Projektowanie aplikacji ASP.NET
Wykłady 04-05/15
ASP.NET Autentykacja, autoryzacja

Wiktor Zychla 2019/2020

Spis treści

1 Wprowadzenie 2
2 Autentykacja, autoryzacja 3
2.1 Autentykacja 302 3
2.2 Autentykacja 401 4
3 Model dostawców (Provider Model) 5
4 Uwierzytelnianie federacyjne, protokoły Single Sign-on 7
4.1 Protokół WS-Federation 7
4.2 Protokół OAuth2 8
4.3 Przykład 9

1 Wprowadzenie

Autentykacja = proces rozpoznania tożsamości użytkownika

Autoryzacja = proces decyzyjny w którym użytkownikowi przyznaje się dostęp do zasobów lub zabrania się dostępu do zasobów

W praktyce, upraszczając, można powiedzieć że autentykacja jest *jakoś* związana z logowaniem, natomiast autoryzacja pozwala sterować dostępem do zasobów (np. „brak dostępu dla niezalogowanych” lub „dostęp tylko dla użytkowników w roli administratorzy” itp.)

2 Autentykacja, autoryzacja

Są dwa podstawowe typy autentykacji

- Oparte o status 401, wbudowane w protokół HTTP
- Oparte o status 302, związane z przekierowaniem na dodatkowy zasób ustalający tożsamość użytkownika (np. stronę logowania)
    - 302 do strony w ramach tej samej witryny, z możliwością współdzielenia stanu (np. ciastka)
    - 302 do strony w ramach innej witryny bez możliwości współdzielenia stanu

Od strony kodu, w aplikacji, pomysł na uwierzytelnianie żądania polega na skojarzeniu z żądaniem obiektu typu **IPrinicipal** który reprezentuje użytkownika – właściciela bieżącego żądania. Obiekt ten może reprezentować użytkownika nie mającego odpowiednika w systemie operacyjnym, dzięki czemu możliwe jest tworzenie aplikacji z *wirtualnymi* rejestrami użytkowników (na przykład przechowywanymi w bazie danych). Taki **IPrincipal** skojarzony z żądaniem jest więc czymś zupełnie innym niż tożsamość użytkownika – właściciela procesu wykonującego kod po stronie serwera.

Pozyskanie tożsamości właściciela procesu:

```
var user = System.Security.Principal.WindowsIdentity.GetCurrent().Name;
```

Pozyskanie wirtualnego użytkownika – właściciela bieżącego żądania

```
var principal = HttpContext.Current.User;
```

W domyślnej aplikacji, w której nie skonfigurowano żadnego uwierzytelniania:

- Właściciel procesu zawsze jest niepusty (każdy proces w systemie ma właściciela) – jest to konto ustawione na serwerze aplikacyjnym jako właściciel procesu
- Właściciel bieżącego żądania jest pusty – żadna logika nie wymaga od użytkowników bycia „zalogowanym”

2.1 Autentykacja 302

Najprostsza poprawnie zrealizowana autentykacja typu „przekierowanie 302” opiera się podtrzymywanie stanu sesji zalogowanego użytkownika w **ciastku.** Za wydawanie i kontrolę ciastka odpowiada **moduł autentykacji**, może to być wbudowany moduł **FormsAuthenticationModule**. Ciastka powinny być szyfrowane/podpisane, aby uniemożliwić sfałszowanie ich zawartości po stronie przeglądarki.

Konfiguracja modułu Forms w potoku przetwarzania wymaga wyłącznie sekcji **authentication** w pliku konfiguracyjnym. Dodatkowo, wymagane są reguły autoryzacji aby moduł mógł wymusić przekierowanie nieautoryzowanego żądania do strony logowania.

```
  <system.web>
    <authentication mode="Forms">
```

```
      <forms loginUrl="LoginPage.aspx" defaultUrl="WebForm1.aspx" />
    </authentication>
    <authorization>
      <deny users="?"/>
      <allow users="*"/>
    </authorization>
```

Moduł **FormsAuthentication** pozwala na

- Automatyczne wydanie ciastka zalogowanemu użytkownikowi i przekierowanie na stronę wskazywaną na stronie logowania przez parametr **ReturnUrl**

```
string username;
FormsAuthentication.RedirectFromLoginPage(username, false);
```

- Pełną kontrolę nad ważnością sesji zapisanej w ciastku autentykacji

```
FormsAuthenticationTicket ticket = new FormsAuthenticationTicket(
    1, username,
    DateTime.Now, DateTime.Now.AddMinutes(20),
    false, string.Empty);

HttpCookie cookie = new HttpCookie(FormsAuthentication.FormsCookieName);
cookie.Value = FormsAuthentication.Encrypt(ticket);

HttpContext.Current.Response.AppendCookie(cookie);
```

2.2 Autentykacja 401

Autentykacja zintegrowana wymaga przełączenia trybu autentykacji w pliku konfiguracyjnym na **Windows**

```
  <system.web>
    <authentication mode="Windows" />
    <authorization>
      <deny users="?"/>
      <allow users="*"/>
    </authorization>
```

W tym trybie sesja użytkownika nie jest podtrzymywana ciastkami i zależy od jednego z obsługiwanych przez przeglądarki protokołów. W szczególności:

- Dla protokołu NTLM sesja negocjowana jest „per połączenie" (per socket)
- Dla protokołu Kerberos sesja podtrzymywana jest w nagłówku **Authorization**

3 Model dostawców (Provider Model)

W celu oddzielenia warstwy aplikacji odpowiadającej za techniczną stronę autentykacji (302/401) od samego procesu uwierzytelniania, zaproponowano tzw. model dostawców (Provider Model), w którym elementy aplikacji odpowiedzialne za

- Uwierzytelnianie
- Autoryzację
- Ale też - zarządzanie sesją, itd.

mają swoje abstrakcje (klasy abstrakcyjne) a zadaniem programisty jest dostarczenie implementacji, np.:

```
public class MyMembershipProvider : MembershipProvider
{
    public MyMembershipProvider()
    {
    }

    …

    public override bool ValidateUser(string username, string password)
    {
        if (username == password)
            return true;

        return false;
    }
}
```

dla autentykacji oraz

```
public class MyRoleProvider : RoleProvider
{
    public MyRoleProvider()
    {
    }

    …

    public override string[] GetRolesForUser(string username)
    {
        return new string[] { username };
    }
}
```

Rejestracja dostawców wymaga wyłącznie odpowiedniej konfiguracji:

```
    <membership defaultProvider ="MyMembershipProvider">
      <providers>
        <add name="MyMembershipProvider" type="MyMembershipProvider"/>
      </providers>
    </membership>
    <roleManager enabled ="true" defaultProvider="MyRoleProvider">
      <providers>
```

```
        <add name="MyRoleProvider" type="MyRoleProvider"/>
      </providers>
    </roleManager>
```

i od tego momentu możliwe jest posługiwanie się w aplikacji fasadami odpowiadającymi dostawcom, np.: **Membership**, czy **Roles.**

Autoryzacja w modelu RBS (Role-based Security) możliwa jest na trzy sposoby:

1. Zapamiętanie w ciastku Forms wyłącznie nazwy użytkownika, odczytywanie ról za każdym żądaniem
2. Zapamiętanie w ciastku Forms ról w sekcji UserData
3. Zapamiętanie ról w dodatkowym ciastku (wymaga ustawienia wartości **true** atrybutu **cacheRolesInCookie** węzła **roleManager**

4 Uwierzytelnianie federacyjne, protokoły Single Sign-on

Uwierzytelnianie za pomocą zewnętrznego dostawcy możliwe jest wyłącznie przy zapewnieniu bezpieczeństwa, w szczególności braku możliwości oszukania przepływu kontroli między dwoma różnymi aplikacjami przez użytkownika.

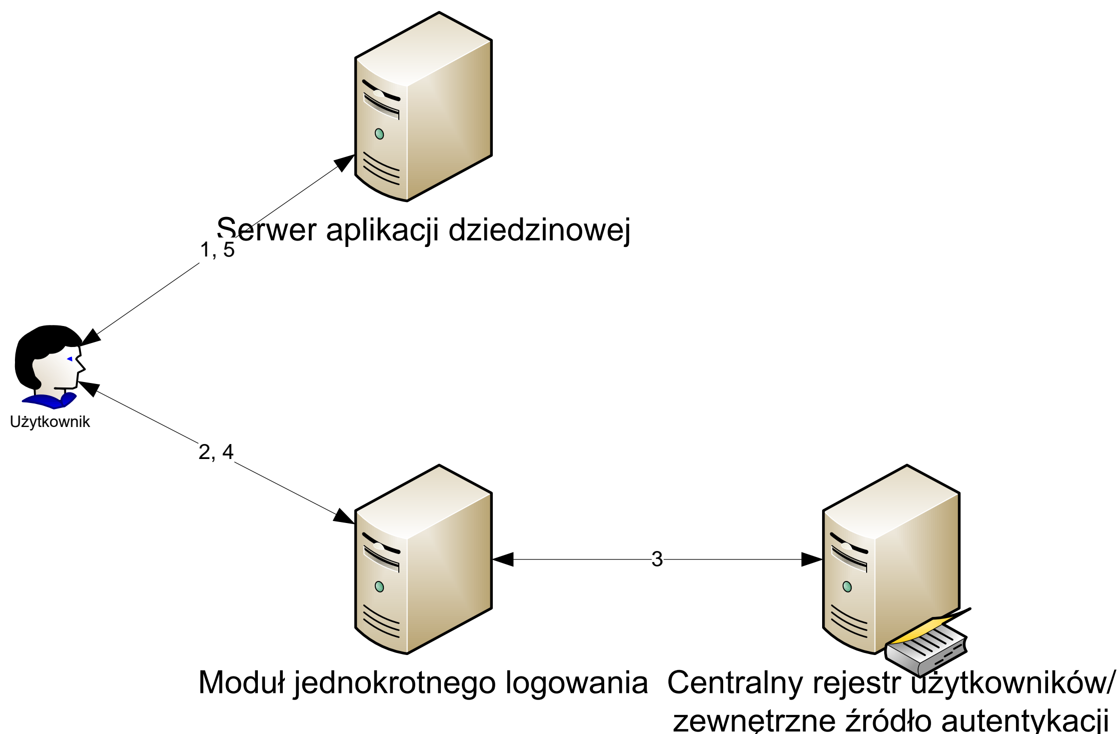
Z tego powodu współcześnie korzysta się z tzw. protokołów SSO, np.:

- protokół **passive** [WS-Federation], który definiuje przepływ komunikatów dla klienta pasywnego (przeglądarka internetowa) i umożliwia uzyskanie poświadczonej przez serwer informacji o tożsamości użytkownika i jego przynależności do ról (tu: grup zabezpieczeń). Protokół należy do rodziny WS-* i jest uznanym, przyjętym powszechnie w przemyśle rozwiązaniem, dla którego istnieją gotowe implementacje części klienckich i serwerowych dla różnych platform technologicznych – w przypadku systemu heterogenicznego jest to duża zaleta, otwierająca perspektywę łatwej rozbudowy systemu o kolejne moduły w przyszłości.
- Protokół [OAuth2/OpenID Connect], szeroko implementowany przez dostawców usług społecznościowych

Na potrzeby każdego wdrożenia systemu identyfikuje się podsystem nazywany dalej **modułem jednokrotnego logowania**, który w nomenklaturze technicznej jest dostawcą tożsamości (security token service, identity provider) protokołu pojedynczego logowania.

4.1 Protokół WS-Federation

Rysunek 1 przedstawia schemat poświadczania tożsamości przy wykorzystaniu WS-Federation i modułu jednokrotnego logowania.



Rysunek 1 Poświadczanie tożsamości przy wykorzystaniu WS-Federation i dostawcy tożsamości

Poszczególne kroki protokołu przedstawiają się następująco:

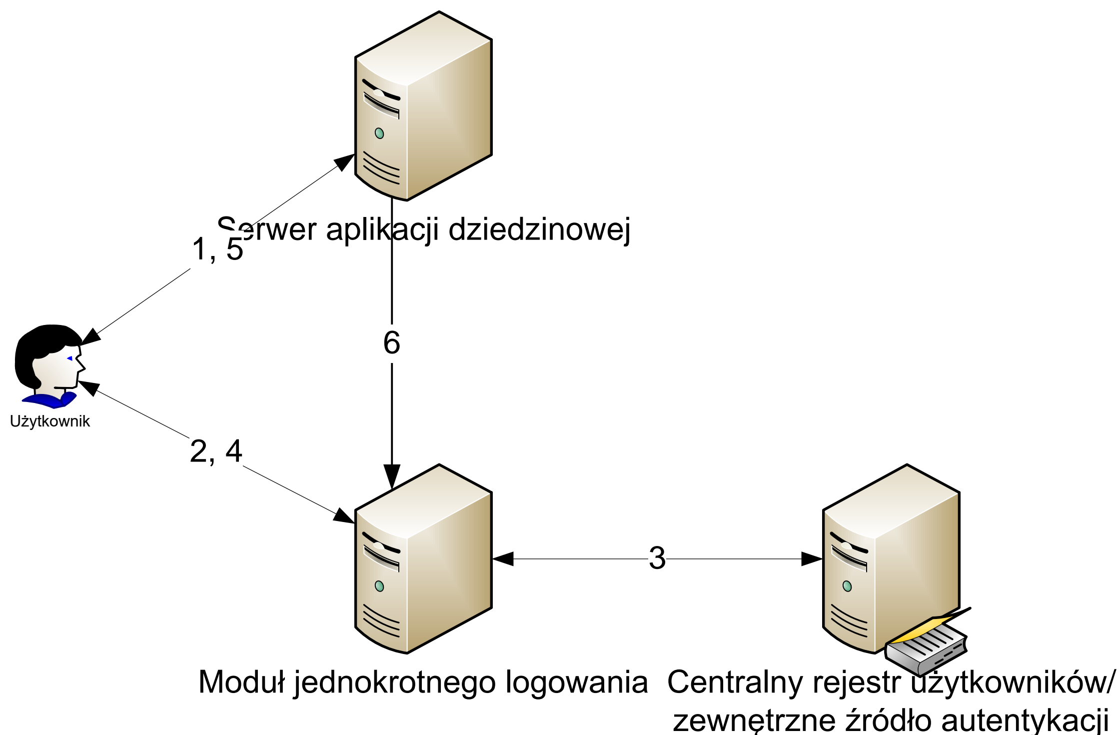
1. Użytkownik kieruje żądanie do wybranego serwera aplikacji obsługującego jeden z modułów systemu
2. Jeśli moduł do tej pory nie przeprowadził autentykacji tego użytkownika, za pośrednictwem przeglądarki kierowane jest żądanie wydania informacji o użytkowniku do serwera modułu jednokrotnego logowania
3. Serwer jednokrotnego logowania poświadcza tożsamość użytkownika, samodzielnie lub delegując autentykację dalej, do zaufanego dostawcy.
4. Serwer jednokrotnego logowania tworzy tzw. *token bezpieczeństwa* użytkownika zgodny ze standardem SAML, zawierający atrybuty opisujące użytkownika (**nazwa logowania, imię, nazwisko, unikalny identyfikator, adres e-mail i przynależność do grup zabezpieczeń**).
5. Serwer jednokrotnego logowania **podpisuje** token bezpieczeństwa, uniemożliwiając w ten sposób jego zafałszowanie i poświadczając jego wiarygodność i za pośrednictwem przeglądarki odsyła informację do właściwego serwera aplikacji. Token bezpieczeństwa (właściwie: token SAML) ma postać dokumentu XML.
6. Serwer aplikacji waliduje integralność przedstawionego tokenu bezpieczeństwa i przydziela użytkownikowi dostęp do właściwych zasobów w ramach zawartej w tokenie bezpieczeństwa informacji o przynależności użytkownika do grup zabezpieczeń

Szczegółowa dokumentacja techniczna protokołu autentykacji WS-Federation, zawartości i sposobu interpretacji tokenów SAML są publicznie dostępne i nie zostaną dołączone do niniejszego opracowania.

Należy zwrócić uwagę, że jedną z pożądanych właściwości specyfikacji WS-Federation jest obsługa scenariusza Single Sign-out, czyli możliwość wylogowania się użytkownika z całego środowiska aplikacyjnego przez jeden wspólny odnośnik. Technicznie realizowane jest to następująco – podczas autentykacji użytkowników na potrzeby konkretnych aplikacji (krok 3) serwer jednokrotnego logowania w sesji użytkownika zapamiętuje odnośniki do tych aplikacji. W ten sposób w każdym momencie serwer jednokrotnego logowania wie do których aplikacji użytkownik jest zalogowany za jego pośrednictwem. Wylogowanie sprowadza się do wygenerowania spreparowanej strony z odnośnikami do poszczególnych aplikacji z dołączonym specjalnym parametrem, który dla aplikacji jest równoznaczny z poleceniem wylogowania się.

4.2 Protokół OAuth2

Rysunek 2 przedstawia schemat poświadczania tożsamości przy wykorzystaniu OAuth2 i modułu jednokrotnego logowania.



Rysunek 2 Poświadczanie tożsamości przy wykorzystaniu OAuth2 i dostawcy tożsamości

Poszczególne kroki protokołu przedstawiają się następująco:

1. Użytkownik kieruje żądanie do wybranego serwera aplikacji obsługującego jeden z modułów systemu
2. Jeśli moduł do tej pory nie przeprowadził autentykacji tego użytkownika, za pośrednictwem przeglądarki kierowane jest żądanie wydania informacji o użytkowniku do serwera modułu jednokrotnego logowania
3. Serwer jednokrotnego logowania poświadcza tożsamość użytkownika, samodzielnie lub delegując autentykację dalej, do zaufanego dostawcy.
4. Serwer jednokrotnego logowania tworzy tzw. *jednokrotny kod bezpieczeństwa*
5. Serwer aplikacji zamienia jednokrotny kod bezpieczeństwa na tzw. *token bezpieczeństwa*, którego następnie używa do uzyskania informacji o użytkowniku (**nazwa logowania, imię, nazwisko, unikalny identyfikator, adres e-mail i przynależność do grup zabezpieczeń**) w module jednokrotnego logowania

4.3 Przykład

W trakcie wykładu zobaczymy przykład użycia dostawcy (Google) do wykonania integracji logowania za pomocą protokołu OAuth2 i biblioteki **DotnetOpenAuth**. Uwaga na użyte w kodzie stałe reprezentujące adresy punktów końcowych usługi OAuth2 – Google aktualizuje te adresy regularnie, aktualny wypis znajduje się na adresie typu discovery, https://accounts.google.com/.well-known/openid-configuration.

accounts.google.com/.well-known/openid-configuration


```
{
 "issuer": "https://accounts.google.com",
 "authorization_endpoint": "https://accounts.google.com/o/oauth2/v2/auth",
 "token_endpoint": "https://oauth2.googleapis.com/token",
 "userinfo_endpoint": "https://openidconnect.googleapis.com/v1/userinfo",
 "revocation_endpoint": "https://oauth2.googleapis.com/revoke",
 "jwks_uri": "https://www.googleapis.com/oauth2/v3/certs",
 "response_types_supported": [
  "code",
  "token",
  "id_token",
  "code token",
  "code id_token",
  "token id_token",
  "code token id_token",
  "none"
 ],
 "subject_types_supported": [
  "public"
 ],
 "id_token_signing_alg_values_supported": [
  "RS256"
 ],
 "scopes_supported": [
  "openid",
  "email",
  "profile"
 ],
 "token_endpoint_auth_methods_supported": [
```


```
public class GoogleClient : WebServerClient
{
    private static readonly AuthorizationServerDescription GoogleDescription =
        new AuthorizationServerDescription
        {
            TokenEndpoint         = new Uri( "https://accounts.google.com/o/oauth2/token" ),
            AuthorizationEndpoint = new Uri( "https://accounts.google.com/o/oauth2/auth" ),
            ProtocolVersion       = ProtocolVersion.V20,
        };

    public const string ProfileEndpoint =
"https://openidconnect.googleapis.com/v1/userinfo";

    public const string OpenId       = "openid";
    public const string ProfileScope = "profile";
    public const string EmailScope   = "email";

    public GoogleClient()
        : base( GoogleDescription )
    {
    }
}

public class GoogleProfileAPI
{
    public string email { get; set; }
    public string given_name { get; set; }
    public string family_name { get; set; }
```

```
    private static DataContractJsonSerializer jsonSerializer =
        new DataContractJsonSerializer( typeof( GoogleProfileAPI ) );

    public static GoogleProfileAPI Deserialize( Stream jsonStream )
    {
        try
        {
            if ( jsonStream == null )
            {
                throw new ArgumentNullException( "jsonStream" );
            }

            return (GoogleProfileAPI)jsonSerializer.ReadObject( jsonStream );
        }
        catch ( Exception ex )
        {
            return new GoogleProfileAPI();
        }
    }
}

public class MyAuthorizationTracker : IClientAuthorizationTracker
{

    public IAuthorizationState GetAuthorizationState(
        Uri callbackUrl,
        string clientState )
    {
        return new AuthorizationState
        {
            Callback = new Uri( callbackUrl.GetLeftPart( UriPartial.Path ) )
        };
    }

}
```

Sam przepływ autentykacji wymaga modyfikacji kodu strony logowania, gdzie odbywa się delegowanie uwierzytelnienia

```
protected void Page_Load( object sender, EventArgs e )
 {
     IAuthorizationState authorization = gClient.ProcessUserAuthorization();
     // Is this a response from the Identity Provider
     if ( authorization == null )
     {
         // no

         // Google will redirect back here
         Uri uri = new Uri( "http://localhost:62889/LoginPage.aspx" );
         // Kick off authorization request with OAuth2 scopes
         gClient.RequestUserAuthorization( returnTo: uri,
             scope: new[] { GoogleClient.OpenId, GoogleClient.ProfileScope,
GoogleClient.EmailScope } );
     }
     else
     {
```

```
        // yes

        var request = WebRequest.Create( GoogleClient.ProfileEndpoint );
        // add an OAuth2 authorization header
        // if you get 403 here, turn ON Google+ API on your app settings page
        request.Headers.Add(
             HttpRequestHeader.Authorization,
             string.Format( "Bearer {0}", Uri.EscapeDataString( authorization.AccessToken )
) );
        // Go to the profile API
        using ( var response = request.GetResponse() )
        {
            using ( var responseStream = response.GetResponseStream() )
            {
                var profile = GoogleProfileAPI.Deserialize( responseStream );
                if ( profile != null &&
                    !string.IsNullOrEmpty( profile.email ) )
                    FormsAuthentication.RedirectFromLoginPage( profile.email, false );
            }
        }
    }
}

public readonly GoogleClient gClient = new GoogleClient
{
    AuthorizationTracker        = new MyAuthorizationTracker(),
    ClientIdentifier            = "my client id",
    ClientCredentialApplicator = ClientCredentialApplicator.PostParameter( "my client
secret" )
};
```