



The first Hub for Developers

Secure Coding with .NET

Basic Authentication

1. Authentication principles,
2. Basic Authentication
3. Role Based Access Control
4. Password Hashing, Password Storage, Vaults



# Agenda

# Authentication

Authentication is the process of verifying the identity of a user or entity.

## **Authentication Principles:**

- Identification
- Verification e.g security token.
- Authorization
- Auditing: Authentication should be audited to ensure that unauthorized access attempts are detected and logged.
- Protection of credentials
- Multi-factor authentication This can include requiring something the user knows (e.g., a password), something the user has (e.g., a security token), and something the user is (e.g., a biometric scan).

# Basic Authentication

Basic authentication is a simple HTTP-based authentication scheme that allows a user to authenticate themselves by providing a username and password. **It is the simplest way of Authentication.**

- **User credentials:** In Basic authentication, the user provides their credentials (i.e., username and password) in the HTTP headers of the request.
- **Encoding:** The username and password are encoded using Base64 encoding before being sent in the HTTP headers. However, it is important to note that this encoding method does not provide encryption or secure hashing.
- **Stateless:** Basic authentication is stateless, meaning that the server does not maintain any session or context information about the user.
- **Insecure:** Basic authentication is considered insecure because the user's credentials are sent in plain text and can be intercepted and read by third parties.
- **No logout mechanism:** Basic authentication does not provide a way for a user to log out or terminate their session. Instead, the user's credentials remain valid until the client application or browser is closed.
- **Limited functionality:** Basic authentication only provides authentication and does not support more advanced security features such as authorization or access control.

# Role Based Access Control Overview

**Role-Based Access Control (RBAC)** is a type of access control that restricts access to system resources based on the roles or job responsibilities of users within an organization. RBAC is a popular access control model used in many industries, including healthcare, finance, and government.

In RBAC, **users are assigned roles based on their job responsibilities, and permissions are assigned to each role.** Users are then granted access to system resources based on the permissions associated with their role. This approach makes it easier **to manage access control, especially in large organizations with many users and resources.**

# Role Based Access Control

**RBAC consists of several components, including:**

- **User:** A user is an individual who requires access to the system or resource.
- **Role:** A role is a collection of permissions that define the actions a user is authorized to perform within the system or on specific resources.
- **Permission:** A permission is a specific action or operation that a user is authorized to perform on a resource.
- **Resource:** A resource is an object or data that a user may access or manipulate.

# Password Hashing 101

## Password hashing

- Transforming a password into a fixed-length string of characters, called a hash, using a **cryptographic algorithm**.
- The main purpose of password hashing is to protect user passwords **in case of a security breach**.
- When a user creates a new password, the password is hashed and the hash value is stored in the database. When the user logs in, the password they provide is hashed and compared to the stored hash. If the hashes match, the user is granted access.

### VERY IMPORTANT

The hash function **should be one-way**, meaning that it should be very difficult (if not impossible) to reverse the hash and obtain the original password.

It is also common **to add a salt**, which is a **random value** added to the password before hashing, to further increase the security of the hash.

# Password Hashing in .NET

1. MD5 (System.Security.Cryptography.MD5) - **VULNERABLE**
2. SHA-1 (System.Security.Cryptography.SHA1) - **VULNERABLE**
3. SHA-256 (System.Security.Cryptography.SHA256)
4. SHA-384 (System.Security.Cryptography.SHA384)
5. SHA-512 (System.Security.Cryptography.SHA512)
6. KeyedHashAlgorithm (System.Security.Cryptography.KeyedHashAlgorithm)
7. HMACSHA1 (System.Security.Cryptography.HMACSHA1)
- 8. HMACSHA256 (System.Security.Cryptography.HMACSHA256)**
9. HMACSHA384 (System.Security.Cryptography.HMACSHA384)
10. HMACSHA512 (System.Security.Cryptography.HMACSHA512)
11. PBKDF2 (System.Security.Cryptography.Rfc2898DeriveBytes) - **SAFER**
12. Argon2 (System.Security.Cryptography.Argon2id) - **SAFER**



# Password Hashing in .NET - Code Example

```
static void Main(string[] args)
{
    // Generate a random salt
    byte[] salt = new byte[16];
    using (var rng = RandomNumberGenerator.Create())
    {
        rng.GetBytes(salt);
    }

    // Hash the password using PBKDF2
    byte[] hashedPassword = HashPassword("mypassword", salt);

    // Store the salt and hashed password in the database
    // ...

    // Verify a password
    bool passwordMatches = VerifyPassword("mypassword", salt, hashedPassword);
    Console.WriteLine(passwordMatches); // True
}
```

# Password Hashing in .NET - Code Example continued (Hashing - Verifying)

```
static byte[] HashPassword(string password, byte[] salt)
{
    const int iterations = 10000;
    const int hashSize = 32;

    using (var pbkdf2 = new Rfc2898DeriveBytes(password, salt, iterations))
    {
        return pbkdf2.GetBytes(hashSize);
    }
}

static bool VerifyPassword(string password, byte[] salt, byte[] hashedPassword)
{
    byte[] expectedHash = HashPassword(password, salt);
    return CryptographicOperations.FixedTimeEquals(expectedHash, hashedPassword);
}
```

# Password Storage in .NET

**What options are available when it comes to storing app passwords in .NET?**





- **Hashing:** Local, most popular way
- **Salted Hashing:** Similar to simple hashing but using a salt
- **Key Derivation Function (KDF):** A KDF, such as PBKDF2 or bcrypt, is a more secure way of hashing passwords. These functions perform multiple rounds of hashing with a salt to make it harder for attackers to crack passwords.
- **ASP.NET Identity:** ASP.NET Identity is a framework that provides password storage features, including hashing, salting, and KDFs. It also includes other security features such as two-factor authentication and account lockout policies.
- **Azure Key Vault:** Azure Key Vault is a cloud-based service that stores and manages cryptographic keys and secrets. It can be used to store passwords securely and can integrate with .NET applications.

# Storage Vaults that work well with .NET

All to store and manage secrets (such as passwords and API keys)

- **.NET Core Secret Manager:** used development, in a JSON file in the user's profile directory, encrypted using the Data Protection API, should not be used in production.
- **Azure Key Vault:** cloud-based service, provides features such as access control, auditing, and backup and recovery.
- **AWS Secrets Manager:** a cloud-based service, integrates with .NET apps
- **Hashicorp Vault:** an open-source tool, can be used with .NET applications through its HTTP API or client libraries.
- **KeePass:** a free and open-source password manager. KeePass can be used with .NET applications through its API or plugins.

# Basic Authentication - Starter Pack

	<b>Microsoft.AspNetCore.Authentication.JwtBearer</b> by Microsoft	6.0.5
	ASP.NET Core middleware that enables an application to receive an OpenID Conn...	
	<b>Microsoft.IdentityModel.Tokens</b> by Microsoft	6.18.0
	Includes types that provide support for SecurityTokens, Cryptographic operations:...	
	<b>Swashbuckle.AspNetCore</b> by Swashbuckle.AspNetCore	6.3.1
	Swagger tools for documenting APIs built on ASP.NET Core	
	<b>System.IdentityModel.Tokens.Jwt</b> by Microsoft	6.18.0
	Includes types that provide support for creating, serializing and validating JSON...	

Thank you!