Rfc2898DeriveBytes(toBeHashed, salt, numOfRounds))

{

return rfc2898.GetBytes(20);

}

}

}