Министерство образования Республики Беларусь

Учреждение образования

Брестский государственный технический университет

Кафедра ИИТ

Лабораторная работа №3

За 5 семестр

По дисциплине «Веб технологии»

Тема: «Защита данных в веб-приложениях»

Выполнил: студент 3 курса

Группы ПО-4(2)

Коташевич С.Н.

Проверил: Михняев А. Л.

Брест 2021

Лабораторная работа №3

Цель: Познакомиться с подходами защиты данных в веб-приложениях.

Ход работы:

1. Передать шифрованные данные в GET в запросе, захешировав при этом параметры URL.





public IActionResult Hash(LoginModel model) => View(model);

[Route("MakeHash")]

public IActionResult MakeHash(LoginModel model)

{

var result = new LoginModel()

{

Email = GetHash(model.Email),

Password = GetHash(model.Password)

};

return RedirectToAction("Hash", result);

}

private string GetHash(string input)

{

var md5 = MD5.Create();

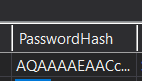
var hash = md5.ComputeHash(Encoding.UTF8.GetBytes(input));

return Convert.ToBase64String(hash);

}

1. Разработать функционал, осуществляющий:

* регистрацию и аутентификацию пользователей
* проверку уникальности учётной записи (логина) при регистрации
* хранение пароля в базе данных в шифрованной форме
* передачу данных и параметров в HTTP-запросах осуществляется в шифрованной форме



public IActionResult Registration()

{

ViewBag.Title = "Shop";

return View();

}

[HttpPost]

public async Task<IActionResult> Registration(RegistrationModel model)

{

if (ModelState.IsValid)

{

var temp = await \_signInManager.UserManager.FindByEmailAsync(model.Email);

if (temp == null)

{

UserModel user = new UserModel() {Email = model.Email, UserName = model.NickName};

var result = await \_userManager.CreateAsync(user, model.Password);

if (result.Succeeded)

await \_signInManager.SignInAsync(user, false);

return RedirectToAction(actionName: "Index", controllerName: "Home");

}

ModelState.AddModelError("err", "This user already exists");

}

return View(model);

}

public IActionResult Authorize() => View();

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Authorize(LoginModel model)

{

var user = await \_signInManager.UserManager.FindByEmailAsync(model.Email);

if (user != null)

{

var result = await \_signInManager.PasswordSignInAsync(user.UserName, model.Password, false, false);

if (result.Succeeded)

return RedirectToAction(actionName: "Index", controllerName: "Home");

}

return RedirectToAction(actionName: "Authorize", controllerName: "Authorization");

}

[HttpPost]

public async Task<IActionResult> LogOut()

{

await \_signInManager.SignOutAsync();

return RedirectToAction(actionName: "Index", controllerName: "Home");

}

Для регистрации/авторизации была использована identity которая сама хеширует данные. Для хеширования используется алгоритм [Rfc2898](https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography.rfc2898derivebytes?view=net-5.0). В нём используется алгоритм SHA-256 с 256 битовый ключом и 128 битовая соль и 10000 итераций.

**Создание хэша**

1. Соль генерируется случайным образом с помощью функции **Rfc2898DeriveBytes,** которая генерирует хэш и соль. **Входными данными** для **Rfc2898DeriveBytes** являются пароль.
2. Затем соль и хеш смешиваются (сначала соль, затем хеш) и кодируются в виде строки (поэтому соль кодируется в хеше). Этот закодированный хэш (который содержит соль и хэш) затем сохраняется в базе данных.

**Проверка пароля по хешу**

Чтобы проверить пароль, который вводит пользователь.

1. Соль извлекается из сохраненного хешированного пароля.
2. Соль используется для хеширования **введенного** пользователем пароля с использованием перегрузки **Rfc2898DeriveBytes,** которая принимает соль вместо ее генерации.
3. Затем сохраненный хэш и тестовый хеш сравниваются.

Вывод: изучил подходы защиты данных в веб-приложении.