OBJECT: Various places to defining security key.

Introduction

In web applications, security is essential. Say that the user wants to use the resources of our system. For that, we need to authenticate the user. Authentication means need to check whether the user is eligible to use the system or not. Generally, we do authentication via username (in the form of a unique user name or email ID) and a password. If the user is authenticated successfully, then we allow that user to use the resources of our system.

But what about subsequent requests? If the user has been already identified then s/he does not need to provide credentials each time. Once authenticated, for a particular period s/he can use the system's resources. In a traditional approach, we used to save Username in Session. This session period is configurable, which means the session is valid for about 15 or 20 minutes. This session is stored in server's memory. After expiration of the session, the user needs to login again.

But here, there are couple of problems.

1. The session can be hijacked.
2. If we have multiple instances of server with load balancer, then if the request goes to a server other than the server which has authenticated the earlier request, then it will invalidate that session. Because the session is not distributed among all the servers, we have to use a 'Sticky' session; that is we need to send each subsequent request to the same server only. Here, we can also store session in database instead of the server's memory. In that case, we need to query the database each time, and that's extra work which may increase the overall latency.

To solve this problem, we can do authentication via JWT i.e. JSON web token. After successful authentication, the server will generate a security token and send it back to the client. This token can be generated using a symmetric key algorithm or an asymmetric key algorithm. On each subsequent request after a successful login, the client will send a generated token back to the server. The server will check whether the sent token is valid or not and also checks whether its expired or not. The client will send this token in Authentication Bearer header.

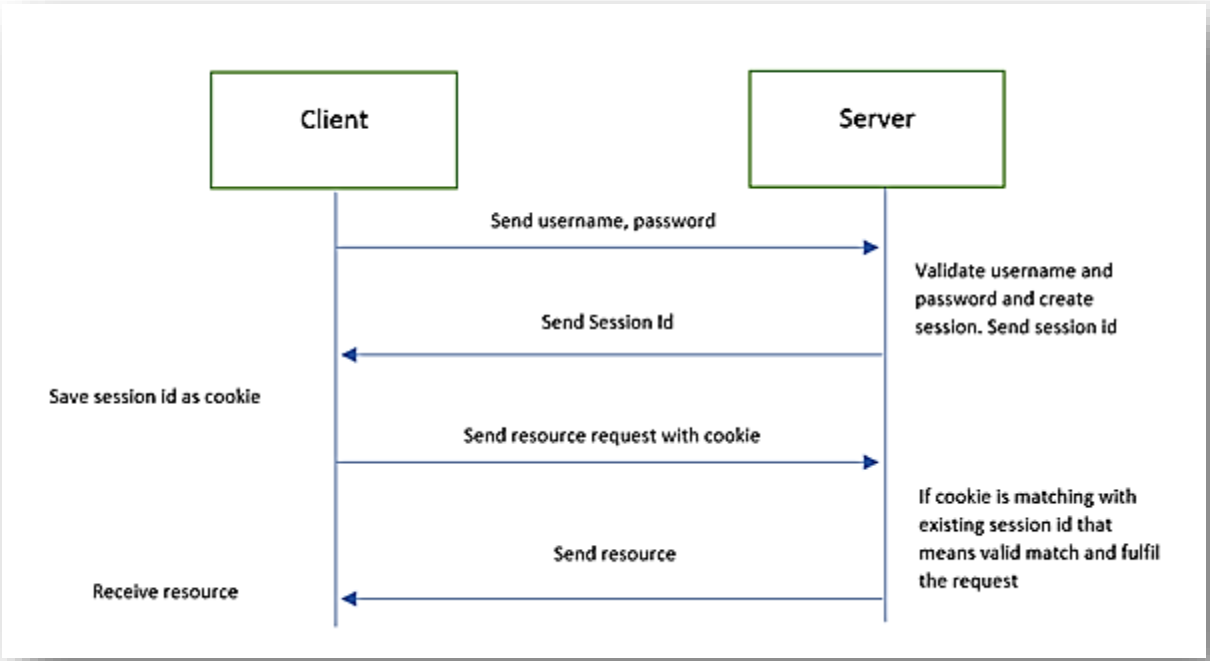
You can see cookie-based authentication requires a server to keep track of all active sessions, this is known as “stateful” sessions. Once the session id is lost or destroyed, a new session needs to be created. Imagine if a client is dealing with multiple resource servers? In this case, that client needs to remember the cookie from each server and keep track of it.

The token-based method overcomes the shortcomings of cookie-based authentication.

In token-based authentication, a client is given token instead of a cookie. The tokens are light-weight JSON (JavaScript Object Notation) and contain encoded information about the user and expiry time.

Benefit

The server only needs to validate the incoming request token instead of storing sessions for each client. So, it is the best fit for “stateless” implementation of services, which is a good fit for RESTful WebAPI.



JWT token has a particular format. Header, Payload, and Signature.

1. **Header**
   - We need to specify which token system we want to use and also need to specify the algorithm type.

2. **Payload**
   - This is a token body. Basically, it contains expiry detail, claims details, issuer detail, etc.

3. **Signature**
   - To create the signature part we have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

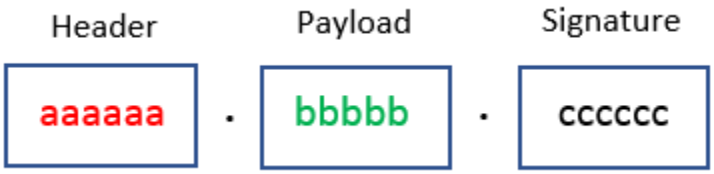
E.g. HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secret)

Benefits of using JWT

1. JSON parser is common in programming languages.
2. Secure. We can use a Symmetric or Asymmetric key algorithm.
3. Less verbose in comparison to SAML.

JWT (JSON Web Token) is a very common format of token-based implementation. It is so popular right now that it has become a de-facto standard for token-based authentication.

JWT is composed of three components, separated by a dot (.)



Header

Header contains standard information, i.e., type of token and the name of the algorithm. The information is coded in base64 format.

Example -

```
1. {
2.    "alg": "HS256",
3.    "typ": "JWT"
4. }
```

Payload

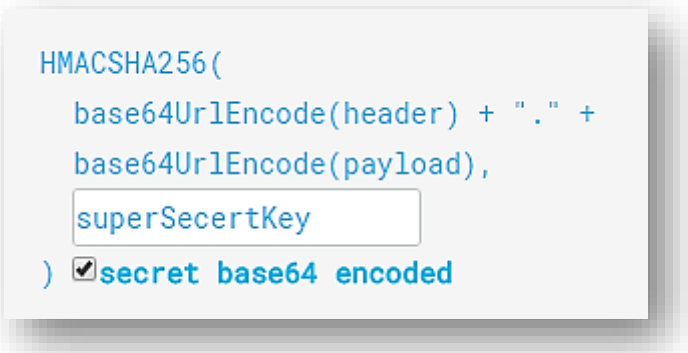
Payload is JSON data that contains information of the user. It can but does not have to be limited to the data regarding user, claims and any other necessary data can also be there.

Example -

```
1. {
2.    “issuer”: “http://www.exampleweb.com”,
3.    “expires”: “2015-11-18T18:25:43.511Z”
4. }
```

Signature

Signature is a “digital signature” of the combination of header and payload. Signature helps the server to verify the authenticity of the content of JWT against malicious alteration of content.



DEFINING THE KEY IN VARIOUS PLACES

1. In start-up file

Startup.cs

```
public void ConfigureServices(IServiceCollection services)
        {

            services.AddControllers().AddNewtonsoftJson(options => {
                options.SerializerSettings.ContractResolver = new
DefaultContractResolver();
            }); ;
            var key = "This is my jwt authentication demo";
            services.AddAuthentication(x =>
            {
                x.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
                x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            }).AddJwtBearer(x =>
            {
                x.RequireHttpsMetadata = false;
                x.SaveToken = true;
                x.TokenValidationParameters = new TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new
SymmetricSecurityKey(Encoding.ASCII.GetBytes(key)),
                    ValidateIssuer = false,
                    ValidateAudience = false

                };
            });
            services.AddSingleton<IJwtAuthenticationManager>(new
JwtAuthenticationManager(key));
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title =
"WEBAPICOREJWTAuthJWTBearerWithHTTP", Version = "v1" });
            });
        }
```

Calling in JwtAuthenticationManager class file.

```
   public interface IJwtAuthenticationManager
    {
       string Authenticate(string email, string password);
       void AddCustomer(Customer customer);

    }

public class JwtAuthenticationManager: IJwtAuthenticationManager
       {

  public JwtAuthenticationManager (string Key)
        {
            this.Key = Key;
        }
public string Authenticate(string email, string password)
        {
            if (!CustomerList.Any(c => c.Email == email && c.Password == password))
            {
                return null;
            }
            var tokenHandler = new JwtSecurityTokenHandler();// install
System.IdentityModel.Tokens.Jwt
            var tokenKey = Encoding.ASCII.GetBytes(Key);
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = new ClaimsIdentity(new Claim[]
                {
                    new Claim(ClaimTypes.Email, email)


                }
                ),
                Expires = DateTime.UtcNow.AddHours(1),
                SigningCredentials = new SigningCredentials(
                    new SymmetricSecurityKey(tokenKey),
                    SecurityAlgorithms.HmacSha256Signature)


            };
            var token = tokenHandler.CreateToken(tokenDescriptor);
            return tokenHandler.WriteToken(token);

           }
```

2. Defining key in the controller

```
Route("api/[controller]")]
    [ApiController]
public class AuthController : ControllerBase
{
    // GET api/values
    [HttpPost, Route("login")]
    public IActionResult Login([FromBody] LoginModel user)
    {
```

```
        if (user == null)
        {
            return BadRequest("Invalid request");
        }

        if (user.UserName == "johncitizen" && user.Password == "abc@123")
        {
            var secretKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes("KeyForSignInSecret@1234"));
            var signinCredentials = new SigningCredentials(secretKey,
SecurityAlgorithms.HmacSha256);

            var tokeOptions = new JwtSecurityToken(
                issuer: "http://localhost:2000",
                audience: "http://localhost:2000",
                claims: new List<Claim>(),
                expires: DateTime.Now.AddMinutes(30),
                signingCredentials: signinCredentials
            );

            var tokenString = new
JwtSecurityTokenHandler().WriteToken(tokeOptions);
            return Ok(new { Token = tokenString });
        }
        else
        {
            return Unauthorized();
        }
    }
}
```

3. Defining key in appsettings.cs

appsettings.cs

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "Jwt": {
    "Key": "22384098209482309840923840983209483209480",
    "Issuer": "Cognizant"
  }
}
```

Calling in controller method file.

```
[EnableCors("AllowMyOrigin")]
    [Route("api/[controller]")]
    [ApiController]
    public class SecurityController : ControllerBase
    {
        // GET: api/Security
        private IConfiguration _config = null;
        public SecurityController(IConfiguration config)
        {
            _config = config;
        }

        private string GenerateJSONWebToken()
        {
            // when he is validated AD
            var securityKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(_config["Jwt:Key"]));
            var credentials = new SigningCredentials(securityKey,
SecurityAlgorithms.HmacSha256);
            var claims = new[] {
                new Claim(JwtRegisteredClaimNames.Sub, "abhishek"),
                new Claim(JwtRegisteredClaimNames.Email,
"abhisharma1108@yahoo.co.in"),
                new Claim("Role", "Admin"),
                new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString())
                };
            var token = new JwtSecurityToken(_config["Jwt:Issuer"],
              _config["Jwt:Issuer"],
              claims,
              expires: DateTime.Now.AddMinutes(120),
              signingCredentials: credentials);

            return new JwtSecurityTokenHandler().WriteToken(token);
        }
     }
```

Some startup configuration

```
public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to
the container.
        public void ConfigureServices(IServiceCollection services)
        {
          services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
         .AddJwtBearer(options =>
         {
             options.TokenValidationParameters = new TokenValidationParameters
             {
                 ValidateIssuer = true,
                 ValidateAudience = true,
                 ValidateLifetime = true,
```

```
                ValidateIssuerSigningKey = true,
                ValidIssuer = Configuration["Jwt:Issuer"],
                ValidAudience = Configuration["Jwt:Issuer"],
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
            };
        });
            services.AddControllersWithViews();
        }
```

JWT TOKEN CONTAINS FOLLOWING PIECES OF INFORMATION.