OWASP Top 10 (2021) ve .NET Core ile Önlemler

1. Broken Access Control (Erişim Kontrol Eksiklikleri)

Sorun: Kullanıcıların, yetkili olmadıkları kaynaklara erişim sağlayabilmesi.

Broken Access Control, uygulamalarda yapılan yanlış veya yetersiz erişim kontrolü ayarları sonucu, yetkisiz kullanıcıların uygulama içerisinde yer alan kaynaklara erişmesine olanak sağlayan bir güvenlik açığıdır. Bu açık, yetkili olmayan bir kullanıcının sisteme veya uygulamaya erişim hakkı olmadığı halde, bu kaynaklara erişmesine veya bunları değiştirmesine neden olabilir.

Broken Access Control açığından korunmak için alınması gereken önlemler:

· **Erişim kontrolleri düzgün yapılandırılmalıdır,**

· **Token-based authentication kullanılmalıdır,**

· **İşlemlerin logları tutulmalı**

Çözümler:

- Role-based veya Policy-based yetkilendirme kullanın.
- Yetkisiz erişim denemelerini kayıt altına alın.

Kod Örneği: Policy-based yetkilendirme:

```
services.AddAuthorization(options =>
{
    options.AddPolicy("AdminOnly", policy =>
        policy.RequireClaim("Role", "Admin"));
});

[Authorize(Policy = "AdminOnly")]
public IActionResult GetAdminData()
{
    return Ok("Admin data");
}
```

2. Cryptographic Failures (Kriptografik Hatalar)

Sorun: Hassas verilerin şifrelenmeden veya zayıf algoritmalarla saklanması.

Cryptographic Failures, kriptografi kullanılarak korunan verilerin, yanlış bir şekilde şifrelendiği veya şifresinin çözüldüğü durumlardır. Bu tür hatalar, uygulamalarda kullanılan şifreleme algoritmalarının yanlış seçilmesi veya uygulanması, anahtar yönetimi hataları veya rastgele sayı üretimindeki eksiklikler nedeniyle oluşabilir. Cryptographic Failures, uygulamaların güvenliği için çok önemli bir risk faktörüdür, çünkü kötü niyetli bir saldırganın bu açıklardan yararlanarak, korunan verilere erişim sağlaması mümkün olabilir.

Cryptographic Failures açığından korunmak için alınması gereken önlemler:

· **Doğru şifreleme algoritmaları kullanılmalıdır**

· **Anahtar yönetimi hatalarından kaçınılmalıdır**

· **Rastgele sayı üretimi doğru yapılmalıdır**

· **Kriptografik anahtarların düzenli olarak yenilenmesi gerekmektedir**

Çözümler:

- AES-256 gibi güçlü algoritmalar kullanın.
- Şifreleme anahtarlarının güvenliğini sağlayın (Azure Key Vault gibi).

Kod Örneği: Veri şifreleme:

```
public static string EncryptData(string
plainText, string key)
{
    using var aes = Aes.Create();
    aes.Key = Encoding.UTF8.GetBytes(key);
    aes.GenerateIV();

    using var encryptor =
aes.CreateEncryptor(aes.Key, aes.IV);
    using var ms = new MemoryStream();
    using var cs = new CryptoStream(ms, encryptor,
CryptoStreamMode.Write);
    using var sw = new StreamWriter(cs);
    sw.Write(plainText);
    var iv = Convert.ToBase64String(aes.IV);
    var encrypted =
Convert.ToBase64String(ms.ToArray());

    return $"{iv}:{encrypted}";
}
```

3. Injection (Enjeksiyon Zafiyetleri)

Sorun: SQL enjeksiyonu, komut enjeksiyonu gibi girdi kontrollerinin eksikliği.

Injection, web uygulamalarında sıklıkla görülen bir güvenlik açığıdır. Bu açık, uygulamalarda kullanılan veri girişleri yoluyla, kötü niyetli kullanıcıların uygulama tarafından yürütülen veritabanı sorgularına, komutlara veya diğer işlevlere veri girişlerinde yapılan hatalı denetlemeler veya filtrelemeler nedeniyle zararlı kod enjekte etmesini sağlar. Saldırganlar, bu açıktan yararlanarak, uygulamaların veritabanını veya sunucularını ele geçirerek, kullanıcı bilgilerine veya diğer hassas verilere erişebilirler.

Injection açığından korunmak için alınması gereken önlemler:

· **Parametrelerin Doğrulanması**

· **SQL Parametreleştirme**

· **Kodlama Standartları**

· **WAF Kullanımı**

Çözümler:

- Parametrik sorgular veya ORM kullanın.
- Kullanıcı girdilerini her zaman doğrulayın.

Kod Örneği: Entity Framework ile SQL enjeksiyondan korunma:

```
var user = await _context.Users
    .FirstOrDefaultAsync(u => u.Username == username);
```

4. Insecure Design (Güvensiz Tasarım)

Sorun: Güvenlik mekanizmalarını tasarımda dikkate almamak.

Insecure Design (Güvensiz Tasarım), bir web uygulamasının tasarımında yapılan hatalar veya eksiklikler nedeniyle ortaya çıkan bir güvenlik açığıdır. Bu, uygulamanın tasarımında yapılan hataların, uygulamanın tüm yaşam döngüsü boyunca devam etmesine ve güvenliğini olumsuz etkilemesine neden olabilir. Güvensiz tasarım, uygulamanın özellikle kimlik doğrulama, yetkilendirme, veri gizliliği ve bütünlüğü gibi önemli güvenlik konularında hatalar içermesiyle ortaya çıkabilir.

Insecure Design açığından korunmak için alınması gereken önlemler:

· **Güvenli tasarım ilkeleri uygulanmalıdır**

· **Uygulama geliştirme ekibi, güvenlik odaklı bir yaklaşım benimsemelidir**

· **Güvenlik açıkları düzeltilmelidir**

· **Uygulama güvenliği için en iyi uygulamalar kullanılmalıdır**

Çözümler:

- Tehdit modellemesi yapın.
- Tüm girdilerde veri doğrulama uygulayın.

Kod Örneği: FluentValidation ile girdi doğrulama:

```
public class LoginRequestValidator : AbstractValidator<LoginRequest>
{
    public LoginRequestValidator()
    {
        RuleFor(x => x.Email).NotEmpty().EmailAddress();
        RuleFor(x => x.Password).NotEmpty().MinimumLength(8);
    }
}
```

5. Security Misconfiguration (Güvenlik Yapılandırma Hataları)

Sorun: Yanlış veya eksik sistem ayarları sebebiyle zafiyet oluşması.

Security Misconfiguration, bir uygulamanın ya da sistemlerin yanlış yapılandırılması veya yapılandırılmamış olması sonucu oluşan bir güvenlik açığıdır. Bu açık, sistemin, uygulamanın veya sunucuların güvenliğini sağlamak için gereken en iyi uygulamaların uygulanmamış veya eksik uygulanmış olmasından kaynaklanabilir.

Örnek olarak, bir web uygulaması sunucusunun yanlış yapılandırılması sonucu uygulama zafiyete uğrayabilir. Bu durumda, uygulama sunucusu için önerilen en iyi uygulamaların uygulanmamış olması nedeniyle, saldırganlar uygulama sunucusuna erişebilirler ve hassas verileri ele geçirebilirler.

Security Misconfiguration açığından korunmak için alınması gereken önlemler:

· **En iyi uygulamaları takip etmek**

· **Yazılımın güvenlik düzeyini kontrol etmek**

· **Güvenlik yamalarını ve güncellemeleri takip etmek**

· **Sistem yapılandırmasını güncellemek**

Çözümler:

- HTTPS zorunlu olsun.
- Varsayılan şifreler ve portları değiştirin.

Kod Örneği: HTTPS ve HSTS kullanımı:

```
public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
{
    app.UseHttpsRedirection();
    app.UseHsts();
}
```

6. Vulnerable and Outdated Components (Güvensiz ve Eski Bileşenler)

Sorun: Eski veya güvenlik zafiyeti olan kütüphaneler kullanılması.

Vulnerable and Outdated Components, bir uygulamada kullanılan üçüncü taraf bileşenlerin güncellenmemesi veya bilinen güvenlik açıklarına sahip olması nedeniyle oluşan bir güvenlik açığıdır. Birçok modern uygulama, üçüncü taraf bileşenlerin kullanımını içerir. Bu bileşenler, genellikle web uygulama çerçeveleri, veritabanı yönetim sistemleri, açık kaynak kütüphaneler, sunucu yazılımı ve diğer araçlar gibi yazılım bileşenleri olabilir. Bu bileşenlerde güvenlik açıkları keşfedilmesi halinde, uygulama açığa karşı savunmasız hale gelebilir.

Vulnerable and Outdated Components açığından korunmak için alınması gereken önlemler:

· **Bileşenleri izlemek**

· **Güncelleme politikaları oluşturmak**

· **Bileşenleri doğrulamak**

Çözümler:

- Paketlerin güncel olmasını sağlayın.
- Dependabot veya benzeri aracılar kullanın.

Kod Örneği: Paket güncelleme komutu:

```
dotnet add package PackageName --version x.x.x
```

7. Identification and Authentication Failures (Kimlik Doğrulama Eksiklikleri)

Sorun: Kötü tasarlanış kimlik doğrulama mekanizmaları.

Identification and Authentication Failures, bir kullanıcının kimliğinin doğrulanması veya yetkilendirilmesi sırasında yaşanan sorunlar nedeniyle oluşan bir güvenlik açığıdır. Bu tür bir açık, bir saldırganın bir kullanıcının kimliğini çalmasına veya sahte bir kimlik kullanarak uygulamaya erişmesine izin verebilir.

Örneğin, bir uygulama, kullanıcı adı ve parola kombinasyonu gibi temel kimlik doğrulama yöntemlerini kullanarak kimlik doğrulama yapabilir. Ancak, uygulama bu bilgileri doğru bir şekilde doğrulamazsa, saldırganların sahte kimlik bilgileri kullanarak sisteme giriş yapması mümkündür.

Identification and Authentication Failures açığından korunmak için alınması gereken önlemler:

· **Güçlü kimlik doğrulama yöntemleri kullanmak**
· **Kimlik doğrulama işlemlerini izlemek**
· **Kimlik bilgilerini şifrelemek**
· **Düzenli olarak kimlik doğrulama politikalarını kontrol etmek**
· **Çok faktörlü kimlik doğrulama yöntemlerini kullanmak**

Çözümler:

- ASP.NET Identity veya JWT kullanın.
- Parola karma algoritmaları (bcrypt) kullanın.

Kod Örneği: Parola hashleme:

```
public string HashPassword(string password)
{
    return BCrypt.Net.BCrypt.HashPassword(password);
}

public bool VerifyPassword(string password, string
hashedPassword)
{
    return            BCrypt.Net.BCrypt.Verify(password,
hashedPassword);
}
```

8. Software and Data Integrity Failures (Yazılım ve Veri Bütünlüğü Hataları)

Sorun: Gönderilen verinin manipüle edilmesi.

Yazılım ve veri bütünlüğü hataları, bir yazılım veya veri sisteminin beklenmeyen şekilde değiştirilmesi bozulması sonucu meydana gelen bir güvenlik açığıdır. Bu zafiyet, bir saldırganın yazılım veya veri sistemini hedef alarak sistemi istismar etmesine veya manipüle etmesine izin verebilir.

Yazılım ve veri bütünlüğü hataları, bir yazılım güncellemesi sırasında, yazılımın kötü amaçlı bir saldırgan tarafından değiştirilmesi veya bir saldırganın bir veri depolama ortamına kötü amaçlı yazılım yerleştirmesi gibi birçok farklı şekilde ortaya çıkabilir. Bu tür saldırılar sonucunda, saldırgan verileri çalabilir, verileri bozabilir veya sistemi kontrol etmeye başlayabilir.

· **Yazılımın veya verilerin beklenen kaynaktan olduğunu ve değiştirilmediğini doğrulamak için dijital imzalar veya benzer mekanizmaların kullanımı**

· **Npm veya Maven gibi kitaplıkların ve bağımlılıkların güvenilir depoları kullandığından emin olmak**

· **Kod ve yapılandırma değişiklikleri için bir inceleme süreci oluşturmak**

· **Yazılım güncellemelerini düzenli olarak yüklemek**

· **Güçlü erişim kontrolü uygulamak**

· **Güvenli yazılım geliştirme yöntemlerini kullanmak**

· **Yazılım ve veri yedeklemeleri oluşturmak**

Çözümler:

- Verileri dijital olarak imzalayın.
- CI/CD boru hatlarında güvenli imzalama kullanın.

Kod Örneği: JWT kullanarak veri imzalama:

```
var token = new JwtSecurityToken(
    issuer: "yourdomain.com",
    audience: "yourdomain.com",
    claims: claims,
    expires: DateTime.Now.AddHours(1),
    signingCredentials: credentials);
```

9. Security Logging and Monitoring Failures (Güvenlik İzleme ve Günlükleme Eksiklikleri)

Sorun: Günlükleme eksikliği sebebiyle zafiyetlerin fark edilmemesi.

Security Logging and Monitoring Failures, güvenlik olaylarının izlenmesi ve kaydedilmesi işlevlerinin yetersizliği veya hatalı yapılandırılması sonucu ortaya çıkan bir güvenlik açığıdır. Bu tür bir açık, kötü amaçlı aktivitelerin tespit edilememesi veya güvenlik olaylarına yanıt verilememesi gibi sonuçlara neden olabilir.

Security Logging and Monitoring Failures açığından korunmak için alınması gereken önlemler:

· **Güvenlik olaylarının izlenmesi ve kaydedilmesi için uygun araçlar kullanmak**

· **Günlük kayıtlarının düzenli olarak incelenmesi**

· **Uyarı ve alarm sistemleri kullanmak**

· **Güvenlik politikalarının ve prosedürlerinin düzenli olarak gözden geçirilmesi**

Çözümler:

- Serilog gibi bir günlükleme sistemi kurun.
- Önemli olayların alarmlarını ayarlayın.

Kod Örneği: Serilog entegrasyonu:

```
Log.Logger = new LoggerConfiguration()
    .WriteTo.Console()
    .WriteTo.File("logs.txt",    rollingInterval:
RollingInterval.Day)
    .CreateLogger();

Log.Information("Uygulama başladı.");
```

10. Server-Side Request Forgery (SSRF)

Sorun: Uygulamanın iç ağ istekleri yapmasını manipüle etmek.

Sunucu Taraflı İstek Sahtekarlığı (SSRF), bir saldırganın hedef sunucuda bir URL'ye istek gönderirken sahte bir kaynak IP adresi ve alan adı sağlayarak sunucunun kendi iç ağlarına veya diğer harici kaynaklara erişmesine izin veren bir web güvenliği açığıdır. Bir SSRF saldırısı, bir uygulamanın doğrulama sürecini atlayarak veya bir URL'de bir sunucu adresi veya IP adresi gibi güvenliği kontrol etmeyen bir parametre kullanarak gerçekleştirilebilir.

SSRF, hedef sunucu için ciddi bir güvenlik tehdidi oluşturur, çünkü saldırgan sunucuya istekler göndererek hassas verileri çalabilir, sunucunun kaynaklarını tüketebilir, sunucunun kontrolünü ele geçirebilir veya sunucunun çalışmasını bozabilir.

· **Giriş doğrulaması**

· **Güvenlik duvarı kurulumu**

· **Güvenli URL işleme**

· **Sunucu ayarlarının kontrol edilmesi**

Son olarak tüm zafiyetlerde korunmak için uygulanabilecek ortak yöntemler:

· **Profesyoneller tarafından güvenlik testlerinin yapılması,**

· **Logların tutulup profesyonel ekipler tarafından faaliyetlerin izlenmesi,**

· **Yazılım geliştirici ve kullanıcılara farkındalık artırma faaliyetleri yapılmasıdır.**

Çözümler:

- Gönderilen URL'leri doğrulayın.
- Erişim listeleri (allowlist) kullanın.