JWT, Autenticación (AUTH) y OAuth 2.0 en C#

Introducción

La seguridad en aplicaciones web y servicios es un aspecto crítico, especialmente cuando se maneja información sensible. La autenticación y la autorización son componentes esenciales para proteger los recursos y garantizar que solo los usuarios autorizados puedan acceder a ciertos servicios. En este tema exploraremos tres conceptos clave: **JWT (JSON Web Tokens)**, **Autenticación (AUTH)** y **OAuth 2.0**. A continuación, desglosaremos cada uno de estos temas en detalle, explicando su importancia, uso y ...

---

1. ¿Qué es JWT (JSON Web Token)?

Definición

JWT (JSON Web Token) es un estándar abierto (RFC 7519) para la transmisión segura de información entre dos partes, como un cliente y un servidor, en forma de un objeto JSON. Este token se utiliza principalmente para **autenticación** y **autorización** en aplicaciones web.

A diferencia de las tradicionales sesiones de servidor, donde se guarda información del usuario en el servidor, los JWT son **autocontenidos**, lo que significa que toda la información necesaria para autenticar a un usuario se almacena dentro del token en sí, lo que facilita la escalabilidad y la seguridad.

Estructura de un JWT

Un JWT está compuesto por tres partes, separadas por puntos (`.`):

1. **Header (Encabezado)**:
    - Contiene dos partes: el tipo de token, que es "JWT", y el algoritmo de firma utilizado, como HMAC SHA256 o RSA.
    - Ejemplo:

```
{
    "alg": "HS256",
    "typ": "JWT"
}
```

2. **Payload (Cuerpo)**:
    - Contiene las **reclamaciones** (claims), que son la información que se quiere transmitir. Las reclamaciones pueden ser sobre un usuario, sus permisos, o la expiración del token.
    - Existen tres tipos de reclamaciones:
        - **Registered Claims**: Como `sub` (subject), `exp` (expiration), `iat` (issued at).

        - **Public Claims**: Pueden ser definidas por la comunidad, pero deben ser únicas.
        - **Private Claims**: Son personalizadas, creadas para compartir información entre partes que acuerdan su uso.

    - Ejemplo:

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "iat": 1516239022
}
```

3. **Signature (Firma)**:

    - La firma se crea combinando el encabezado y el cuerpo, y firmándolos con un **secreto** o una clave privada.
    - Ejemplo de generación de la firma en pseudocódigo:

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Ejemplo de un JWT completo:

```
Header:
{
  "alg": "HS256",
  "typ": "JWT"
}

Payload:
{
  "sub": "1234567890",
  "name": "John Doe",
  "iat": 1516239022
}

Signature:
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

Uso de JWT en C#

Para trabajar con JWT en C#, se puede usar la biblioteca `System.IdentityModel.Tokens.Jwt`. A continuación se muestra un ejemplo de cómo crear y verificar un JWT en C#.

Ejemplo de creación de un JWT en C#:

```
public string CreateJWT(string username)
{
    var claims = new[]
    {
        new Claim(ClaimTypes.Name, username),
        new Claim(ClaimTypes.Role, "User")
    };

    var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("mi_clave_secreta"));
    var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

    var token = new JwtSecurityToken(
        issuer: "mi_api",
        audience: "mi_cliente",
        claims: claims,
        expires: DateTime.Now.AddMinutes(30),
        signingCredentials: credentials
    );

    var tokenHandler = new JwtSecurityTokenHandler();
    return tokenHandler.WriteToken(token);
}
```

Validación de JWT

Para validar un JWT, se verifica su firma y los valores de las reclamaciones, como la expiración (`exp`), el emisor (`iss`) y la audiencia (`aud`).

Ejemplo de validación de JWT en C#:

```
public ClaimsPrincipal ValidateJWT(string token)
{
    var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("mi_clave_secreta"));
    var tokenHandler = new JwtSecurityTokenHandler();

    try
    {
        var principal = tokenHandler.ValidateToken(token, new
TokenValidationParameters
        {
            ValidateIssuer = true,
```

```
            ValidateAudience = true,
            ValidateLifetime = true,
            ValidIssuer = "mi_api",
            ValidAudience = "mi_cliente",
            IssuerSigningKey = securityKey
        }, out SecurityToken validatedToken);

        return principal;
    }
    catch (Exception)
    {
        return null; // Token inválido
    }
}
```

---

2. Autenticación (AUTH)

Definición

La **autenticación** es el proceso mediante el cual una aplicación verifica la identidad de un usuario. En términos sencillos, es la forma en que el sistema se asegura de que quien está accediendo es quien dice ser.

Existen diferentes tipos de autenticación, como la autenticación básica (usuario y contraseña), la autenticación basada en tokens (como JWT) y métodos más avanzados como la autenticación multifactor (MFA).

Tipos de Autenticación

1. **Autenticación Básica**: Se basa en enviar las credenciales (usuario y contraseña) en cada solicitud HTTP, generalmente utilizando la cabecera `Authorization` con el formato `Basic base64(usuario:contraseña)`.
    - **Problema**: Aunque es fácil de implementar, no es segura si no se utiliza HTTPS.
2. **Autenticación con JWT**: El servidor autentica al usuario y le devuelve un token JWT. Este token se utiliza en futuras solicitudes para acceder a recursos protegidos, sin necesidad de enviar las credenciales nuevamente.
3. **Autenticación Multifactor (MFA)**: Es un sistema más seguro que requiere al menos dos factores de autenticación, como algo que el usuario sabe (contraseña), algo que el usuario tiene (un código enviado al móvil) o algo que el usuario es (biometría).

Ejemplo de autenticación con JWT:

```
public IActionResult Login([FromBody] LoginModel login)
{
    if (IsValidUser(login))
    {
        var claims = new[]
```

```
        {
            new Claim(ClaimTypes.Name, login.Username),
            new Claim(ClaimTypes.Role, "User")
        };

        var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("mi_clave_secreta"));
        var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);

        var token = new JwtSecurityToken(
            issuer: "mi_api",
            audience: "mi_cliente",
            claims: claims,
            expires: DateTime.Now.AddMinutes(30),
            signingCredentials: credentials
        );

        var tokenHandler = new JwtSecurityTokenHandler();
        return Ok(new { token = tokenHandler.WriteToken(token) });
    }

    return Unauthorized();
}
```

---

3. OAuth 2.0

Definición

OAuth 2.0 es un protocolo de autorización que permite a las aplicaciones obtener acceso limitado a los recursos de un usuario sin compartir sus credenciales. OAuth 2.0 se utiliza en escenarios en los que una aplicación necesita acceder a los recursos de un usuario en otro servicio.

Flujos de OAuth 2.0

1. **Authorization Code Grant**: El flujo más seguro, recomendado para aplicaciones web. El cliente obtiene un **código de autorización** que luego intercambia por un token de acceso.
2. **Implicit Grant**: Utilizado para aplicaciones del lado del cliente (como aplicaciones JavaScript o SPA), donde el token de acceso se emite directamente sin necesidad de un código de autorización.
3. **Client Credentials Grant**: Utilizado por aplicaciones que acceden a sus propios recursos sin necesidad de que un usuario esté involucrado, como los servicios backend.
4. **Password Credentials Grant**: El usuario proporciona directamente su nombre de usuario y contraseña para obtener un token de acceso. Este flujo es menos seguro y generalmente no recomendado.

Ejemplo de implementación de flujo de autorización (Authorization Code Grant):

```
public async Task<IActionResult> Authorize()
{
    var client = new HttpClient();
    var request = new HttpRequestMessage(HttpMethod.Post, "https://authorization-
server.com/token");

    request.Content = new FormUrlEncodedContent(new Dictionary<string, string>
    {
        { "grant_type", "authorization_code" },
        { "code", "authorization_code_received" },
        { "redirect_uri", "https://myapp.com/callback" },
        { "client_id", "your_client_id" },
        { "client_secret", "your_client_secret" }
    });

    var response = await client.SendAsync(request);
    var content = await response.Content.ReadAsStringAsync();

    return Ok(content);
}
```

4. Taller Práctico

Ejercicio 1: Creación de JWT

Desarrolla una API en C# que reciba un nombre de usuario y devuelva un JWT con la información del usuario. El JWT debe incluir el rol de usuario y la fecha de expiración.

Ejercicio 2: Implementación de Autenticación Básica

Implementa un sistema de autenticación básica que reciba las credenciales del usuario y devuelva un mensaje de bienvenida si las credenciales son válidas.

Ejercicio 3: Implementación de OAuth 2.0

Crea una aplicación cliente que implemente el flujo de **Authorization Code Grant** para obtener un token de acceso y acceder a un servicio protegido.

Ejercicio 4: Validación de JWT

Desarrolla un middleware en C# que valide un JWT en cada solicitud HTTP antes de permitir el acceso a recursos protegidos.

Ejercicio 5: Comparación de Flujos de OAuth 2.0

Implementa los distintos flujos de OAuth 2.0 y compara su funcionamiento, analizando ventajas y desventajas de cada uno.