DZone

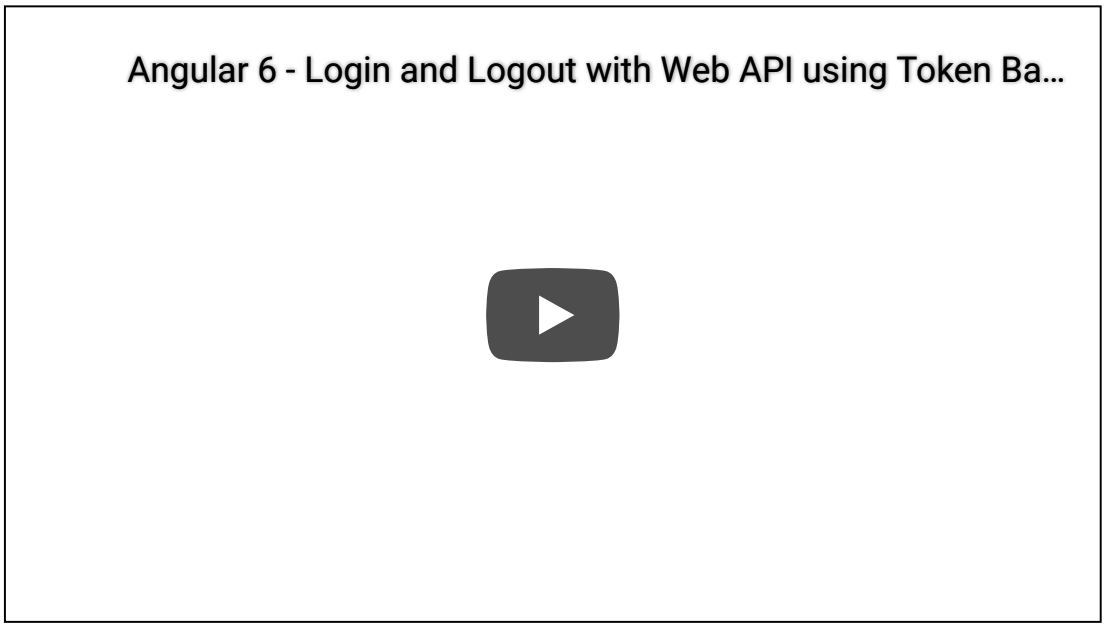
Token-Based Authentication in Angular 6 With Web API

by Deep Gautam · Sep. 25, 18 · Security Zone · Tutorial

Discover how to provide active runtime protection for your web applications from known and unknown vulnerabilities including Remote Code Execution Attacks.

In this tutorial, I will show how to perform token-based authentication with OWIN Middleware and a Web API that has the same integration with Angular 6.

If you prefer to watch a video on how to do this, here is the link for same, explaining token-based authentication with a Web API and Angular 6.



Now, let us get started! Here are the basic steps:

1. Angular 6 login and logout with the Web API using token-based authentication
2. Design login form in Angular 6 application
3. Web API token-based authentication using OWIN and ASP.Net Identity.

Let us start with the Web API:

Create a Web API Project

You will do this by creating a class, as shown below:

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Web;

namespaceDeepCart.Models
{
publicclassUser
{
[Key]
publicint Id { get; set; }

[Required, MinLength(3), MaxLength(50)]
publicstring UserName { get; set; }

[Required, MinLength(3), MaxLength(50)]
publicstring Password { get; set; }
}
}
```

Now, let's build the project.

Install the Following Packages in Your Web API Project

```
Microsoft.Owin

Microsoft.Owin.Host.SystemWeb

Microsoft.Owin.Security.OAuth

Microsoft.Owin.Security

Microsoft.AspNet.Identity.Owin

Microsoft.Owin.Cors
```

These packages are the important packages to give support for the OWIN Middleware and OAuth.

In this tutorial, we will be talking about bearer authentication.

Write a Provider Class

```
using Microsoft.Owin.Security;
using Microsoft.Owin.Security.OAuth;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;
using System.Web.Http.Cors;
using DeepCart.Models;
using System;

namespaceDeepCart.DtProvider
{
// This tutorial is by DotNet Techy YouTube Channel
// For more info about channel You can visit this link
// [](https://www.youtube.com/c/dotnettechy)

[EnableCors(origins: "*", headers: "*", methods: "*")]
publicclassDotNetTechyAuthServerProvider : OAuthAuthorizationServerProvider
{

publicoverrideasync Task ValidateClientAuthentication(OAuthValidateClientAuthenticationConte
{
context.Validated();
}

publicoverrideasync Task GrantResourceOwnerCredentials(OAuthGrantResourceOwnerCredentialsCon
{
var identity = new ClaimsIdentity(context.Options.AuthenticationType);

context.OwinContext.Response.Headers.Add("Access-Control-Allow-Origin", new[] { "*" });

using (var db = new DeepCartContext())
{
if (db != null)
{
var user = db.Registrations.ToList();
if (user != null)
{
```

```
if (!string.IsNullOrEmpty(user.Where(u => u.UserName == context.UserName && u.Password == cor
{
var currentUser = user.Where(u => u.UserName == context.UserName && u.Password == context.Pas
identity.AddClaim(new Claim("Role", currentUser.Role));
identity.AddClaim(new Claim("Id",Convert.ToString( currentUser.Id)));

var props = new AuthenticationProperties(new Dictionary<string, string>
{
{
"DisplayName", context.UserName
},
{
"Role", currentUser.Role
}
});

var ticket = new AuthenticationTicket(identity, props);
context.Validated(ticket);
}
else
{
context.SetError("invalid_grant", "Provided username and password is not matching, Please ret
context.Rejected();
}
}
}
else
{
context.SetError("invalid_grant", "Provided username and password is not matching, Please ret
context.Rejected();
}
return;
}
}

}
}
```

This `DotNetTechyAuthServerProvider` class is inherited from the `OAuthAuthorizationServerProvider`, and this

comes from `Microsoft.Owin.Security.OAuth`.

It has two basic methods that need to be overridden in order to validate the username and password from your database and return the token if the username and password are valid. The first method is `theValidateClientAuthentication`, and the second method is the `GrantResourceOwnerCredentials`. You will better understand this by looking at the code.

Create a Startup Class

```
using DeepCart.DtProvider;

using Microsoft.Owin;

using Microsoft.Owin.Cors;

using Microsoft.Owin.Security.OAuth;

using Owin;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http;

[assembly: OwinStartup(typeof(DeepCart.Startup))]

namespace DeepCart

{

// This tutorial is by DotNet Techy YouTube Channel

// For more info about channel You can visit this link

//

public class Startup

{
```

```
public void ConfigureAuth(IAppBuilder app)

{

app.UseCors(CorsOptions.AllowAll);
var OAuthOptions = newOAuthAuthorizationServerOptions
{
AllowInsecureHttp = true,
TokenEndpointPath = newPathString("/token"),
AccessTokenExpireTimeSpan = TimeSpan.FromMinutes(20),
Provider = newDotNetTechyAuthServerProvider()

};

app.UseOAuthBearerTokens(OAuthOptions);
app.UseOAuthAuthorizationServer(OAuthOptions);
app.UseOAuthBearerAuthentication(newOAuthBearerAuthenticationOptions());


HttpConfiguration config = newHttpConfiguration();
WebApiConfig.Register(config);

}

public void Configuration(IAppBuilder app)
{
//app.UseCors(Microsoft.Owin.Cors.CorsOptions.AllowAll);
ConfigureAuth(app);
GlobalConfiguration.Configure(WebApiConfig.Register);

}


}

}

This is the imprtant line which is loading Owin when application gets intilized.

[assembly: OwinStartup(typeof(DeepCart.Startup))]

Then in

```

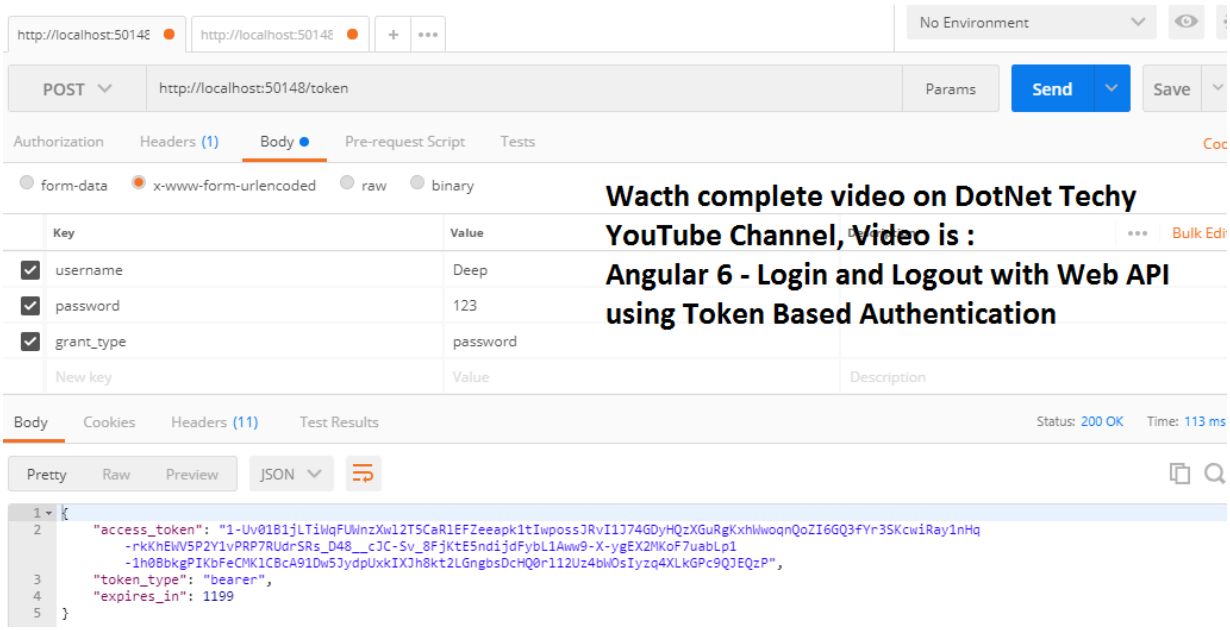
```
var OAuthOptions = new OAuthAuthorizationServerOptions

{

AllowInsecureHttp = true,

TokenEndpointPath = new PathString("/token"),

AccessTokenExpireTimeSpan = TimeSpan.FromMinutes(20),

Provider = new DotNetTechyAuthServerProvider()

};
```

We are providing our earlier created provider: `DotNetTechyAuthServerProvider`.

Testing the Access Token Generation

Next, we will make a request from the postman where the username and password should be from your database and whatever you have given registration.

Here is the screenshot of testing the token.



Integrate This Access Token in Angular 6

This involves multiple steps:

a) Create a service to call the Web API and get the token back.

b) Store the token for the next request to pass into the header

b) Store the token for the next request to pass into the header

c) Call the `ValidateUser` method form, and from your login button, click event

d) Create an `Auth Guard` and override the `CanActivate` method

Service Method

```
// This tutorial is by DotNet Techy YouTube Channel

// For more info about channel You can visit this link

//

ValidateUser (user:any)

{

var userData = "username=" + user.UserName + "&password=" + user.Password + "&grant_type=pass

var reqHeader = new HttpHeaders({ 'Content-Type': 'application/x-www-form-urlencoded','No-Aut

returnthis.httpClient.post(this.apiURL+ '/token',userData,{ headers: reqHeader })
.pipe(
map(res => res),
catchError( this.errorHandler)
);



}

Auth Guard

// This tutorial is by DotNet Techy YouTube Channel

// For more info about channel You can visit this link

// https://www.youtube.com/c/dotnettechy

canActivate(): boolean {

if (!this.auth.isAuthenticated()) {

//this.router.navigate(['login']);
```

```
37
38  console.log('You are not authrised to view this page')
39
40  return false;
41
42  }
43
44  return true;
45
46  }
47
48  store token in local storage
49
50  storeToken(token: string) {
51
52  localStorage.setItem("token", token);
53
54  }
```

Once the login is successful, you should call the `storeToken` method.

Find out how Waratek's award-winning application security platform can improve the security of your new and legacy applications and platforms with no false positives, code changes or slowing your application.

Like This Article? Read More From DZone

KeyValue Pipe in Angular 6.1

Best of DZone: Angular Tutorials for Beginners and Experts Alike

Deploying an Angular 6 Application to Cloud Foundry

Free DZone Refcard
Securing Mobile Applications With Cert Pinning

Topics: ANGULAR 6 PROJECT , ANGULAR TUTORIALS , WEB API , C# , DOTNET , ANGULARJS , SECURITY

Opinions expressed by DZone contributors are their own.

ecurity Partner Resources

ase Study: Virtual Patching While Under Attack
aratek
|
ghtweight plugin for Java and .NET to apply current and historical virtual patches, Critical Patch Updates (CPUs).
aratek
|
nd out how using compiler techniques means 100% Accuracy against common attacks and zero days.
aratek
|

How Does Elliptic Curve Cryptography Work?

by Anastasios Arampatzis · Mar 21, 19 · Security Zone · Tutorial

Diffie-Hellman and RSA cryptographic methods are based on the creation of keys by using very large prime numbers. Hence, key creation requires a lot computational power.

Elliptic curve cryptography (ECC) is a public key encryption technique based on an elliptic curve theory that can be used to create faster, smaller, and more efficient cryptographic keys. ECC generates keys through the properties of the elliptic curve equation instead of the traditional method of generation as the product of very large prime numbers.

The technology can be used in conjunction with most public key encryption methods, such as RSA and Diffie-Hellman. According to some researchers, ECC can achieve the same level of security with a 164-bit key that other systems require a 1,024-bit key. Because ECC helps to establish equivalent security with lower computing power and battery resource usage, it is becoming widely used for mobile applications. The use of elliptic curves in cryptography was suggested independently by Neal Koblitz and Victor S. Miller in 1985 and elliptic curve cryptography algorithms entered wide use around 2004.

The Need for a New Cryptographic Algorithm

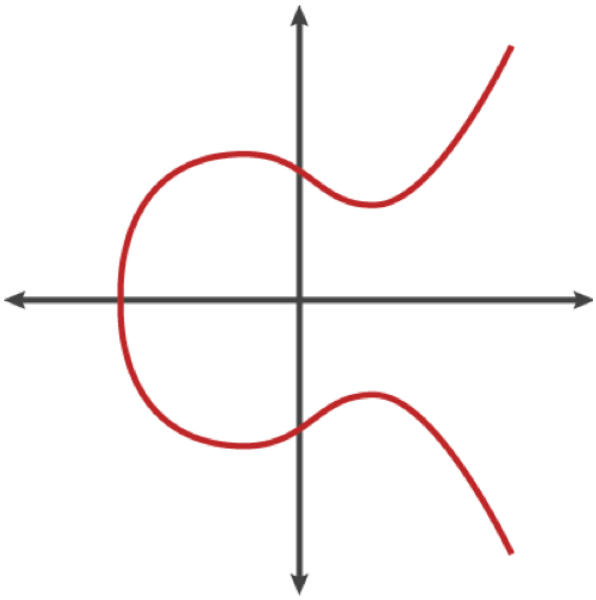
The key characteristic of an RSA algorithm is that it contains one process that is easy to do but difficult to undo. The easy part of the algorithm multiplies two prime numbers while the difficult pair is factoring the product of the multiplication into its two component primes. Algorithms that have this characteristic are known as Trapdoor Functions. Finding a good Trapdoor Function is critical to making a secure public key cryptographic system. Simply speaking, the bigger the spread between the difficulty of going one direction in a Trapdoor Function and going the other, the more secure a cryptographic system based on it will be.

Factoring is a well-known problem and has been studied since antiquity like the Sieve of Eratosthenes for finding all prime numbers up to any given limit. With the advancement of mathematics and computational power, the gap between the difficulty of factoring large numbers and multiplying large numbers is shrinking, hence the size of the keys need to grow even faster. This is not a sustainable situation for mobile and low-powered devices that have limited computational power. The gap between factoring and multiplying is not

sustainable in the long term. This means is that RSA is not the ideal system for the future of cryptography. In fact, recent research has demonstrated that even 2048-bits long RSA keys can be effectively downgraded via either man-in-the-browser or padding oracle attacks. Going one step further, the report suggests that the best countermeasure is to deprecate the RSA key exchange and switch to ECC.

How Does ECC work?

But what exactly is an elliptic curve and how does the underlying Trapdoor Function work? An elliptic curve is the set of points that satisfy a specific mathematical equation. The equation for an elliptic curve looks like this $y^2=x^3+ax+b$ and is being represented graphically like the image below.



Picture 1: Elliptic curve (source: blog.cloudflare.com)

Multiplying a point on the curve by a number will produce another point on the curve, but it is very difficult to find what number was used, even if you know the original point and the result. Equations based on elliptic curves have a characteristic that is very valuable for cryptography purposes: they are relatively easy to perform, and extremely difficult to reverse.

The elliptic curve discrete logarithms is the hard problem underpinning elliptic curve cryptography. Despite almost three decades of research, mathematicians still haven't found an algorithm to solve this problem that improves upon the naive approach. In other words, unlike with factoring, based on currently understood mathematics, there doesn't appear to be a shortcut that is narrowing the gap in a Trapdoor Function based around this problem. This means that for numbers of the same size, solving elliptic curve discrete logarithms is significantly harder than factoring. Since a more computationally intensive hard problem means a stronger cryptographic system, it follows that elliptic curve cryptosystems are harder to break than RSA and Diffie-Hellman.

To visualize how much harder it is to break, Lenstra, Kleinjung and Thome introduced in 2013 the concept of "Global Security." You can compute how much energy is needed to break a cryptographic algorithm and compare that with how much water that energy could boil. This is a kind of cryptographic carbon footprint.

compare that with how much water that energy could boil. This is a kind of cryptographic carbon footprint. By this measure, breaking a 228-bit RSA key requires less energy than it takes to boil a teaspoon of water. Comparatively, breaking a 228-bit elliptic curve key requires enough energy to boil all the water on earth. For this level of security with RSA, you'd need a key with 2,380-bits.

With ECC, you can use smaller keys to get the same levels of security. Small keys are important, especially in a world where more and more cryptography is done on less powerful devices like mobile phones. While multiplying two prime numbers together is easier than factoring the product into its component parts, when the prime numbers start to get very long, even just the multiplication step can take some time on a low powered device. While you could likely continue to keep RSA secure by increasing the key length that comes with a cost of slower cryptographic performance on the client. ECC appears to offer a better tradeoff: high security with short, fast keys.

Elliptic curve cryptography is now used in a variety of applications: the U.S. government uses it to protect internal communications; the Tor project uses it to help assure anonymity; it is the mechanism used to prove ownership of Bitcoins; it provides signatures in Apple's iMessage service; it is used to encrypt DNS information with DNSCurve; and it is the preferred method for authentication for secure web browsing over SSL/TLS.

Although there are certain cautions when implementing ECC, such as the use of a good source of random numbers on the machine making the signatures, the advantages of elliptic curve cryptography over traditional RSA are widely accepted.

Conclusions

Many experts are concerned that the mathematical algorithms behind RSA and Diffie-Hellman could be broken within 5 years. In fact, the developments on quantum computing will derive computational powers that will allow us to solve mathematical problems, such as factoring very large prime numbers that were once unsolvable. It is the ability of quantum computing to crunch so much data so fast that creates game-changing possibilities in various science fields including the data protection methods. One area of data protection that will be affected by quantum computing capabilities is encryption because current encryption algorithms will become obsolete since it could be deciphered in essentially less time. Will ECC be left as the only reasonable alternative or will Quantum Key Distribution prevail? Only time will show.

Like This Article? Read More From DZone

Writing SSL Proxy, Part I: Routing

Why Should Every eCommerce Website Have an SSL Certificate?

Trust Models for Secure Network Connections

Free DZone Refcard
Securing Mobile Applications With Cert Pinning

Topics: CRYTOGRAPHY, ELLIPTIC CURVES, ALGORITHM, SSL CERTIFICATE, SSL, TLS, TLS SECURITY, SECURITY, CYBERSECURITY

Opinions expressed by DZone contributors are their own.