CYBER WISE

ASP.NET UYGULAMALARINDA GÜVENLİK SIKILAŞTIRMALARI REHBERİ

Yasal Sorumluluklar

Cyberwise Research Task Force

- Harun Poyraz
- Rıdvan Ethem Canavar

İçindekiler

Yönetici Özeti

Yazı kapsamında .NET ekosisteminde genel olarak yer alan konfigürasyon dosyalarının güvenliği ve ASP.NET uygulamalarında bulunan Web.config konfigürasyon dosyası için alınabilecek güvenlik önlemleri yer almaktadır. Giriş bölümünde konfigürasyon dosyaları hakkında temel seviye bilgilere ve Web.config dosyasının yapısı hakkında bilgilere yer verilirken, genel güvenlik önlemleri bölümünde ise temel anlamda alınması gereken önlemler anlatılmıştır. Son bölümde yazılım geliştirme sürecinde uygulanması gereken önlemlere değinilmiştir.

1. Giriş

Konfigürasyon dosyaları uygulamalara özel ayarları içermektedir. ASP.NET uygulamaları bir veya birden fazla Web.config dosyasına sahip olabilirken, Windows uygulamaları opsiyonel olarak app.config dosyasına sahip olabilir.

ASP.NET uygulamalarında Web.config dosyası uygulama ana dizininde yer almakta, gereksinime binaen değişik uygulama seviyelerinde birden çok Web.config dosyası bulunabilmektedir. Konfigürasyon bilgileri XML tabanlı text dosyalarında saklanmaktadır.

Genel olarak bir Web.config dosyası içeriği;

- Veri tabanı bağlantı bilgileri (connectionStrings)
- Cache ayarları
- Session ayarları
- Hata ayıklama
- Güvenlik

şeklinde sıralanabilmektedir.

1.1. Web.config Dosya Yapısı

Web.config dosyaları XML kullanılarak yapılandırılmaktadır. Bu XML dosyalarında etiketler ve her etiket için de bir veya birden fazla alt etiketlerden oluşmaktadır. Aşağıda bir Web.config dosyası örneği verilmiştir.

```
1.          <?xml version=’1.0” encoding=”utf-8”?>
2.          <configuration>
3.            <connectionStrings>
4.              <add name="yourconnectinstringName" connectionString="Data Source= DatabaseServerName;
Integrated Security=true;Initial Catalog= YourDatabaseName; uid=YourUserName; Password=yourpassword; "
providerName="System.Data.SqlClient" />
5.            </connectionStrings>
6.            <appSettings>
7.              <add key="webpages:Version" value="3.0.0.0" />
8.              <add key="webpages:Enabled" value="false" />
9.              <add key="ClientValidationEnabled" value="true" />
10.           <add key="UnobtrusiveJavaScriptEnabled" value="true" />
11.           </appSettings>
12.           <configSections>
13.             <section name="SampleCustomTag"   type="System.Configuration.NameValueFileSectionHandler
" />
14.           </configSections>
15.           <system.net> <!–Include details about network classes, if any ?
  </system.net>
16.           <system.web>
17.             <compilation defaultLanguage="c#" debug="false"/>
18.             <customErrors mode="On" defaultRedirect="defaultCusErrPage.htm"/>
19.             <authentication mode=”Forms”>
20.               <forms loginUrl = "frmSampleLogin.aspx" timeout="20"/>
21.             </authentication>
22.             <authorization>
23.               <allow users="User1, User2" />
24.               <deny users=”*” />
25.             </authorization>
26.             <trace enabled="true" requestLimit="20" />
27.             <sessionState mode="SQLServer" stateConnectionString=”tcpip=127.0.0.1:8040”
sqlConnectionString="data source=127.0.0.1; Integrated Security=SSPI Trusted_Connection=yes”;
cookieless="true" timeout="40">
28.             </sessionState>
29.             <httpModules>
```

```
30.                 <add type="sampleClass,sampleAssembly" name="sampleModule" />
31.                 <remove name="modulename"/>
32.               </httpModules>
33.               <httpHandlers>
34.                  <add verb="*" path="sampleFolder/*.aspx" type="sampleHttpHandler,sampleAssembly" />
35.               </httpHandlers>
36.               <httpRuntime appRequestQueueLimit="200" executionTimeout="500" />
37.             </system.web>
38.             <sampleCustomTag>
39.               <add key="sampleKey" value="sample key value" />
40.             </sampleCustomTag >
41.           </configuration>
```

Dosya yukarıda görülebileceği üzere **<configuration>** adında bir kök elemente sahiptir. Diğer tüm elementler kök element altında bulunmaktadır. Bu kök element altında aşağıdaki alt elementlere yer verilebilmektedir.

connectionStrings: Bir uygulama içerisinde veri tabanı bağlantılarını gerçekleştirmek için kullanılan elementtir. Bu element içerisinde veri tabanına bağlanabilmek için gerekli olan bilgiler tanımlanmaktadır. Connection Strings, Web.config dosyası haricinde harici bir dosyaya da tanımlanabilmektedir.

appSettings: Geliştirilen uygulama içerisinde kullanılabilecek connection stringler, sunucu isimleri, dosya yolları gibi uygulama ile alakalı bir çok ayarın tanımlanabileceği elementtir.

configSections: Web.config dosyasının içerisinde istenildiği takdirde herhangi bir özel element tanımlanabilmektedir. Bu özel elementler **<configSections>** içerisinde **<section>** alt elementi kullanılarak tanımlanmaktadır. Örnek olarak yukarıdaki dosya içerisinde görülebileceği üzere **<samleCustomTag>** adında özel bir element dosyanın sonuna eklenmiştir. **<samleCustomTag>** içerisine istenildiği kadar key tanımlanabilmektedir.

system.net: Eğer herhangi bir ağ sınıfınız var ise bu element içerisine tanımlanabilmektedir.

system.web: ASP.NET sınıfları için gerekli olan konfigürasyon ayarları bu element altında tanımlanmaktadır. Birçok bölüm ve alt element tanımlaması bu element altında tanımlanabilir haldedir. Genel olarak en çok kullanılan ve en önemli alt elementler aşağıda verilmiştir.

- **compilation:** Bu element kullanılarak geliştirilen uygulamada hangi dilin kullanıldığı ve uygulamada debug mekanizmasının etkinleştirilip etkinleştirilemeyeceği gibi ayarların tanımlanabildiği bir elementtir.
- **customErrors:** Bu element içerisindeki mode ”**on**” olarak ayarlandığında uygulama geliştirilirken çözümlenmeyen herhangi bir hata meydana geldiğinde, varsayılan hata sayfasına yönlendirme yapılmasını sağlamaktadır. Ek olarak, bir hata koduna yönelik geliştirilecek hata sayfası da bu element altında element olarak tanımlanabilmektedir.
- **authentication:** Uygulama içerisinde hangi kimlik doğrulama mekanizmasının kullanılması gerektiğinin tanımlanabildiği elementtir.
- **authorization:** Kullanıcıların uygulama içerisindeki bazı dizin ve özelliklere erişebilmesi için tanımlanan yetki ayarlarının bulunduğu elementtir.
- **trace:** Bu element tanımlanarak tüm uygulama içerisindeki trace log kayıtları gözlemlenebilmektedir.
- **sessionState:** Kullanıcı oturumlarını ayarlamak için kullanılan bir elementtir.
- **httpModules:** Geliştirilen uygulama özel bir httpModules içeriyorsa bu element kullanılarak yapılandırılabilmektedir.
- **httpHandlers:** Geliştirilen uygulama herhangi bir özelleştirilmiş HTTP Handlers mekanizmasına sahipse bu element kullanılarak ayarlanabilmektedir.

httppRunTime: Bu element kullanılarak uygulamanın çalışma zamanındaki ayarları yapılandırılabilmektedir.

2. Genel Güvenlik Önlemleri

2.1. Konfigürasyon Dosyalarının Korunması

ASP.NET platformu için Microsoft tarafından birçok güvenlik önlemi sağlanmaktadır. Tavsiye edilen yöntemler belirli seviyede güvenlik sağlamaktadır. Bu yöntemler, sunucu üzerinde yönetici haklarına sahip bir saldırgan için etkili olmamaktadır. Günümüzde üçüncü parti firmalar tarafından geliştirilen bir takım güvenlik mekanizmaları, bu tarz durumlara karşı etkili olmaktadır. Bu sebeple konfigürasyon dosyalarının korunması ikiye ayrılmıştır. İlk bölüm güvenliğin tam olarak sağlanamadığı önerileri, ikinci bölüm ise tavsiye edilen güvenlik önerisini içermektedir.

2.1.1. Yetersiz Güvenlik Önlemleri

2.1.1.1. Connection Stringlerin Konfigürasyon Dosyalarında Saklanması

Connection Stringler'in uygulama kaynak koduna açık olarak yerleştirilmemesi gerekmektedir. Bu sebeple Connection Stringler'in konfigürasyon dosyalarında bulunması, uygulama kaynak kodunda bu değerlerin kullanımı gereksinimini ortadan kaldırmaktadır. Connection String'ler konfigürasyon dosyalarına alındıktan sonra bu dosyaların güvenliğinin sağlanması gerekmektedir.

Connection String'in yer aldığı örnek Web.config dosyası:

```
1.          <connectionStrings>
2.              <add name="yourconnectinstringName" connectionString="Data Source= DatabaseServerName;
Integrated Security=true;Initial Catalog= YourDatabaseName; uid=YourUserName; Password=yourpassword; "
providerName="System.Data.SqlClient" />
3.          </connectionStrings>
```

2.1.1.2. Veri tabanı bağlantısı için Windows Authentication Kullanımı

User ID, password gibi hassas bilgilerin ifşasını önlemek amacıyla Windows Authentication (*integrated security*) kullanımı önerilmektedir. Windows Authentication kullanımı ile User ID, password ihtiyacı tamamen kalkmakta, sunucuda ve veri tabanında sahip olduğu izinler kullanıcının yer aldığı Windows kullanıcı grupları tarafından sağlanmaktadır.

Windows Authentication kullanımının mümkün olmadığı durumlarda, hassas bilgilerin Connection String'ler içinde ifşa olmaması için ayrıca dikkat etmek gerekmektedir.

ASP.NET uygulamalarında bir Windows User, veri tabanı ve network kaynaklarına erişim için sabit kullanıcı olarak tanımlanabilmektedir. Bunun için Web.config **içerisinde** impersonation=”**true”** değeri tanımlanmalıdır.

```
1.          <identity impersonate="true"
2.                  userName="MyDomain\UserAccount"
3.                  password="*****" />
```

Tanımlanan kullanıcı **low-privilege** ilkesine göre tanımlanmalı, veri tabanı üzerinde sadece ihtiyacı olan izinler verilmelidir. Buna ek olarak kullanıcı adı ve parolaların konfigürasyon dosyalarında açık bir şekilde yer almaması için ilgili dosya bölümleri şifrelenmelidir.

2.1.1.3. Universal Data Link (UDL) Kullanımından Kaçınılması

Connection String'lerin Universal Data Link (UDL) dosyaları içinde **OleDbConnection** için saklanmasından kaçınılmalıdır. UDL'ler açık metin olarak saklanmakta ve .NET tarafından şifreleme olanağı sunmamaktadır.

Connection String içeren güvenli dosya örneği:

```
1.          using System;
2.          using System.Data.OleDb; /* OleDbConnection sınıfı, Data.OleDb isim uzayında yer almaktadır.
*/
3.           namespace OleDbCon1
4.          {
5.               class Class1
6.               {
7.                    static void Main(string[] args)
8.                    {
9.                         OleDbConnection conFriends=new OleDbConnection();
10.                        conFriends.ConnectionString="Provider=SQLOLEDB;Data
Source=localhost;Database=Friends;Integrated Security=SSPI";
```

```
11.
12.                         try
13.                         {
14.                              conFriends.Open();
15.                              Console.WriteLine("Conneciton open...");
16.                              conFriends.Close();
17.                              Console.WriteLine("Error");
18.                         }
19.                         catch(Exception error)
20.                         {
21.                              Console.WriteLine(error.Message.ToString());
22.                         }
23.                    }
24.               }
25.          }
```

2.1.1.4. Konfigürasyon Dosyaları Bölümlerinin Şifrelenmesi

ASP.net, "**protected configuration**" isimli güvenlik özelliğini konfigürasyon dosyası bölümlerinin şifrelenmesi için sunmaktadır. "**protected configuration**" ile Web.config içerisindeki username, password, connection string ve encryption key gibi değerler şifrelenebilmektedir. Bu bölümlerin şifrelenmesiyle, bir konfigürasyon dosyasının ifşa olması durumunda, kritik bilgilerin korunması sağlanmaktadır.

Şifreleme işlemi **ProtectedConfigurationProvider** sınıfı kullanılarak yapılmaktadır. Class iki farklı protected configuration provider'ı sunmaktadır.

- **DpapiProtectedConfigurationProvider:** Windows Data Protection API (DPAPI) kullanmaktadır.
- **RsaProtectedConfigurationProvider:** RSA encryption algoritmalarını kullanmaktadır.

Her iki provider'da verinin şifrelenmesi için olanak sağlamasına rağmen, şifrelenmiş konfigürasyon dosyasının farklı sunuculardaki kullanılması isteniyorsa, **RsaProtectedConfigurationProvider**'ın kullanılması gerekmektedir.

Konfigürasyon dosyalarında şifreleme işlemi için "**Aspnet_regiis.exe**" aracı kullanılmaktadır. Aşağıdaki örnek komut, SampleApplication isimli uygulamanın "**connectionStrings**" bölümünü **RsaProtectedConfigurationProvider** kullanarak şifrelemektedir.

```
1.          aspnet_regiis -pe "connectionStrings" -app "/SampleApplication" -prov
"RsaProtectedConfigurationProvider"
```

Şifre çözme işlemi de "**Aspnet_regiis.exe**" aracı kullanılarak yapılabilmektedir. Aşağıdaki örnek komut SampleApplication isimli uygulamanın "**connectionStrings**" bölümünü çözmektedir.

```
1.          aspnet_regiis -pd "connectionStrings" -app "/SampleApplication"
```

Sayfaya istekte bulunulduğunda.NET Framework, ilgili bölümün şifresini çözerek uygulamaya servis etmektedir. Bu servis esnasında açık veriler hafıza alanında bulunmaktadır. Uygulama hafıza alanının ele geçirilmesi durumunda ilgili verilerin açığa çıkabileceği unutulmamalıdır.

ASP.NET 2.0 öncesindeki versiyonlarda; konfigürasyon dosyasındaki processModel, runtime, mscorlib, startup, system.runtime.remoting, configProtectedData, satelliteassemblies, cryptographySettings, cryptoNameMapping, ve cryptoClasses bilgilerini içeren configProtectedData bölümü "protected configuration" ile şifrelenememektedir. Bu bölümdeki bilgilerin şifrelenmesi için ASP.NET Set Registry uygulanmasının kullanımı önerilmektedir.

Örnek Şifrelenmiş Web.config dosyası:

```
1.          <connectionStrings configP rotectionProvider="RsaProtectedConfigurationProvider">
2.              <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
3.                xmlns="http://www.w3.org/2001/04/xmlenc#">
4.                <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
5.                <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
6.                  <EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
7.                    <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
8.                    <KeyInfo xmlns="http://www.w3.org/2000/09/xmldsig#">
9.                      <KeyName>RSA Key</KeyName>
10.                   </KeyInfo>
11.                   <CipherData>
```

```
12.
<CipherValue>RXO/zmmy3sR0iOJoF4ooxkFxwelVYpT0riwP2mYpR3FU+r6BPfvsqb384pohivkyNY7Dm4lPgR2bE9F7k6TblLVJFvnQu
7p7d/yjnhzgHwWKMqb0M0t0Y8DOwogkDDXFxs1UxIhtknc+2a7UGtGh6Di3N572qxdfmGfQc7ZbwNE=
13.                     </CipherValue>
14.                   </CipherData>
15.                 </EncryptedKey>
16.               </KeyInfo>
17.               <CipherData>
18.
<CipherValue>KMNKBuV9nOid8pUvdNLY5I8R7BaEGncjkwYgshW8ClKjrXSM7zeIRmAy/cTaniu8Rfk92KVkEK83+UlQd+GQ6pycO3eM8
DTM5kCyLcEiJa5XUAQv4KITBNBN6fBXsWrGuEyUDWZYm6Eijl8DqRDb11i+StkBLlHPyyhbnCAsXdz5CaqVuG0obEy2xmnGQ6G3Mzr74j4
ifxnyvRq7levA2sBR4lhE5M80Cd5yKEJktcPWZYM99TmyO3KYjtmRW/Ws/XO3z9z1b1KohE5Ok/YX1YV0+Uk4/yuZo0Bjk+rErG505YMfR
VtxSJ4ee418ZMfp4vOaqzKrSkHPie3zIR7SuVUeYPFZbcV65BKCUlT4EtPLgi8CHu8bMBQkdWxOnQEIBeY+TerAee/SiBCrA8M/n9bpLlR
JkUb+URiGLoaj+XHym//fmCclAcveKlba6vKrcbqhEjsnY2F522yaTHcc1+wXUWqif7rSIPhc0+MT1hB1SZjd8dmPgtZUyzcL51DoChy+h
Z4vLzE=
19.                   </CipherValue>
20.                 </CipherData>
21.               </EncryptedData>
22.             </connectionStrings>
```

Bölümde bahsedilen şifreleme mekanizması, her ne kadar sistemin güvenliği için önem arz etse de, saldırganın IIS servisi üzerinde administrator yetkisine sahip olduğu veya "crypt32.dll" gibi Windows API'larına erişim sağladığı durumlarda, şifre çözme işlemi yapılarak, hassas veriler ortaya çıkarılabilecektir. Bu sebeple sisteme eklenen tüm kullanıcıların "**least privilage**" ilkesine binaen oluşturulması ve doğru ağ izolasyonun sağlanması önem arz etmektedir.

2.1.1.5. "Persist Security Info=False" İfadesinin Kullanılması

Persist Security Info değeri bağlantı kurulduktan sonra, güvenlik bilgilerinin (user ID ve password) tutulup tutulmayacağını belirtmek için kullanılmaktadır. Varsayılan değeri **False** olup, bağlantı sağlandıktan sonra bu veriler saklanmaz. Değer **True** olarak girildiğinde, hassas bilgilerin doğrulanmamış kaynaklar tarafından okunabilmesine olanak sağlanmaktadır.

"**Persist Security Info=False**" değeri kullanılarak Windows Authentication üzerinden veri tabanı bağlantısı örneği:

```
1.            "Persist Security Info=False;Integrated Security=true;
2.                Initial Catalog=AdventureWorks;Server=MSSQL1"
3.            "Persist Security Info=False;Integrated Security=SSPI;
4.                database=AdventureWorks;server=(local)"
5.            "Persist Security Info=False;Trusted_Connection=True;
6.                database=AdventureWorks;server=(local)"
```

2.1.1.6. Connection String Builder Kullanımı

Connection String injection saldırısı, dinamik olarak Connection String'lerin oluşturulması için string'lerin birleştirilmesi esnasında ortaya çıkmaktadır. İnput validasyonu ve zararlı karakterin filtrelenmesi sağlanmadığında, saldırgan hassas bilgilere ve sunucudaki diğer kaynaklara erişim sağlayabilmektedirler. Bu saldırıdan korunabilmek için Connection String Builder kullanımı sağlanmalıdır.

Konfigürasyon dosyası kullanılarak **SqlConnectionStringBuilder** kullanımı:

```
1.            <connectionStrings>
2.              <clear/>
3.              <add name="partialConnectString"
4.                connectionString="Initial Catalog=Northwind;"
5.                providerName="System.Data.SqlClient" />
6.            </connectionStrings>
```

Kodu çalıştırmak için projede "**System.Configuration.dll**" dosyasına referans verilmelidir.

```
1.            private static void BuildConnectionString(string dataSource,
2.                string userName, string userPassword)
3.            {
4.                // Retrieve the partial connection string named databaseConnection
5.                // from the application's app.config or Web.config file.
6.                ConnectionStringSettings settings =
7.                    ConfigurationManager.ConnectionStrings["partialConnectString"];
8.
9.                if (null != settings)
```

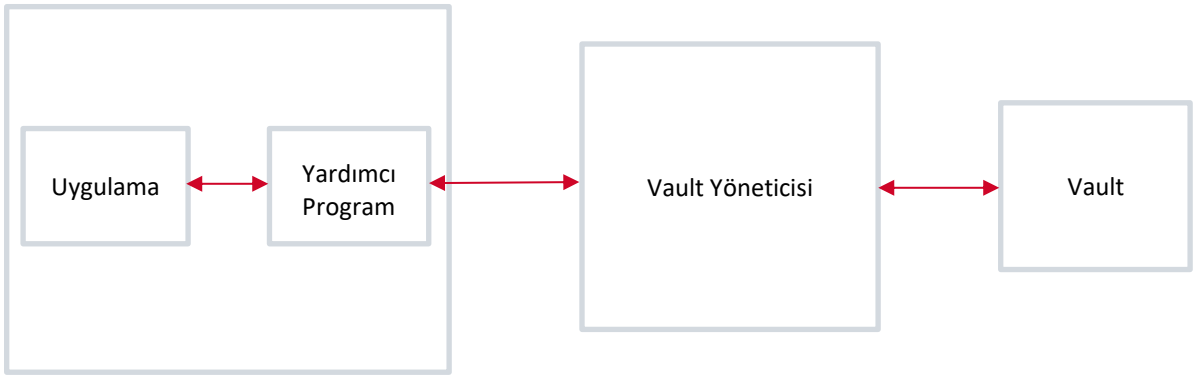
```
10.             {
11.                 // Retrieve the partial connection string.
12.                 string connectString = settings.ConnectionString;
13.                 Console.WriteLine("Original: {0}", connectString);
14.
15.                 // Create a new SqlConnectionStringBuilder based on the
16.                 // partial connection string retrieved from the config file.
17.                 SqlConnectionStringBuilder builder =
18.                     new SqlConnectionStringBuilder(connectString);
19.
20.                 // Supply the additional values.
21.                 builder.DataSource = dataSource;
22.                 builder.UserID = userName;
23.                 builder.Password = userPassword;
24.                 Console.WriteLine("Modified: {0}", builder.ConnectionString);
25.             }
26.         }
```

2.1.2. Önerilen Güvenlik Önlemleri

2.1.2.1. Üçüncü Parti Güvenlik Mekanizmaları

Günümüzde, saldırganların bahsettiğimiz şifreleme işlemini gerçekleştiren servisler üzerinde yönetim hakkına sahip olduğu durumlarda, hassas verilerin güvenliğini sağlamak amacıyla, üçüncü parti yazılım üreticileri tarafından birtakım çözümler geliştirilmiştir. Bu çözümlerin başında, hassas verileri "**Vault**" adı verilen uygulamadan ayrı bir lokasyonda saklama yaklaşımı yer almaktadır.

Bu yaklaşım farklı mekanizmalar ile sağlanabilmektedir. Vault yöneticisi ve uygulamanın birbirlerinden ayrı ortamlarda yer aldığı mimariler gibi, uygulama ve vault sisteminin aynı ortamda yer aldığı veya uygulama ile vault sistemi arasındaki iletişimin yardımcı programlar üzerinden sağlandığı çözümler de mevcuttur.



Örnekte, uygulama sunucusunda Vault ile iletişimi kurmak için yardımcı program kullanılmaktadır. Uygulamalar, ilgili kütüphaneler vasıtasıyla, Vault yöneticisi ile iletişim kurabilen yardımcı programa istek göndererek bilgileri elde etmekte, Vault yöneticine yapılacak isteği bu yardımcı program gerçekleştirmekte veya istenilen hassas veriler yardımcı programın cache'inde mevcutsa veriler istek gerçekleştirilmeden direkt olarak cache'den temin edilmektedir. Vault'lardaki veriler, şifreli olarak tutulmakta ve Vault'lara erişim yalnızca Vault yöneticisi tarafından sağlanmaktadır. Bu yaklaşımda, Vault'ların istek yapan uygulamayı doğrulaması için;

- İsteği gerçekleştiren OS kullanıcısı
- İsteği gerçekleştiren uygulamanın veya çalışan process'in hash değeri
- İstek gerçekleştirilen yol
- Uygulamanın imzalı olması durumunda sertifika seri numarası

gibi bilgiler, yardımcı program tarafından kullanılmaktadır. Bu doğrulama sayesinde şifrenin okunması amacıyla kodda yapılacak değişiklikler ve uygulamanın analiz programları ile çalıştırılması gibi durumlarda, doğrulama esnasında kullanılan bilgilerin hash değerleri değişeceğinden Vault'a yapılacak istekler başarısız olacaktır.

Bu yaklaşımın diğer bir avantajı, Vault'larda saklanan bilgilerin (bağlantı bilgileri, şifreler, tokenler, sertifikalar vb.) belli bir yaşam süresi tanımlanarak, belirli aralıklarla yenisiyle değiştirilmesidir. Bu şekilde bilgiler sadece belirli bir süre için geçerli olacaktır. Hatta

belirli bir süre için tanımlanabilen bilgiler, tek seferlik kullanım için de tanımlanabilmektedir. Bu yöntemle bağlantı kurulduktan hemen sonra şifrelerin değiştirilmesi yoluyla, saldırganın hafızadan alanından alabileceği şifreyi geçersiz kılmaktadır. Böylece maksimum önlem alınmış olmaktadır.

2.2. Hata Mesajlarının Güvenli Bir Şekilde İşlenmesi

Uygulamanın yapılandırmasını değiştirerek veya uygulamanın Global.asax dosyasına bir Application_Error işleyicisi ekleyerek, varsayılan hatalar uygulama düzeyinde güvenli bir şekilde işlenebilmektedir.

Web. config dosyasına bir **customErrors** elementi ekleyerek varsayılan HTTP hataları işlenebilmektedir. **customErrors** elementi, bir hata oluştuğunda kullanıcıların yeniden yönlendirileceği varsayılan sayfayı veya sayfaları belirtmenize olanak tanımaktadır. Ayrıca hata durum kodları için ayrı sayfalar tanımlamanızı sağlamaktadır.

Web.config dosyasında aşağıdaki tanımlama yapılarak hata mesajlarında bazı verilerin açığa çıkmasının önüne geçilebilmektedir. [2]

```
1.            <system.web>
2.              <customErrors mode="On"
defaultRedirect="ErrorPage.aspx?handler=customErrors%20section%20-%20Web.config">
3.                <error statusCode="404"
redirect="ErrorPage.aspx?msg=404&handler=customErrors%20section%20-%20Web.config"/>
4.              </customErrors>
5.            </system.web>
```

2.3. ViewStateMac Özelliğinin Aktifleştirilmesi

MAC (Message Authentication Code) sunucu tarafından oluşturulmuş ve **_VIEWSTATE**'in değerine eklenmiş bir kriptografik gizli form verisidir.

ViewStateMac güvenlik amaçlı konulmuş bir özelliktir. ViewState üzerindeki orijinal verinin bozulmasına yönelik herhangi bir saldırı olup olmadığını kontrol etmekte ve verileri base64 encoding yerine bir hash değeri ile tutmaktadır. Varsayılan değeri tüm ASP.NET versiyonlarında "**true**" olarak tanımlanmıştır ve ViewState özelliğinin olduğu her yerde bu özellik de vardır. ViewStateMac, sayfa tarayıcıya giderken verileri hash işleminden geçirmekte ve sayfa post back olduğunda gelen verilerle daha önceden oluşturduğu hash değerini karşılaştırmaktadır. Dolayısıyla bu özelliğin "**true**" olarak tanımlanması ASP.NET uygulamasının güvenliği için önemli olmaktadır.

Web.config dosyasında aşağıdaki tanımlama yapılarak, **_VIEWSTATE** ile uygulamanın yapısına uygun olmayan girdiler ile yapılan saldırıların önüne geçilebilmektedir.[3]

```
1.          <system.web>
2.            <pages enableViewStateMac="true">
3.          …
4.            </pages>
5.          </system.web>
```

2.4. Hata Ayıklamanın Doğru Yapılandırılması

Hata ayıklama, bir uygulamanın geliştirilme aşamasında çeşitli hataları hızlıca gidermek için geliştirilmiş bir mekanizmadır. Bu mekanizma uygulamanın yayınlandıktan sonra da aktif olarak bırakılması güvenli bir uygulama için olmaması gereken durumlardandır. Örneğin **debug="true"** olarak yayınlanmış bir uygulamada sunucu üzerinde daha fazla kaynak tüketimine neden olan servis dışı bırakma saldırıları gerçekleştirilebilmektedir.

Ek olarak ASP.NET uygulamalarında debug esnasında sonuçların binary olarak çıktı verilebilmesi için ayar yapılabilmektedir. Bu debug çıktı mesajları uygulama hakkında detay bilgiler verdiğinden dolayı, uygulama yayınlandıktan sonra saldırganların uygulama hakkında detaylı bilgiler edinmesine neden olmaktadır. Dolayısıyla debug özelliğinin sadece geliştirme veya test aşamalarında kullanılması gerekmektedir.

Web.config içerisine aşağıdaki tanımlama yapılarak bu mekanizma devre dışı bırakılabilmektedir.

```
1.            <system.web>
2.                <compilation  debug="false" ... >
3.                  …
4.                </compilation>
5.            </system.web>
```

2.5. ASP.NET Request Validation Özelliğinin Aktifleştirilmesi

Request validation, bir ASP.NET uygulamasının HTTP istekleri üzerinde olası tehlikeli içerikleri incelemesini sağlayan bir özelliktir. Bu özellik, istek içerisinde gönderilen bazı sorgular ve betikler ile gerçekleştirilmeye çalışılan saldırılar için koruma sağlamaktadır. XSS vb. saldırılarına karşı önlem alma konusunda da etkilidir.

Web sitesi bu özellik tetiklendiği zaman "**Potansiyel bir tehlike algılandı.**" uyarısını ekrana getirmektedir.

Web.config dosyasına aşağıdaki tanımlama yapılarak bahsedilen zararlı istekler için koruma sağlanabilmektedir.[6]

```
1.              <system.web>
2.                 <pages validateRequest="true">
3.                  …
4.                 </pages>
5.              </system.web>
```

2.6. ASP.NET Tracing Özelliğinin Kapatılması

ASP.NET Tracing, bir sayfa üzerindeki bazı problemleri tespit etmek için kullanılabilen bir özelliktir. Bu özellik ile bir sayfanın çalıştırılma yolu görülebilmekte, çalışma zamanındaki bazı problemler tespit edilebilmekte ve bazı debug işlemleri gerçekleştirilebilmektedir.

ASP.NET Tracing'in yayına alınan bir uygulamada aktif olması, gerçekleştirilen istekler hakkındaki tüm bilgileri herhangi bir kullanıcıya sunabilmektedir. Sonuç olarak uygulama hakkındaki hassas bilgilerin ortaya çıkmasına neden olmaktadır. Dolayısıyla yayınlanan bir uygulamanın Tracing özelliği kapalı olmalıdır.

Web.config içerisinde aşağıdaki tanımlama yapılarak Tracing özelliği kapatılabilmektedir. [7][8]

```
1.             <system.web>
2.               <trace enabled="false" />
3.             </system.web>
```

2.7. Web.config İçerisinde Parola Bilgisinin Boş Bırakılmaması

Uygulama geliştirme süreçlerinde geliştiriciler config dosyalarının içerisine hassas veriler bırakabilmektedir. Böyle bir durumda config dosyalarının içeriğini okuyabilecek saldırganlar için adeta saldırıya davet çıkartılmış olunmaktadır.

Fakat bu durumun farkında olan geliştiriciler ise farklı bir zafiyete neden olabilmektedir. Örneğin bir veri tabanı bağlantısı için tanımlanmış olan connectionString içerisinde parola bilgisini, aşağıda görülebileceği gibi boş bir şekilde ekleyebilmektedirler.

```
1.           <connectionStrings>
2.              <add name="ud_DEV" connectionString="connectDB=uDB; uid=db2admin; pwd=; dbalias=uDB;"
providerName="System.Data.Odbc" />
3.           </connectionStrings>
```

connectionString içeriğinde görülebileceği üzere **pwd** boş olarak tanımlanmıştır. Bu durum saldırganların uygulamaya yetkisiz olarak erişim sağlamasına neden olmaktadır. **Kullanıcı adı ve parola bilgileri config dosyalarına açık bir şekilde eklenmemelidir.**[9][10]

2.8. Cookie'nin HttpOnly Olarak Tanımlanması

XSS (Cross-site Scripting) saldırıları ile kötü amaçlı kişiler tarayıcı üzerinde kendi JavaScript kodlarını çalıştırabilmektedirler. Aynı zamanda saldırganlar JavaScript kodları ile Cookie'leri elde edip kullanıcıların oturumlarını da elde edebilmektedirler.

Sunucu **HttpOnly** flag'ini kullanarak, tarayıcıya ilgili Cookie'nin JavaScript yoluyla erişilmemesini sağlamaktadır. Bu yöntem Cookie'nin XSS yoluyla çalınmasının önüne geçmektedir. Aynı zamanda bu flag ile Session Cookie de elde edilemediğinden dolayı kullanıcı oturumunun elde edilmesi engellenmiş olmaktadır.

Web.config dosyasına aşağıdaki tanımlama yapılarak XSS saldırılarından Cookie'ler korunabilmektedir.[11][12]

```
1.           <system.web>
2.               <httpCookies httpOnlyCookies="true" … />
3.           </system.web>
```

2.9. Cookie'nin Secure Olarak Tanımlanması

Web uygulamalarında sunucu-istemci arası iletişimin şifreli bir şekilde gerçekleşmesi güvenlik açısından önemlidir. Bu şekilde gerçekleşen bir iletişimde, saldırganlar ağı dinleseler bile gizli bilgilere erişimleri engellenmiş olmaktadır. Cookie'lerin güvenli bir şekilde iletilmesi için de **Secure** flagine sahip olmaları gerekmektedir.

Sunucu **Secure** flag'ini kullanarak tarayıcıya Cookie'nin sadece HTTPS üzerinden yapılan güvenli isteklerde gönderilmesini söylemektedir. Bu sayede Cookie'nin ağı dinleyen kişilerce çalınması engellenmektedir.

Web.config dosyasına aşağıdaki tanımlama yapılarak Cookie'lerin güvenli bir şekilde iletilmesi sağlanabilmektedir.[11][12]

```
1.          <system.web>
2.              <httpCookies requireSSL="true" />
3.          </system.web>
```

2.10. Session Fixation Saldırısının Önlenmesi

Session fixation saldırıları ile saldırganlar bir kullanıcı oturumunu elde edebilmektedirler. Bu saldırı web uygulamasının oturum yönetimindeki zafiyetinden kaynaklanmaktadır. Zafiyet, bir kullanıcının kimliği uygulama tarafından doğrulandıktan sonra, uygulamanın kullanıcıya önceden var olan bir oturum ID'si atamasıyla ortaya çıkmaktadır. Saldırganlar burada daha önce atanmış olan oturum ID'sini elde edebilmek için kurbanın tarayıcısını kullanmak zorundadırlar.

Session Fixation saldırısı aslında bir **Session Hijack** türünden saldırıdır. Fakat **Session Hijack** kullanıcı oturumunun, sunucu-istemci arasında bağlantı kurulup kullanıcının uygulamaya giriş yaptıktan sonra çalınmasıyla gerçekleşmektedir. Session Fixation saldırısında ise kullanıcı oturumu, kullanıcı tarayıcısından elde edilmektedir. Dolayısıyla kullanıcı giriş yapmadan önce de oturumu elde edilebilmektedir.

Bu saldırı düzgün bir oturum yapılandırmasıyla önlenebilmektedir. Fakat ek önlem olarak kullanıcı oturumu belirli periyotlarda sonlandırılmalıdır.

Web.config dosyasına aşağıdaki tanımlama yapılarak kullanıcı oturumlarının timeout süresi, 20 dakika olarak ayarlanabilmektedir.[13][14]

```
1.          <system.web>
2.             <sessionState timeout="20"/>
3.          </system.web>
```

2.11. Tamamlanmamış Rollerin Korunması

Bir web uygulaması geliştirirken rol yönetimi oldukça önemlidir. Yanlış yapılandırılmış bir rol yönetimi, güvenlik açısından kritik sonuçlara yol açabilmektedir.

ASP.NET uygulamalarında **cacheRolesInCookie** değişkeni, Web.config dosyasında system.web/authentication/forms elementinde "**true**" olarak tanımlandığında her kullanıcı için tanımlanan roller bir cookie içerisinde cache olarak tutulmaktadır. Eğer bu bilgiler açık bir şekilde tutuluyorsa saldırganlar tarafından ele geçirilebilmektedir. Dolayısıyla da bu rol bilgileri yetkisiz kişiler tarafından düzenlenebilir.

Web.config dosyasına aşağıdaki tanımlama yapılarak bu durumun önüne geçilebilmektedir. [15]

```
1.          <system.web>
2.             <authentication mode="Forms">
3.                <forms name=".ASPXFORMSDEMO" loginUrl="logon.aspx"
4.                    protection="All" path="/" timeout="30" cacheRolesInCookie=”false” />
5.             </authentication>
6.          </system.web>
```

2.12. HTTP Header Injection Saldırılarının Önlenmesi

Bu saldırıda saldırganlar özel bir başlık kullanarak kullanıcıyı başka bir sayfaya yönlendirme, XSS atağını tetikleme ve sayfanın tamamının yeniden düzenlenmiş bir şekilde yüklenmesini sağlama gibi saldırıların gerçekleşmesi sağlanabilmektedir.

Web.config dosyasına aşağıdaki gibi **enableHeaderChecking** tanımlaması yapılarak HTTP Header Injection saldırılarının önüne geçilebilmektedir. Detaylı bilgi için şuraya bakılabilir. [16][17]

```
1.          <system.web>
2.            <httpRuntime targetFramework="4.6" enableHeaderChecking="true" />
3.          </system.web>
```

2.13. Cross-Site Request Forgery (CSRF) Saldırılarının Önlenmesi

Cross Site Request Forgery (CSRF), saldırganın kendi hazırladığı zararlı bir web sitesi üzerinde, kullanıcının bir takım tıklamalar yapmasını sağlayarak, başka bir domain'deki web sitesine kendine göre hazırladığı isteği post ettirmesini sağlamasıdır

Bu saldırılar, SameSite Cookie'ler kullanılarak önlenebilmektedir.

Web.config dosyasına aşağıdaki gibi tanımlama gerçekleştirilebilmektedir.

```
1.          <system.webServer>
2.            <rewrite>
3.              <outboundRules>
4.                <clear />
5.                <rule name="Add SameSite" preCondition="No SameSite">
6.                  <match serverVariable="RESPONSE_Set_Cookie" pattern=".*" negate="false" />
7.                  <action type="Rewrite" value="{R:0}; SameSite=lax" />
8.                </rule>
9.                <preConditions>
10.                 <preCondition name="No SameSite">
11.                   <add input="{RESPONSE_Set_Cookie}" pattern="." />
12.                   <add input="{RESPONSE_Set_Cookie}" pattern="; SameSite=strict" negate="true" />
13.                 </preCondition>
14.               </preConditions>
15.             </outboundRules>
16.           </rewrite>
17.         </system.webServer>
```

2.14. Cross-Site Tracing (XST) Saldırılarının Önlenmesi

Cookie'lerin **HTTPS** üzerinden aktarılması ve **HttpOnly** kullanımı cookie'lerin korunması için yeterli değildir. Cross-Site Tracing adı verilen teknik ile saldırganlar **TRACE** ve **TRACK** metotlarını kullanarak cookie aktarımı yapabilirler.

Günümüzde modern tarayıcılar, bu metotların JavaScript ile yapılmasını engellemektedirler. Ek güvenlik yapılandırması olarak, Web.config dosyasına aşağıdaki gibi tanımlama yapılarak bahsedilen metotlar engellenebilmektedir. [18]

```
1.          <system.webServer>
2.            <security>
3.              <requestFiltering>
4.                <verbs>
5.                  <add verb="TRACE" allowed="false" />
6.                  <add verb="TRACK" allowed="false" />
7.                </verbs>
8.              </requestFiltering>
9.            </security>
10.         </system.webServer>
```

2.15. Kullanıcı Girişi Formunda Cookie Kullanımın Zorlanması

ASP.NET, giriş formunda, cookie kullanımı için 4 seçenek sunmaktadır; UseUri, UseCookies, AutoDetect, UseDeviceProfile. Bunlardan **UseCookies** seçeneği, kullanıcı giriş formlarında, güvenli mekanizma olan Cookie kullanımını zorunlu kılmaktadır. ASP.NET varsayılan yetkilendirme sayfası yapılandırmasında yer alan değer **UseCookies** olmakla birlikte, bu değerin gerekli olmadıkça değiştirilmemesi önerilmektedir. [19]

Web.config dosyasındaki varsayılan yetkilendirme yapılandırması aşağıdaki gibidir.

```
1.          <system.web>
2.            <authentication mode="Forms">
3.              <forms name=".ASPXAUTH"
4.                     loginUrl="login.aspx"
5.                     protection="All"
6.                     timeout="30"
7.                     path="/"
8.                     requireSSL="false"
9.                     slidingExpiration="true"
10.                    defaultUrl="default.aspx"
11.                    cookieless="UseCookies" />
12.           </authentication>
13.         </system.web>
```

2.16. Kullanıcı Girişi Formunda SlidingExpiration Özelliğinin Kapatılması

SlidingExpiration özelliği, kullanıcının cookie son kullanım süresini sıfırlamak için kullanılmaktadır. Kullanıcı, giriş yaptıktan sonra, cookie son kullanım süresinin yarısı geçmiş ve başarılı bir istek gerçekleştirilmişse, SlidingExpiration özelliği, cookie son kullanım süresini sıfırlamaktadır.

SlidingExpiration özelliğinin **False** olarak yapılandırılması, yapılandırılan son kullanım süresinin bağlı olarak, kimlik doğrulama tanımlama bilgisinin geçerli olduğu süreyi sınırlayarak uygulamanın güvenliğine katkı sağlayabilmektedir. [20]

Web.config dosyasındaki örnek yapılandırma aşağıdaki gibidir.

```
1.          <authentication mode="Forms">
2.            <forms loginUrl="member_login.aspx"
3.              name=".ASPXFORMSAUTH"
4.              cookieless="UseCookies"
5.              requireSSL="true"
6.              slidingExpiration="false" />
7.          </authentication>
```

2.17. Kullanıcı Girişi Formunda CookieDomain Özelliğinin Aktif Edilmesi

CookieDomain özelliği, yetkilendirme formunda, cookie değerinin aktif olacağı domaini belirtmek için kullanılmaktadır. ASP.NET ekosisteminde, varsayılan değeri boş **(“ ”)** olarak tanımlanmıştır. Bu değerin forma sağlanması önerilmektedir. [21]

Web.config dosyasındaki örnek yapılandırma aşağıdaki gibidir.

```
1.            <authentication mode="Forms">
2.              <forms loginUrl="member_login.aspx"
3.                cookieless="UseCookies"
4.                domain="contoso.com" />
5.            </authentication>
```

2.17. Varsayılan Session ID İsminin Değiştirilmesi

ASP.NET ekosisteminde, varsayılan session ismi “**ASP.NET_SessionId**”dir. Bu değer sistemin saldırganlar tarafından kolayca tespit edilmesini sağladığından, varsayılan ismin değiştirilmesi önerilmektedir.

Web.config dosyasında, aşağıdaki ifade kullanılarak, session ismi değiştirilebilmektedir. [22]

```
1.            <sessionState cookieName="newvalue" />
```

2.18. X-Permitted-Cross-Domain-Policies

Flash içeriklerde, cross-origin isteklerinin engellenmesi için Web.config dosyasına, **X-Permitted-Cross-Domain-Policies** ifadesinin **none** olarak eklenmesi gerekmektedir. [18]

Örnek Web.config yapılandırması aşağıdaki gibidir.

```
1.            <system.webServer>
2.              <httpProtocol>
3.                <customHeaders>
4.                  <add name="X-Permitted-Cross-Domain-Policies" value="none" />
5.                </customHeaders>
6.              </httpProtocol>
7.            </system.webServer>
```

2.19. Form Authentication Timeout Tanımlanması

Form Authentication Timeout değeri, yetkilendirme sayfalarında, kullanıcı giriş yaptıktan sonra oluşturulan cookie’nin aktif olma süresini belirlemektedir. Kalıcı veya uzun süreli değerler, güvenlik riski oluşturmaktadır. Değerin varsayılan değeri, 30 dakikadır. Bu ifadenin kullanılması ve sürenin çok uzun tutulmaması önerilmektedir.

Süre değişikliği için Web.config dosyasında, aşağıdaki ifade yapılandırılabilmektedir. [23]

```
1.            <forms loginUrl="~/Login" timeout="20" />
```

2.20. Session State Timeout Tanımlanması

Session State Timeout değeri, yetkilendirme sayfalarında, kullanıcı giriş yaptıktan sonra oluşturulan session'ın, hafızada (SQL Server vb.) tutulma süresini belirlemektedir. Süresiz veya uzun süreli değerler, güvenlik riski oluşturmaktadır. Değerin varsayılan değeri 20 dakikadır. Bu ifadenin kullanılması ve sürenin çok uzun tutulmaması önerilmektedir.

Süre değişikliği için Web.config dosyasında, aşağıdaki ifade yapılandırılabilmektedir. [23]

```
1.          <sessionState timeout="30" mode="InProc" />
```

Form Authentication Timeout ve Session State Timeout değerlerinin birbiriyle uyumlu şekilde kullanılması önerilmektedir. Aksi durumda, kullanıcı deneyimi açısından uyumsuzluk hatalarıyla karşılaşılabilmektedir.

2.21. Event Validasyonu Kullanımı

Event validasyonu, ASP.NET ekosisteminde, injection saldırılarını önlemek amacıyla geliştirilmiş bir güvenlik mekanizmasıdır. Varsayılan olarak aktif yapılandırılmaktadır. Bu mekanizmanın performans veya diğer bir amaçla kapatılmaması önerilmektedir.

Web.config dosyasında, aşağıdaki tanım ile Event Validation, aktif olarak yapılandırılabilmektedir. [24]

```
1.          <system.web>
2.             <pages enableEventValidation="true"/>
3.          </system.web>
```

2.22. Maksimum İstek Boyutunun Sınırlandırılması

Saldırganların, bazı zafiyetleri istismar edebilmek için büyük boyutta **HTTP** istekleri gönderebilmektedirler. İstek boyutunun sınırlandırılması, uygulamaların bazı zafiyetler tarafından istismar edilememesi için gerekli olabilmektedir. ASP.NET ekosisteminde, istek boyutunun belirlenebilmesi için **maxRequestLength** ifadesi kullanılmaktadır.

Web.config dosyası için aşağıdaki yapılandırma örnek olarak verilmiştir. Örnekte, istek boyutu 20kb olarak sınırlandırılmıştır. [25]

```
1.          <configuration>
2.           <system.web>
3.           <httpRuntime maxRequestLength="200"/>
4.           </system.web>
5.          </configuration>
```

2.23. Güvenli Şifreleme Algoritmalarının Kullanılması

ASP.NET, farklı şifreleme algoritmalarını desteklemektedir. Bu algoritmaların kullanımı tercih edilirken, algoritma güvenliğine dikkat edilerek tercih edilmesi, uygulama güvenliği açısından önemlidir. ASP.NET 4 ile SHA-2 algoritma ailesi, ekosisteme eklenmiştir. Önceki versiyonlardan beri desteklenen AES ve SHA1 gibi algoritmaların kullanımından kaçınılması ve SHA-2 algoritma ailesinin kullanılması önerilmektedir. [26]

2.24. Cross-Origin Resource Sharing (CORS) Kullanımı

Güvenlik önlemleri sebebiyle, tarayıcılarda farklı kaynaklar arasında istek yapılması engellenmektedir. Tarayıcılar **Same-Origin Policy** adı verilen güvenlik mekanizması sayesinde, yapılan istek; aynı **schema**, **host** ve **port**'taki **URL** tarafından alınması gerekmektedir. Kimi durumlarda, bu tarz isteklerin yapılması gerektiğinde, **Cross-Origin Resource Sharing** mekanizması kullanılarak istek yapılabilmektedir. Web sunucusunun, farklı kaynaktan gelen isteği alabileceğini belirtmek için aşağıdaki **HTTP** başlığı, **value** kısmına domain adı eklenerek kullanılabilmektedir. [27]

```
1.          Access-Control-Allow-Origin" value="example.com"
```

Kimi durumlarda geliştiriciler, uygulama geliştirme sürecinde kolaylık sağlaması amacıyla, CORS tanımını geniş tutulabilmektedirler. Aşağıdaki yapılandırmada tavsiye edilmeyen bir tanım gerçekleştirilmiştir. CORS tanımı yapılırken, **kapsamın kısıtlı tutulması** önerilmektedir.

```
1.          <httpProtocol>
2.              <customHeaders>
3.                 <add name="Access-Control-Allow-Origin" value="*" />
4.              </customHeaders>
5.          </httpProtocol>
```

2.25. Varsayılan Hata Sayfalarının Değiştirilmesi

ASP.NET ekosisteminde varsayılan hata sayfaları, doğru yapılandırılmadığı takdirde, saldırganların uygulama hakkında bilgi elde etmesini kolaylaştırmaktadır. Bu yüzden varsayılan hata sayfalarının bilgi açığa çıkartmayacak şekilde yapılandırılması önerilmektedir.

Web.config dosyasında, uygun yapılandırmanın gerçekleştirilmesi için aşağıdaki tanımlamalar kullanılabilmektedir. [28]

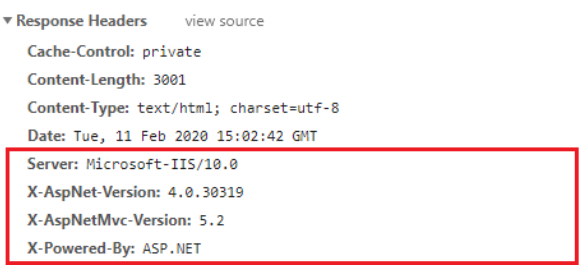
```
1.          <system.webServer>
2.            <httpErrors errorMode="Custom">
3.              <remove statusCode="502" subStatusCode="-1" />
4.              <remove statusCode="501" subStatusCode="-1" />
5.              <remove statusCode="500" subStatusCode="-1" />
6.              <remove statusCode="412" subStatusCode="-1" />
7.              <remove statusCode="406" subStatusCode="-1" />
8.              <remove statusCode="405" subStatusCode="-1" />
9.              <remove statusCode="404" subStatusCode="-1" />
10.             <remove statusCode="403" subStatusCode="-1" />
11.             <remove statusCode="401" subStatusCode="-1" />
12.             <remove statusCode="400" subStatusCode="-1" />
13.
14.             <error statusCode="400" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
15.             <error statusCode="401" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
16.             <error statusCode="402" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
17.             <error statusCode="403" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
18.             <error statusCode="404" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
19.             <error statusCode="405" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
20.             <error statusCode="406" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
21.             <error statusCode="412" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
22.             <error statusCode="500" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
23.             <error statusCode="501" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
24.             <error statusCode="502" subStatusCode="-1" path="/error-handler.aspx"
prefixLanguageFilePath="" responseMode="ExecuteURL" />
25.           </httpErrors>
26.         </system.webServer>
```

2.27. HTTP Başlık Bilgilerinin Yapılandırılması

Keşif çalışmalarında, keşfi gerçekleştiren kişi için uygulamalar hakkındaki bilgileri elde etmesi önemlidir. Bir web uygulamasının geliştirilme sürecinde hassas bilgilerin veya saldırganların işine yarayabilecek bilgilerin açığa çıkmaması gerekmektedir. Bu amaçla HTTP protokolünün bizlere sunduğu başlık bilgileri bulunmaktadır. Web.config içerisinde bazı zafiyetlerin önlenebilmesi için tanımlanabilecek başlık bilgisi yapılandırmaları aşağıda verilmiştir.

2.27.1. Sunucu Versiyon Bilgisinin Açığa Çıkarılmasının Önlenmesi

Bir web sunucusu; normal olarak sunucu bilgisi, uygulama versiyon bilgisi, işletim sistemi bilgisi gibi bazı bilgileri **HTTP Response** başlıklarında döndürebilmektedir. Bunun bir örneği olarak aşağıda IIS ve ASP.NET bilgilerini döndüren bir başlık örneği sunulmuştur.



Şekil – 1. HTTP Response

"**X-Powered-By**" ve "**X-AspNetMvc-Version**" bilgilerini kaldırmak için **<httpProtocol>** elementinin alt elementi olan **<customHeaders>** elementi aşağıdaki gibi kullanılabilmektedir.

```
<system.webServer>
   <httpProtocol>
     <customHeaders>
       <remove name="X-Powered-By" />
       <remove name="X-AspNetMvc-Version" />
     </customHeaders>
   </httpProtocol>
</system.webServer>
```

"**X-AspNet-Version**" bilgisinin kaldırılabilmesi için **<httpRuntime>** elementi aşağıdaki gibi kullanılabilmektedir.

```
<system.web>
    <httpRuntime targetFramework="4.6" enableVersionHeader="false" />
</system.web>
```

"**Server**" bilgisinin kaldırılabilmesi için **global.asax** dosyasında örnek olarak aşağıdaki gibi bir değişiklik yapılabilmektedir.

```
protected void Application_PreSendRequestHeaders()
{
      if (HttpContext.Current != null)
      {
         HttpContext.Current.Response.Headers.Remove("Server");
      }    }
```

2.27.2. Clickjacking Zafiyetinin Önlenmesi

Clickjacking, kullanıcılara farkında olmadan tıklama yaptırmak için kullanılan bir gizli saldırı tekniğidir. Bu saldırıyı önleyebilmek için "X-Frame-Options" başlık bilgisini istekte eklememiz gerekmektedir. Örnek ayar aşağıdaki gibi tanımlanabilmektedir.

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="X-Frame-Options" value="DENY" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

2.27.3. Cross-Site Scripting Saldırılarının Önlenmesi

XSS olarakta bilinen bu zafiyet kullanıcıların tarayıcılarında genellikle JavaScript kodu çalıştırılarak yapılan bir saldırı türüdür. Bu saldırı, "**1;mode=block**" değeriyle "**X-Xss-Protection**" başlığının eklenerek kullanılması sonucunda tarayıcılarda varsayılan olarak XSS filtrelemenin etkinleştirilmesini sağlamaktadır. Eğer zararlı bir JavaScript satırı tespit edilirse, tarayıcı sayfada bu zararlı ifadeyi kod içerisinden temizleyecektir. Tabii ki ilk olarak güvenli kod geliştirildikten sonra bu başlık sayesinde güvenlik seviyesini arttırmak tavsiye edilmektedir.

Web.config içerisine aşağıdaki tanımlama yapılarak XSS saldırılarına karşı filtreleme etkinleştirilebilmektedir. [1]

```
<system.webServer>
  <httpProtocol>
    <customHeaders>
      <add name="X-Xss-Protection" value="1; mode=block" />
    </customHeaders>
  </httpProtocol>
</system.webServer>
```

2.27.4. Content Sniffing Saldırılarının Önlenmesi

Media type sniffing veya MIME sniffing olarakta bilinen durumda saldırganlar veri akışındaki dosya biçimini saptamaya çalışmak için bayt akışının içeriğini incelemeye çalışmaktadırlar. Bu durumu önlemek için "**nosniff**" değeri ile "**X-Content-Type-Options**" başlık bilgisi kullanılabilmektedir.

```
<system.webServer>
```

```
2.              <httpProtocol>
3.                <customHeaders>
4.                  <add name="X-Content-Type-Options" value="nosniff" />
5.                </customHeaders>
6.              </httpProtocol>
7.            </system.webServer>
```

2.27.5. Referer-Policy'nin Tanımlanması

Bir websitesi üzerinden başka bir websitesine erişim yapıldığında, Referer headerı ile bulunulan web sitesi, ziyaret edilen siteye adresini iletir. Hedef site de yine Referer headerı ile bu ziyarete kaynak teşkil eden websitesini görebilir. Referer headerı sadece websitesi üzerindeki bir link tıklandığında değil; aynı zamanda stil, imaj ve script yüklemelerinde, form gönderimlerinde de yapılan isteğe eklenir. Referrer-Policy'nin düzgün yapılandırılmaması bazen hassas verilerin açığa çıkmasına neden olabilmektedir. Bu durumu önlemek için "**origin"** değeri ile birlikte aşağıdaki gibi bir Referrer-Policy tanımlaması yapılabilmektedir.

```
1.            <system.webServer>
2.              <httpProtocol>
3.                <customHeaders>
4.                  <add name="Referrer-Policy" value="origin" />
5.                </customHeaders>
6.              </httpProtocol>
7.            </system.webServer>
```

2.27.6. HTTP Strict Transport Security (HSTS) Başlığının Etkinleştirilmesi

Bu başlık bilgisi sayesinde bir web uygulamasının yalnızca HTTPS üzerinden iletişim kurması zorlanmaktadır. Dolayısıyla sunucu – istemci arasındaki ortadaki adam saldırıları önlenmektedir. "**max-age=31536000; includeSubDomains**" değerleri ile birlikte "**Strict-Transport-Security**" başlık bilgisi aşağıdaki gibi tanımlanarak bu durum sağlanmaktadır.

```
1.            <system.webServer>
2.              <httpProtocol>
3.                <customHeaders>
4.                  <add name="Strict-Transport-Security" value="max-age=31536000; includeSubDomains" />
5.                </customHeaders>
6.              </httpProtocol>
7.            </system.webServer>
```

3. Yazılım Geliştirme Sürecinde Alınması Gereken Önlemler

3.1. Parolaların Yönetilmesi

Hassas verilerin konfigürasyon dosyalarında saklanmaması gerekmektedir. Bunu sağlamak amacıyla **<appSettings>** elementinde **file** bileşeni yer almaktadır. Bu element ile hassas konfigürasyon ayarları harici bir dosyada saklanabilmektedir. Harici dosyanın source tree'de yer almadığından emin olunmalıdır. Bu şekilde "**git add ***" komutunun kullanılmasıyla harici dosya repository'e eklenmeyecektir.

ASP.NET çalışma zamanında harici dosya içeriğini Web.config'le birleştirerek sunmaktadır. Eğer runtime'da dosya bulunmazsa dosya yok sayılarak işlemlere devam edilmektedir.

Ayrıca IIS'in config dosyalarını sunmaması sebebiyle, harici dosyanın **.config** uzantısı ile kullanılması önerilmektedir.

Örnek Web.config dosyası aşağıdaki gibidir.

```
1.            </connectionStrings>
2.               <appSettings file="..\..\AppSettingsSecrets.config">
3.                  <add key="webpages:Version" value="3.0.0.0" />
4.                  <add key="webpages:Enabled" value="false" />
5.                  <add key="ClientValidationEnabled" value="true" />
6.                  <add key="UnobtrusiveJavaScriptEnabled" value="true" />
7.               </appSettings>
8.              <system.web>
```

Yukarıdaki örnekte bir takım yapılandırma bilgileri "**AppSettingsSecrets.config**" dosyasından getirilmektedir. Örnek "**AppSettingsSecrets.config**" dosyası aşağıda yer almaktadır.

```
<appSettings>
   <!-- SendGrid-->
   <add key="mailAccount" value="My mail account." />
   <add key="mailPassword" value="My mail password." />
   <!-- Twilio-->
   <add key="TwilioSid" value="My Twilio SID." />
   <add key="TwilioToken" value="My Twilio Token." />
   <add key="TwilioFromPhone" value="+12065551234" />

   <add key="GoogClientID" value="1.apps.googleusercontent.com" />
   <add key="GoogClientSecret" value="My Google client secret." />
</appSettings>
```

3.2. Connection Stringlerin Yönetilmesi

Visual Studio kullanılarak ASP.NET projesi oluşturulduğunda varsayılan olarak **LocalDb** kullanılmaktadır. Bu yapı rastgele bir şifre kullanmakta ve hassas bilgilerin kaynak kod tarafından kontrol edilmemesini sağlamaktadır.

Parolaların yönetilmesi bölümde bahsedilen **<appSettings>** elementindeki **file** bileşeni gibi Connection String'ler içinde **<connectionStrings>** elementinde **configSource** bileşeni yer almaktadır. Fakat **file** bileşeninin birleştirilerek kullanımının aksine **configSource** bileşeni ifadeyi değiştirerek (replace) kullanmaktadır.

Ek olarak, **AppSettingsSecrets.config** dosyasının aksine harici Connection String dosyaları, ana Web.config dosyası ile aynı dizinde yer almalıdır. Bu sebeple bu dosyaların kaynak repository'de yer almadığından emin olunmalıdır.

Ayrıca, best practice olarak production'da yer alan hassas bilgilerin test ve geliştirme sürecinde kullanılmaması önerilmektedir. Aynı bilgilerin kullanımı, bilgilerin yüksek olasılıkla ifşası ile sonuçlanacaktır.

Örnek ifade:

```
<connectionStrings configSource="ConnectionStrings.config"> </connectionStrings>
```

3.3. Webjobs Konsol Uygulamaları

Konsol uygulaması tarafından kullanılan app.config dosyası relative dizin yapısını desteklememekte absolute dizin kullanımı gerekmektedir.

Örnek Web.config dosyası aşağıdaki gibidir.

```
<configuration>
  <appSettings file="C:\secrets\AppSettingsSecrets.config">
    <add key="TwitterMaxThreads" value="24" />
    <add key="StackOverflowMaxThreads" value="24" />
    <add key="MaxDaysForPurge" value="30" />
  </appSettings>
</configuration>
```

3.4. Hassas Bilgilerin ve Connection Stringlerin Azure Ortamına Aktarılması

Uygulamalar Azure ortamına aktarılırken, güvenlik gerekçeleriyle harici **config** dosyaları aktarılmamaktadır. Azure Yönetim Portalı'ndan değerlerin el ile eklenmesi gerekmektedir. El ile eklenen app setting ve connection değerleri uygulama çalıştırıldığında Web.config dosyasındaki ilgili yerlerin üzerine yazılmaktadır.

On-premise sunucu kullanımında da Azure ortamında kullanılan adımlar izlenebilir. Fakat Azure ortamında otomatik olarak ortama eklenen değişkenlerin çekilmesi, bu değişkenlerin yapılandırma dosyasına eklenmesi ve ortam değişkeni bulunmadığı durumlarda yapılandırma dosyasında bulunan değerlerin kullanımı gibi işlemlerin ek olarak yazılması gerekmektedir.

Devops best practice olarak bu değerlerin (app setting ve connection değerleri) Azure ortamına aktarılması prosedürünün Azure powershell veya Chef Puppet benzeri frameworkler ile otomatize edilmesi önerilmektedir.

Kaynaklar

[1]https://www.mehmetince.net/http-security-headerlari-neden-ve-nasil-kullanilmalidir/
[2]https://docs.microsoft.com/tr-tr/aspnet/web-forms/overview/getting-started/getting-started-with-aspnet-45-web-forms/aspnet-error-handling
[3]https://devblogs.microsoft.com/aspnet/farewell-enableviewstatemac/
[4]https://www.troyhunt.com/owasp-top-10-for-net-developers-part-6/
[5]https://cwe.mitre.org/data/definitions/11.html
[6]https://docs.microsoft.com/tr-tr/aspnet/whitepapers/request-validation
[7]https://portswigger.net/kb/issues/00100280_asp-net-tracing-enabled
[8]https://docs.microsoft.com/en-us/previous-versions/aspnet/bb386420(v=vs.100)
[9]https://cwe.mitre.org/data/definitions/258.html
[10]https://cwe.mitre.org/data/definitions/13.html
[11]https://medium.com/@gokhansengun/http-cookie-nedir-hangi-ama%C3%A7larla-ve-nas%C4%B1l-kullan%C4%B1l%C4%B1r-5e9b9141fb7b
[12]https://blog.elmah.io/the-ultimate-guide-to-secure-cookies-with-web-config-in-net/
[13]https://owasp.org/www-community/attacks/Session_fixation
[14]https://dopeydev.com/session-fixation-attack-and-prevention-in-asp-net/
[15]https://vulncat.fortify.com/en/weakness?q=cacheRolesInCookie
[16]http://hwang.cisdept.cpp.edu/swanew/Code.aspx?m=HTTP-Header-Injection
[17]https://vulncat.fortify.com/en/detail?id=desc.config.dotnet.asp_dotnet_misconfiguration.header_checking_disabled#C%23%2fVB.NET%2fASP.NET
[18]https://blog.elmah.io/the-ultimate-guide-to-secure-cookies-with-web-config-in-net/
[19]https://www.informit.com/articles/article.aspx?p=351414&seqNum=4
[20]https://docs.microsoft.com/tr-tr/dotnet/api/system.web.security.formsauthentication.slidingexpiration?view=netframework-4.8
[21]https://docs.microsoft.com/en-us/dotnet/api/system.web.security.formsauthentication.cookiedomain?view=netframework-4.8
[22]https://techcommunity.microsoft.com/t5/iis-support-blog/session-state-and-session-cookies-best-practices/ba-p/714333
[23]https://docs.microsoft.com/tr-tr/aspnet/web-forms/overview/older-versions-security/introduction/forms-authentication-configuration-and-advanced-topics-cs
[24]https://odetocode.com/blogs/scott/archive/2006/03/22/asp-net-event-validation-and-invalid-callback-or-postback-argument-again.aspx
[25]https://msrc-blog.microsoft.com/2011/12/27/more-information-about-the-december-2011-asp-net-vulnerability/
[26]https://devblogs.microsoft.com/aspnet/cryptographic-improvements-in-asp-net-4-5-pt-1/
[27]https://docs.microsoft.com/en-us/aspnet/web-api/overview/security/enabling-cross-origin-requests-in-web-api
[28]https://gist.github.com/marcbarry/47644b4a43fbfb63ef54

- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/connection-strings-and-configuration-files
- https://docs.microsoft.com/en-us/aspnet/identity/overview/features-api/best-practices-for-deploying-passwords-and-other-sensitive-data-to-aspnet-and-azure
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/protecting-connection-information
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/privacy-and-data-security
- https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/ef/security-considerations
- https://docs.microsoft.com/en-us/previous-versions/aspnet/hh8x3tas(v=vs.100)
- https://docs.microsoft.com/en-us/previous-versions/aspnet/53tyfkaw(v=vs.100)
- https://docs.microsoft.com/en-us/previous-versions/aspnet/dtkwfdky(v=vs.100)
- https://www.c-sharpcorner.com/article/secure-web-application-via-web-config-file-in-asp-net-mvc/
- https://www.dotnet-guide.com/understanding-the-structure-and-content-of-web-config-file.html

Biznet Bilişim Sistemleri ve Danışmanlık Sanayi Tic. A.Ş.

Ticari Sicil No: 159433

İSTANBUL
Nida Kule Plaza,
Kozyatağı Mah.
Değirmen Sok. No:18
Kat:19 34742 Kozyatağı,
Kadıköy, İstanbul
+90 216 688 8182

ANKARA
ODTÜ Teknokent İkizler
Binası Üniversiteler Mah.
İhsan Doğramacı Bulvarı
No:35 B Blok Kat:106800
Çankaya / Ankara
+90 312 210 1177

DUBAI
SECURRENT ME FZ LLC
214, Building 12, DIC
502318, UAE - Dubai
+9 9714 390 16 46-49

LAHEY/ HOLLANDA
Penetra Cyber Security
Strawinskylaan 411
1077XX Amsterdam
The Netherlands
+31(0)70-2045180

CYBER WISE