OAuth2 Token Management for asp.net core 3.x

Herb Stahl
Sep 25 · 4 min read

github source code.

The following are requirements I have for how my OAuth2 tokens are managed in my asp.net core application;

1. The tokens will be stored in a distributed cache
2. The tokens will be encrypted at rest, so anyone getting access to the cache can't harvest any of them
3. The tokens, when they are session based, will be protected with a unique session key
4. The tokens can be refreshed, whatever that means.

Calling the refresh_token endpoint or calling a client_credentials flow when needed. I also have a need for more complex flows where a refreshing is really a token exchange flow that makes a call to the OAuth2 token endpoint with an custom extension grant.

5. Tokens can be managed at the app level and session level. When a new user logs in a new session key is generated in my case.

6. The Token Management Service is going to be offered as injectable interfaces.

Technology Used

Data Protection is to be provided by **IDataProtectionProvider**. This is a sweet service where I let them handle the symmetric encryption stuff. I use it for protecting cookies and anything I want protected in a cache.

Any calls I make to the OAuth2 endpoints will be done using **IdentityModel.** I use this for getting the discovery document as well as making calls to the refresh_token and token endpoints.

My caching is going to use **IDistributedCache** so its up to me to protect the data before I put anything in there.

Session Based Protection

The first thing I do when a user is logged in is to create and store a new protection key in the session. GuidGen is just fine for the unique protection key. Once I have this key, I then use **IDataProtectionProvider** to create an **IDataProtector** using the session key.

Another option is to store that data in the session itself and if you have a session provider that is implemented to protect everything then you don't need the IDataProtectionProvider stuff. I am assuming that my distributed session cache isn't going to be fully encrypted.

App Based Protection

Similar to the Session Based Protection, here we have a configured string that we use to create an **IDataProtector**.

OAuth2 Credential Manager

I want a service that will serve up OAuth2 credentials by name. I will be registering all the credentials that are in my configuration. This is a convenience service where all I am doing is storing the credentials in a local memory cache. Given an OAuth2 Authority and client_id/client_secret the OAuth2 Credential Manager will cache the discovery response and everything it needs for the OAuth2 calls.

A little history on this one is that your app will probably need to interact with multiple OAuth2 services because the downstream services tend to use their own private OAuth2 services.

CustomTokenRequest

This service is to acknowledge that there is no one way to request a token. Even for client_credentials flow, a request may differ simply because the scopes being requested.

I have an OAuth2 service that implemented a custom extension grant.

I have flows where getting a new token is a complex token exchange flow that involves multiple OAuth2 services.

```
public interface ICustomTokenRequestManager
{
        void AddTokenRequestFunction(string key, Func<ManagedToken,
IServiceProvider,IOAuth2CredentialManager, CancellationToken,
Task<ManagedToken>> func);
        Func<ManagedToken, IServiceProvider,IOAuth2CredentialManager,
CancellationToken, Task<ManagedToken>> GetTokenRequestFunc(string
key);
}
```

Here I can register a bunch of custom OAuth2 functions that do everything from a simple client_credentials flow, refresh_token flow, or a custom extension grant flow.

This is just a Map of functions.

Below is an example of a simple client_credentials flow call.

```
static async Task<ManagedToken> ExecuteClientCredentialsRequestAsync(
           ManagedToken managedToken,
           IServiceProvider serviceProvider,
           IOAuth2CredentialManager oAuth2CredentialManager,
           CancellationToken cancellationToken = default)
        {
            var creds = await
oAuth2CredentialManager.GetOAuth2CredentialsAsync(managedToken.Creden
tialsKey);
            var client = new HttpClient();
            var response = await
client.RequestClientCredentialsTokenAsync(
                new ClientCredentialsTokenRequest
                {
                    Address =
creds.DiscoveryDocumentResponse.TokenEndpoint,
                    ClientId = creds.ClientId,
                    ClientSecret = creds.ClientSecret,
                    Scope = managedToken.RequestedScope
                },
                cancellationToken);
            if (response.IsError)
            {
                throw new Exception(response.Error);
            }
            managedToken.RefreshToken = response.RefreshToken;
            managedToken.AccessToken = response.AccessToken;
            managedToken.ExpiresIn = response.ExpiresIn;
            return managedToken;
        }
```

Usage

Registering the services;

```
services.AddManagedTokenServices();
```

Registering the client credentials;

The following does this in my `Startup.Configure();`

But basically you can do this from anywhere, it just depends on where you are pulling your secrets from. Please use `Azure KeyVault` if you are hosting there.

```
var oAuth2CredentialManager =
serviceProvider.GetRequiredService<IOAuth2CredentialManager>();

// Register the credentials for my test OAuth2 service
oAuth2CredentialManager.AddCredentialsAsync("test", new
OAuth2Credentials
            {
                Authority = "https://demo.identityserver.io",
                ClientId = "m2m",
                ClientSecret = "secret"
            }).GetAwaiter().GetResult();
```

Registering the Managed Token;

The following is a global registration. The following token is under management for the lifetime of the application.

```
// Next register the token request.  In this case its a simple
client_credentials call
var globalTokenManager = serviceProvider
.GetRequiredService<ITokenManager<GlobalDistributedCacheTokenStorage>
>();
globalTokenManager.AddManagedTokenAsync("test", new ManagedToken
            {
                CredentialsKey = "test",
                RequestFunctionKey = "client_credentials",
                RequestedScope = null // everything
            }).GetAwaiter().GetResult();
```

The following is for a session. Once the session is gone, so go all the tokens under management there.

```
var sessionTokenManager = serviceProvider
.GetRequiredService<ITokenManager<SessionDistributedCacheTokenStorage
>>();
```

Get and use the tokens;

Inject the following service into your other services

```
ITokenManager<GlobalDistributedCacheTokenStorage> globalTokenManager
```

Get the token

```
ManagedToken = await _globalTokenManager.GetManagedTokenAsync("test",
true);
```

Service Method;

```
public interface ITokenManager<T> where T : TokenStorage
{
  Task<ManagedToken> AddManagedTokenAsync(string key, ManagedToken
tokenConfig);
  Task RemoveManagedTokenAsync(string key);
  Task<ManagedToken> GetManagedTokenAsync(string key, bool
forceRefresh = false, CancellationToken cancellationToken = default);
  Task RemoveAllManagedTokenAsync();
}
```

Aspnetcore   Oauth2   Openid Connect   Authentication