Owasp top 10

Broken Access Control Nedir

Broken Access Control (bozuk erişim kontrolü), bir web uygulamasının kullanıcıların yalnızca belirli kaynaklara (veriler, sayfalar, özellikler vb.) belirli bir izin seviyesine sahip olmalarını sağlayan bir mekanizmadır.

Broken Access Control, bu erişim kontrolünün yanlış yapılandırılmasından veya eksik uygulanmasından kaynaklanır. Bu durumda, bir saldırgan normalde erişemeyeceği kaynaklara erişebilir veya bu kaynaklara daha yüksek bir izin seviyesinde erişebilir.

Neden kaynaklanır ?

bir yazılımın ya da sistemin kullanıcıların erişim haklarını düzgün bir şekilde denetleyememesi durumudur. Bu güvenlik açığı, yetkisiz kullanıcıların erişmemesi gereken verilere veya işlevlere erişmesine olanak tanır. "Broken Access Control" zafiyetine sebeb olan durumlar şunlardır:

1. **Yanlış veya Eksik Erişim Denetimleri**: Erişim kontrollerinin doğru şekilde uygulanmaması, belirli kullanıcıların yetkili oldukları alanların dışında da işlemler yapabilmesine yol açar.
2. **Yanlış Yapılandırma:** Güvenlik politikalarının veya erişim denetimlerinin yanlış yapılandırılması, yetkisiz erişimlere neden olabilir.
3. **Güvenlik Politikalarının Zayıf Uygulanması**: Eğer bir sistemde güvenlik politikaları yeterince sıkı değilse veya bu politikalar tutarsız bir şekilde uygulanıyorsa, yetkisiz kullanıcılar erişim kazanabilir.
4. **Veri Doğrulama Eksikliği**: Kullanıcının kimliğini doğrulamak için gerekli olan doğrulama kontrollerinin eksik ya da zayıf olması, yetkisiz erişimlere olanak tanır.
5. **URL Manipülasyonu**: Erişim kontrollerinin düzgün uygulanmadığı durumlarda, bir saldırgan URL'leri manipüle ederek farklı kaynaklara yetkisiz bir şekilde erişebilir.
6. **Eksik veya Yanlış Rollere Dayalı Erişim Kontrolü (RBAC):** Kullanıcıların rollerine uygun olarak belirlenen erişim haklarının eksik ya da yanlış tanımlanması da bu tür zafiyetlere yol açabilir.

Bir örnek üzerinden açıklamak gerekirse:

Bir web uygulamanız var ve bu uygulama üzerinden kullanıcılar kendi profillerini düzenleyebiliyorlar. Kullanıcılar, profillerini düzenlemek için "profil düzenleme" sayfasına giderken, URL şu şekilde görünüyor:

```
https://example.com/profile/edit?user_id=123
```

Burada user_id=123 parametresi, kullanıcının kimlik numarasını temsil ediyor. Uygulama, bu kullanıcı kimliği üzerinden, doğru kullanıcının profilinin düzenlenmesini sağlıyor.

Eğer sunucu bu isteği sadece gelen user_id parametresine göre işler ve ek bir yetkilendirme kontrolü yapmazsa, saldırgan user_id=256 şeklinde değiştirilerek 256 kimlik numarasına sahip kullanıcının profilini düzenleme yetkisine sahip olur. Bu durumda, saldırgan yetkisi olmadan başka bir kullanıcının bilgilerini değiştirebilir.

Broken Access Control Türleri

"Broken Access Control" (Bozuk Erişim Kontrolü) farklı şekillerde ortaya çıkabilir ve çeşitli türleri vardır. Bu türler, sistemlerdeki farklı güvenlik açıklarını ve zafiyetleri ifade eder. İşte bazı yaygın türleri:

1. Doğrudan Referans (Insecure Direct Object References - IDOR):

- Bu türde, sistemdeki nesnelere (örneğin dosyalar, veri tabanındaki kayıtlar) doğrudan referans verilir ve erişim kontrolleri düzgün uygulanmaz. Bir saldırgan, URL'deki veya form verilerindeki bir parametreyi değiştirerek yetkisiz bir nesneye erişebilir.

Örnek: Bir kullanıcının hesabına ait `https://example.com/account?id=12345` URL'sinde, `id` parametresini değiştirerek başka bir kullanıcının hesabına erişmesi (`id=67890` gibi).

2. Yatay Yetki Yükseltme (Horizontal Privilege Escalation):

- Bu türde, aynı yetki düzeyindeki farklı kullanıcıların verilerine erişim sağlanır. Kullanıcılar arasında veri yalıtımı doğru şekilde uygulanmadığında ortaya çıkar.

Örnek: Bir kullanıcının, kendi profilini düzenlemek için bir URL'ye erişimi olduğunu varsayalım. Eğer bu kullanıcı, başka bir kullanıcının profilini düzenlemek için URL'deki kullanıcı ID'sini değiştirirse ve sistem bunu engellemezse, yatay yetki yükseltme gerçekleşir.

3. Dikey Yetki Yükseltme (Vertical Privilege Escalation):

- Bu türde, daha düşük yetkiye sahip bir kullanıcı, daha yüksek yetkiye sahip bir işlemi gerçekleştirmeye çalışır. Sistem, kullanıcının bu işlemi yapmaya yetkili olup olmadığını kontrol etmezse, saldırgan yetkilerini artırabilir.

Örnek: Bir kullanıcı, yönetici (admin) paneline erişmek için URL'de bir değişiklik yaparak (`/admin`) paneldeki işlemleri gerçekleştirebilir. Eğer sistem bu kullanıcıyı doğrulamazsa, dikey yetki yükseltme açığına sebep olur.

4. Kapsamlı veya Yetersiz Yetki Denetimleri (Overly Permissive or Insufficient Authorization Checks):

- Bu türde, sistemde belirli işlemleri gerçekleştirmek için gerekli olan yetki düzeyleri ya çok geniş tutulmuştur ya da yeterli denetim yapılmamaktadır. Bu durum, kullanıcıların yetkisiz işlemler yapmasına yol açabilir.

Örnek: Bir içerik yönetim sistemi (CMS), herhangi bir kullanıcıya tüm içeriklere erişim veya düzenleme yetkisi verebilir, bu da yetkisiz kişilerin içeriklerde değişiklik yapmasına neden olabilir.

5. Güvenli Olmayan Geçici Dizinler veya Dosyalar (Insecure Temporary Directories or Files):

- Geçici dosyalar veya dizinler yeterince korunmadığında, yetkisiz kullanıcılar bu dosyalara veya dizinlere erişebilir, değiştirebilir veya silebilir.

Örnek: Bir uygulama, kullanıcıların yüklediği dosyaları geçici bir dizinde depoluyorsa ve bu dizine herkes erişebiliyorsa, yetkisiz erişim mümkün hale gelir.

6. Güvenli Olmayan URL Yeniden Yazma (Insecure URL Redirection and Forwarding):

- Erişim kontrolleri düzgün bir şekilde yapılmadığında, kullanıcılar URL'leri değiştirerek veya manipüle ederek yetkisiz sayfalara yönlendirilebilir.

Örnek: Bir kullanıcı, oturum açma sonrasında belirli bir sayfaya yönlendirilmesi gerekiyorsa ve bu yönlendirme URL'si manipüle edilerek yetkisiz bir sayfaya yönlendirme yapılabiliyorsa, bu bir güvenlik açığıdır.

Bu türler, "Broken Access Control"un çeşitli biçimlerini temsil eder ve her biri, güvenlik açıklarını kapatmak için özel önlemler gerektirir. Erişim kontrollerinin doğru şekilde uygulanması ve sürekli olarak gözden geçirilmesi, bu tür güvenlik açıklarını önlemenin anahtarıdır.

Broken Access Control açıklarını önlemek için:

1. Erişim Kontrol Listesi (ACL) Kullanın: Her kullanıcıya ve role özel erişim hakları tanımlayın.
2. Yetki Denetimi Yapın: Her işlem öncesi kullanıcı yetkisini doğrulayın.
3. Kullanıcı Rolleri: Rolleri net belirleyin ve kullanıcılar arası erişim izolasyonu sağlayın.
4. Güvenli Nesne Referansları: Doğrudan ID kullanmak yerine, doğrulanmış tokenlar gibi güvenli yöntemler kullanın.
5. URL Manipülasyonunu Önleyin: Parametrelerin manipüle edilemeyeceğinden emin olun.
6. Merkezi Yetkilendirme: Yetkilendirme kararlarını merkezi bir noktada yönetin.
7. Güvenlik Testleri: Düzenli penetrasyon testleri yapın.
8. En Az Ayrıcalık: Kullanıcıların sadece gerekli minimum yetkilere sahip olmasını sağlayın.
9. İzleme ve Günlük Kayıtları: Anormal erişim denemelerini tespit etmek için izleyin.
10. Güvenli Kodlama: Geliştiricileri güvenli kodlama konusunda eğitin ve kod incelemeleri yapın.

Cryptographic Failures Nedir ?

Bu zafiyet türü *verilerin şifrelenmemesi* veya şifrelenen verilerin *eski, varsayılan, açığa çıkmış şifreleme algoritmaları* kullanılmasından ortaya çıkıyor. Geçerliliğini yitirmiş algoritmalar veya şifrelenmeden iletilen datalar zafiyet oluşmasına sebep olmaktadır.

Kredi kartı bilgileri, parolalar, kişisel kayıtlar, firma belgeleri ve daha birçok önemli bilgilerin şifrelenmesi gerekmektedir.

Neden Kaynaklanır ?

1. Zayıf veya Eskimiş Algoritmalar: MD5, SHA-1 gibi güvenli olmayan algoritmaların kullanılması.
2. Yanlış Anahtar Yönetimi: Kriptografik anahtarların güvenli olmayan ortamlarda saklanması, zayıf şifreleme anahtarları kullanılması.
3. Yetersiz Şifreleme: Verilerin şifrelenmeden saklanması veya iletilmesi.
4. Yanlış Şifreleme Modları: Yanlış blok şifreleme modlarının kullanılması, bu da veri bütünlüğünü tehlikeye atabilir.
5. Hatalı Uygulama: Kriptografik işlemlerin hatalı uygulanması, örneğin, şifreleme sırasında gerekli olan tuz (salt) değerinin kullanılmaması.

Örnek:

Bir web uygulamasında kullanıcı şifrelerinin MD5 ile hash edilmesi bir "Cryptographic Failure" örneğidir. MD5, modern hesaplama gücüyle kolayca kırılabilir, bu yüzden güçlü bir hash algoritması (örneğin, bcrypt) kullanılmalıdır.

Cryptographic Failures Açıkları nasıl önlenir ?

Güçlü ve Güncel Algoritmalar Kullanın: AES, SHA-256 gibi güvenilir algoritmalar tercih edilmelidir.

Güvenli Anahtar Yönetimi: Anahtarlar güvenli ortamlarda saklanmalı, güçlü şifreleme anahtarları kullanılmalıdır.

Şifreleme Zorunluluğu: Hassas veriler daima şifrelenmeli ve güvenli iletim yöntemleri (TLS gibi) kullanılmalıdır.

Doğru Şifreleme Modları: Uygun blok şifreleme modları (örneğin, CBC, GCM) kullanılmalıdır.

Düzenli Güncellemeler ve Testler: Kriptografik uygulamalar düzenli olarak gözden geçirilmeli ve güvenlik testlerinden geçirilmelidir.

İnjection zafiyetleri nedir ?

Injection zafiyetleri genellikle kullanıcıdan alınan, kontrol edilmeyen ya da önlem alınmayan verilerin komut olarak çalıştırılması ya da sorguya dahil edilmesi yüzünden oluşan zafiyetlerdir. İstatistiklere göre, şirketlerin % 28'i bu zafiyete maruz kalmaktadır.

Bu güvenlik açığı aşağıdaki saldırı vektörlerine bölünmüştür:

- SQL, LDAP, XPath sorguları aracılığıyla enjeksiyon
- İşletim Sistemlerindeki komutlarla enjeksiyon
- XML ayrıştırma yoluyla enjeksiyon
- SMTP başlıkları aracılığıyla enjeksiyon

Saldırgan bu vektörleri (*farklı bir veri göndererek sistemde komut çalıştırabilir*) kullanarak hem bir hesaba hem de erişmemesi gereken bu kaynağın tüm istemci veri tabanına erişim sağlayabilir. SQL veri tabanı türüne bağlı olarak farklı söz dizimi kullanarak veri tabanıyla çalışmak için yalnızca özel karakterler ve ek operatörler kullanarak hak yükseltme zafiyetini kullanmış olur. Sistemde ki etkinliğini artırır, hatta bütün organizasyona sızabilir. Bu zafiyet kullanıcıdan gelen verinin filtrelenmemesi veya kötü kodlardan temizlenmemesi sonucu ortaya çıkar.

Yaygın Injection Türleri:

1. SQL Injection:
    - Saldırgan, bir veritabanı sorgusuna kötü niyetli SQL kodu enjekte eder. Bu, veri tabanındaki verilere yetkisiz erişim sağlamaya, veri manipülasyonuna veya silmeye yol açabilir.
    - Örnek: "SELECT * FROM users WHERE username = 'admin' AND password = '' OR '1'='1';"
        - Bu sorgu, herhangi bir kullanıcı adı ve parola ile giriş yapılmasına olanak tanır.
2. Command Injection:
    - Saldırgan, işletim sistemi komutlarına kötü niyetli komutlar ekleyebilir. Bu, sunucu üzerinde yetkisiz komutların çalıştırılmasına neden olabilir.
    - Örnek: http://example.com/delete?file=filename.txt;rm%20-rf%20/
        - Bu URL, sunucuda dosya silme komutunun yanı sıra tüm dosyaların silinmesine neden olabilir.

3. LDAP Injection:
    - LDAP sorgularına kötü niyetli girdi enjekte ederek, kimlik doğrulama bypassı veya veri sızdırma gibi saldırılar gerçekleştirilebilir.
    - Örnek: "(&(user=*)(password=*))"
        - Bu, tüm kullanıcıların bilgilerinin çekilmesine neden olabilir.
4. XPath Injection:
    - XML verilerini işleyen sorgulara kötü niyetli girdilerin enjekte edilmesiyle XML verilerinin yetkisiz erişim veya manipülasyonuna yol açabilir.
    - Örnek: "//users/user[username/text()='admin' or '1'='1']"
        - Bu, saldırganın XML tabanlı veri yapısını manipüle etmesine olanak tanır.
5. NoSQL Injection:
    - NoSQL veri tabanlarına yönelik olarak yapılan bu saldırıda, kötü niyetli girdiler NoSQL sorgularına enjekte edilir.
    - Örnek: {"$ne": null, "$ne": ""}
        - Bu, herhangi bir kimlik doğrulama kontrolünü atlamak için kullanılabilir.
6. HTML/JavaScript Injection (XSS):
    - Kullanıcı girdisi olarak alınan HTML veya JavaScript kodunun doğrudan web sayfasına enjekte edilmesiyle, saldırganlar tarayıcıda zararlı kod çalıştırabilir.
    - Örnek: <script>alert('XSS');</script>
        - Bu, bir kullanıcı ziyaret ettiğinde tarayıcıda zararlı kodun çalışmasına neden olur.

Nasıl Önlenir ?

1. Girdi Doğrulama: Kullanıcıdan gelen girdileri doğru bir şekilde doğrulayın ve yalnızca beklenen formatta veri kabul edin.
2. Prepared Statements (Hazırlanmış İfadeler): SQL sorgularında, kullanıcı girdilerinin güvenliğini sağlamak için hazırlanan ifadeleri kullanın.
3. Stored Procedures (Saklı Yordamlar): SQL komutlarının güvenli bir şekilde çalışmasını sağlamak için saklı yordamları tercih edin.

4. Parametrik Sorgular: Sorgu yaparken parametreleri doğrudan sorguya eklemek yerine, bu parametreleri güvenli bir şekilde bağlayın.
5. Çıktı Kodlama: Web uygulamalarında, kullanıcı girdilerini sayfa çıktısına eklerken uygun kodlamayı (örneğin HTML, JavaScript kodlaması) kullanın.
6. En Az Ayrıcalık İlkesini Uygulayın: Veritabanı kullanıcılarına ve uygulamalara sadece gerekli olan minimum yetkileri verin.
7. Güvenlik Duvarları: Web uygulama güvenlik duvarları (WAF) kullanarak injection saldırılarını tespit edin ve engelleyin.
8. Düzenli Güvenlik Testleri: Uygulamanızı düzenli olarak güvenlik testlerinden geçirerek injection zafiyetlerini tespit edin ve giderin.

Insecure Design Nedir ?

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.

Örnek bir zafiyetli senaryoda bir online bankacılık sistemi, müşterilerine hesapları üzerinden para transferleri, hesap bakiyesi sorgulamaları ve diğer finansal işlemleri yapma imkanı sunarken, güvenlik açısından ciddi eksiklikler içermektedir. Sistem, çok faktörlü kimlik doğrulama (2FA) veya otomatik şüpheli işlem algılama gibi temel güvenlik önlemlerini ve savunma mekanizmalarını tasarım aşamasında göz ardı etmiştir. Bu durum, kötü niyetli bir saldırganın, phishing veya keylogging gibi yöntemlerle bir kullanıcının kimlik bilgilerini ele geçirmesine ve ardından bu bilgileri kullanarak sisteme rahatça erişmesine olanak tanır. Sistem, anormal işlem davranışlarını tespit etmekte yetersiz kaldığı için, saldırgan hesaptan büyük miktarlarda para transferi yapabilir. Bu güvenlik zaafiyeti, kullanıcıların ciddi finansal kayıplar yaşamasına ve sistemdeki diğer kullanıcıların benzer saldırılara maruz kalmasına neden olabilir.

Insecure Design Neden Kaynaklanır ?

1. Güvenlik Bilincinin Eksikliği:

- Tasarım sürecine dahil olan geliştiricilerin, mimarların ve proje yöneticilerinin güvenlik konularında yeterli bilgiye sahip olmaması. Güvenlik bilincinin eksik olması, kritik güvenlik önlemlerinin göz ardı edilmesine yol açabilir.

2. Güvenlik Gereksinimlerinin Belirlenmemesi:

- Projenin başlangıç aşamasında güvenlik gereksinimlerinin net bir şekilde tanımlanmaması veya bu gereksinimlerin proje boyunca dikkate alınmaması. Sonuç olarak, sistemin güvenliğine yönelik temel zafiyetler ortaya çıkabilir.

3. Zaman ve Maliyet Baskıları:

- Projelerin zamanında tamamlanması ve maliyetlerin düşük tutulması için güvenlik önlemleri ihmal edilebilir. Bu tür baskılar, güvenlik testlerinin yapılmaması veya eksik yapılması gibi durumlara yol açabilir.

4. Yanlış Varsayımlar:

- Kullanıcı davranışları, sistem etkileşimleri veya çevresel faktörler hakkında yanlış veya yanıltıcı varsayımlar yapılması. Örneğin, tüm kullanıcıların güvenilir olduğu varsayımıyla, gerekli güvenlik kontrolleri atlanabilir.

5. Tehdit Modelleme Eksikliği:

- Tasarım aşamasında potansiyel tehditlerin yeterince analiz edilmemesi veya göz ardı edilmesi. Tehdit modellemesi yapılmadığında, sistemin zayıf noktaları belirlenemez ve bu da güvenlik zafiyetlerine yol açabilir.

6. Karmaşık veya Kötü Yapılandırılmış Sistemler:

- Aşırı karmaşık veya kötü yapılandırılmış sistemler, güvenlik açıklarını artırır. Karmaşık sistemler, güvenlik testlerini zorlaştırır ve açıkların gözden kaçmasına neden olabilir.

7. Güvenli Geliştirme Pratiklerinin Eksikliği:

- Geliştiricilerin güvenli kodlama pratiklerini takip etmemesi veya bu konularda eğitim almamış olması. Güvenli olmayan geliştirme pratikleri, tasarım aşamasında kritik hatalara neden olabilir.

8. Güvenlik Testlerinin İhmal Edilmesi:

- Tasarım sürecinde güvenlik testlerinin yapılmaması veya yeterince kapsamlı yapılmaması. Bu durum, güvenlik açıklarının tespit edilememesine ve sistemin güvensiz bir şekilde piyasaya sürülmesine yol açabilir.

Nasıl Önlenir ?

Güvenlik Tasarım Prensiplerini Uygulayın:

- "En Az Ayrıcalık" ve "Savunma Derinliği" gibi güvenlik prensiplerini tasarım aşamasında uygulayın.

Güvenlik Gereksinimlerini Belirleyin:

- Proje başlangıcında güvenlik gereksinimlerini net bir şekilde tanımlayın ve tasarımı buna göre yapın.

Tehdit Modelleme:

- Olası güvenlik tehditlerini belirlemek için tehdit modelleme yapın ve bunlara karşı önlem alın.

Güvenlik Testlerini Entegre Edin:

- Tasarım ve geliştirme aşamalarında güvenlik testlerini entegre ederek, potansiyel zafiyetleri erken aşamada tespit edin.

Güvenlik Eğitimleri:

- Geliştiricilere ve tasarımcılara güvenli tasarım prensipleri hakkında eğitim verin.

Düzenli Tasarım Gözden Geçirmeleri:

- Tasarım aşamasında düzenli olarak güvenlik gözden geçirmeleri yaparak, potansiyel zafiyetleri belirleyin ve düzeltin.

Security Misconfiguration Nedir ?

Security Misconfiguration (Güvenlik Yanlış Yapılandırması), bir sistemin, uygulamanın veya ağın güvenlik ayarlarının eksik, yanlış veya hatalı yapılandırılması durumunu ifade eder. Bu tür yapılandırma hataları, sistemlerin güvenlik açıklarına maruz kalmasına neden olur ve saldırganların bu açıklardan faydalanarak sisteme yetkisiz erişim elde etmesine, verileri çalmasına veya sistemi manipüle etmesine yol açabilir.

Security Misconfiguration Neden Ortaya Çıkar ?

1. **Varsayılan Ayarların Kullanılması:**
    - Varsayılan kullanıcı adı, parola veya ayarların değiştirilmemesi. Bu, saldırganların yaygın bilinen varsayılan kimlik bilgilerini kullanarak sisteme erişmesine olanak tanır.
2. **Güvenlik Yamalarının Uygulanmaması:**
    - Yazılım güncellemelerinin veya güvenlik yamalarının zamanında uygulanmaması, bilinen zafiyetlerin kullanılmasına neden olabilir.
3. **Yanlış Yapılandırılmış Güvenlik Politikaları:**

- Erişim kontrollerinin yanlış yapılandırılması, gereksiz açık portlar, güvenlik duvarı kurallarının eksik veya hatalı olması gibi sorunlar.

4. **Hassas Bilgilerin Yanlış Korunması:**
   - Hassas bilgilerin (şifreler, API anahtarları, kimlik doğrulama bilgileri) yanlış yerlerde saklanması veya korunmaması. Örneğin, bir dosyada açık metin olarak saklanan şifreler.
5. **Eksik veya Yanlış Ayarlanmış SSL/TLS Sertifikaları:**
   - SSL/TLS sertifikalarının eksik veya hatalı yapılandırılması, şifreli bağlantıların güvenliğini tehlikeye atabilir.
6. **Aşırı İzinler:**
   - Gereksiz yetkilerin verilmesi veya kullanıcıların fazla ayrıcalıklara sahip olması. Bu, bir saldırganın sisteme erişmesi durumunda daha fazla hasar vermesine neden olabilir.
7. **Yanlış Yapılandırılmış Sunucu ve Servisler:**
   - Web sunucuları, veritabanı sunucuları, bulut servisleri gibi altyapıların yanlış yapılandırılması, güvenlik açıklarına yol açabilir.
8. **Eksik Güvenlik İzleme:**
   - Güvenlik olaylarının izlenmemesi veya logların yetersiz tutulması, saldırıların fark edilmemesine neden olabilir.

Birkaç örnek ile açıklanırsa :

- **Varsayılan Parolaların Değiştirilmemesi: Bir router'ın varsayılan "admin/admin" kimlik bilgileri ile bırakılması, saldırganların bu cihazı kolayca ele geçirmesine neden olabilir.**
- **Açık Yönetim Panelleri: Web uygulamasının yönetim panelinin, internete açık bir şekilde bırakılması ve yeterli kimlik doğrulama olmaması.**
- **Yanlış Ayarlanmış Güvenlik Duvarı: Gereksiz portların açık bırakılması, saldırganların bu portlar üzerinden sisteme erişmesine olanak tanır.**

Nasıl Önlenir ?

1. Varsayılan Ayarların Değiştirilmesi:
    - Tüm varsayılan kullanıcı adı, parola ve ayarları güvenli seçeneklerle değiştirin.
2. Düzenli Güncellemeler ve Yamalar:
    - Sistemleri ve yazılımları düzenli olarak güncelleyin ve güvenlik yamalarını zamanında uygulayın.
3. Güvenlik Politikalarının Gözden Geçirilmesi:
    - Erişim kontrollerini ve güvenlik politikalarını düzenli olarak gözden geçirin ve yanlış yapılandırmaları düzeltin.
4. Şifreleme Kullanımı:
    - Hassas verilerin şifreli bir şekilde saklandığından ve iletildiğinden emin olun. SSL/TLS sertifikalarının doğru şekilde yapılandırıldığını kontrol edin.
5. Aşırı İzinlerin Kısıtlanması:
    - Kullanıcı ve sistem izinlerini "en az ayrıcalık" ilkesi doğrultusunda ayarlayın.
6. Güvenlik İzleme ve Denetim:
    - Güvenlik olaylarını izleyin ve logları düzenli olarak inceleyin. Yanlış yapılandırmaları tespit etmek için düzenli denetimler yapın.
7. Otomatik Yapılandırma Araçları Kullanımı:
    - Otomatik güvenlik yapılandırma araçları kullanarak, hatalı yapılandırmaları tespit edin ve düzeltin.

Vulnerable and Outdated Components Nedir ?

Sistemlerimizde kullandığımız pek çok program vardır. Bunlar çoğunlukla sistemin güvenli ve performanslı olması gibi kaygılarla başvurulan ürünlerdir. Sistemin güvenli olması ve stabil çalışması gibi sebeplerle aldığımız bu önlemler bazen tam tersi sonuçlara yol açabilir. Bu sorunlardan biri de Vulnerable and Outdated Components olarak adlandırılan ve Türkçeye Savunmasız ve Güncel Olmayan Bileşenler şeklinde çevrilen zafiyettir. Zafiyet 2021 yılında OWASP Top 10 listesinde 6. sıraya yerleşerek günümüzde ne kadar yaygın olduğunu kanıtlamıştır.

Zafiyetin yıkıcı etkisine, 2017 yılında açık kaynaklı Apache Struts yazılımında keşfedilen remote code execution zafiyetinin yetkililer tarafından vaktinde kapatılamaması ile 143 milyon kişinin hassas verilerinin sızdırıldığı Equifax olayı örnek gösterilebilir.

Zafiyet, artık desteklenmeyen, eski veya zafiyet barındırdığı bilinen bileşenlerin kullanılmasıyla ortaya çıkar. Saldırganlar zafiyeti sömürmek için sistemlerde kullanılan bileşenleri ve bunların sürümlerini öğrenmeye çalışır. Zafiyetli herhangi bir bileşen keşfedildiğindeyse bunu sömürmek artık çok kolaydır.

Bazen güvenlik amacıyla üretilmiş yazılımlar da zafiyet içerebilir. Örneğin Nessus'un CVE'de listelenen pek çok güvenlik açığı vardır. Programlar bu açığı kapatacak yamalar yayınlamadığı sürece, programın kullanıldığı her sistem risk altındadır. Bu sebeple düzenli olarak, kullanılan ürünlerin zafiyet durumları kontrol edilmeli ve varsa yamalar hemen uygulanmalıdır. Uygulamalardaki güvenlik açıkları uygulama ile aynı izinlere sahip olduğu için uygulamanın yetkileri arttıkça risk de artar.

Vulnerable and Outdated Components Zafiyetlerinin Nedenleri:

1. **Güncelleme Eksiklikleri:**
    - Yazılım bileşenlerinin ve kütüphanelerinin güncellenmemesi. Güvenlik güncellemeleri ve yamaları zamanında uygulanmadığında, bilinen zafiyetler açık kalır.
2. **Yazılım Bağımlılıklarının Yönetilmemesi:**
    - Projelerde kullanılan üçüncü taraf kütüphanelerin ve bağımlılıkların güncellenmemesi veya denetlenmemesi.
3. **Eski Versiyonların Kullanımı:**
    - Eski ve artık desteklenmeyen yazılım sürümlerinin kullanılmaya devam edilmesi. Bu sürümler genellikle güvenlik açıkları içerir ve yeni güvenlik özelliklerini desteklemez.
4. **Güvenlik Açıklarının Bilinmemesi:**
    - Kullanılan bileşenlerin mevcut güvenlik açıklarından habersiz olunması. Bileşenlerin güvenlik durumunun düzenli olarak izlenmemesi.
5. **Yetersiz Güvenlik Testleri:**
    - Güncellenmiş bileşenlerin güvenlik testlerinin yapılmaması veya testlerin yetersiz olması. Bu, bileşenlerdeki zafiyetlerin fark edilmemesine neden olabilir.

Örnekleri ve Çeşitleri :

- Güncel Olmayan Kütüphaneler: Bir web uygulamasında eski bir versiyon kullanılan şifreleme kütüphanesi, bilinen bir güvenlik açığı taşıyabilir.
- Desteklenmeyen Bileşenler: Bir e-ticaret uygulamasında, artık güncellenmeyen bir açık kaynaklı ödeme modülü kullanılması.
- Zayıf Bağımlılıklar: Bir yazılımda kullanılan bir bağımlılığın güvenlik açığı içermesi ve bu bağımlılığın güncellenmemesi.

Vulnerable and Outdated Components Zafiyetlerinin Önlenmesi :

1. **Düzenli Güncellemeler:**
    - Kullanılan yazılım bileşenlerinin ve kütüphanelerinin güncellemelerini düzenli olarak yapın ve güvenlik yamalarını zamanında uygulayın.
2. **Güvenlik Bilgilendirmelerini Takip Edin:**
    - Kullandığınız bileşenlerin güvenlik güncellemeleri ve açıkları hakkında bilgi sağlayan kaynakları düzenli olarak takip edin.
3. **Bağımlılık Yönetimi Araçları Kullanın:**
    - Bağımlılıkların güvenlik açıklarını tarayan ve güncellemeleri takip eden araçları (örneğin, Dependabot, Snyk) kullanın.
4. **Desteklenmeyen Bileşenlerden Kaçının:**
    - Desteklenmeyen veya eski versiyonları kullanmaktan kaçının. Desteklenen ve güvenlik güncellemeleri sunan bileşenlere geçiş yapın.
5. **Güvenlik Testleri ve Denetimler:**
    - Yazılımın güvenlik testlerini ve denetimlerini düzenli olarak gerçekleştirin. Güncellenmiş bileşenlerle uyumlu testler yapın.
6. **Kaynak Kodunun İncelenmesi:**
    - Kullanılan bileşenlerin ve bağımlılıkların kaynak kodunu inceleyerek güvenlik açıklarını değerlendirin.

Identification and Authentication Failures Nedir ?

Identification and Authentication Failures (Kimlik Belirleme ve Kimlik Doğrulama Hataları), bir sistemin kullanıcıların kimliğini doğru şekilde tanımlayamaması veya doğrulayamaması durumunu ifade eder. Bu tür hatalar, bir kullanıcının yetkisiz olarak sisteme erişmesine, hassas verilere ulaşmasına veya başka güvenlik ihlallerine yol açabilir.

Kimlik Belirleme ve Kimlik Doğrulama Süreçleri:

- **Kimlik Belirleme (Identification):** Kullanıcının kendini tanıtması sürecidir. Genellikle kullanıcı adı veya kimlik numarası gibi bir bilgi sağlar.
- **Kimlik Doğrulama (Authentication):** Kullanıcının sağladığı kimlik bilgilerini doğrulama sürecidir. Bu, genellikle şifre, biyometrik veri veya çok faktörlü kimlik doğrulama gibi yöntemlerle yapılır.

En yaygın tanımlama ve kimlik doğrulama hataları şunlardır:

1. **Zayıf veya Tekrar Kullanılan Parolalar:**
    - **Tanım:** Kullanıcılar, "123456" veya "password" gibi basit ve tahmin edilmesi kolay parolalar kullanabilirler. Bu tür parolalar, saldırganların hesaplara erişim sağlamasını kolaylaştırır.
    - **Örnek:** Bir kullanıcı "123456" şifresi ile giriş yapıyorsa, saldırganlar bu şifreyi tahmin etmekte zorluk çekmezler.
2. **Kaba Kuvvet Saldırıları:**
    - **Tanım:** Saldırganlar, bir şifreyi kırmak için tüm olası karakter kombinasyonlarını denerler. Şifre zayıfsa, bu saldırı türü başarılı olabilir.
    - **Örnek:** Bir saldırgan, "password123" şifresini kırmak için tüm olası kombinasyonları deneyerek bu şifreye ulaşabilir.
3. **Kimlik Bilgisi Doldurma:**
    - **Tanım:** Saldırganlar, çalınan kullanıcı adları ve parolalarla farklı web sitelerine giriş yapmayı denerler. Eğer web sitesi parolayı sızdırmışsa, saldırgan bu yöntemle erişim sağlayabilir.
    - **Örnek:** Bir kullanıcı, "username

" kombinasyonunu farklı sitelerde denediğinde, bu bilgiler başka bir sitede de geçerli olabilir.

4. **Eksik veya Zayıf Çok Faktörlü Kimlik Doğrulama (MFA):**

- **Tanım:** MFA, kullanıcıların kimliklerini doğrulamak için iki veya daha fazla yöntem sunmalarını gerektirir. Eksik veya zayıf MFA, saldırganların yetkisiz erişim sağlamasını kolaylaştırır.
- **Örnek:** Bir hesap sadece şifre ile korunuyorsa ve SMS doğrulama gibi ek bir güvenlik adımı yoksa, saldırganlar şifreyi ele geçirdiklerinde hesaba erişim sağlayabilirler.

5. **Geçersiz Yönlendirmeler ve Yönlendirmeler:**
   - **Tanım:** Bu açık, kullanıcıları kötü amaçlı web sitelerine yönlendirme riskini içerir. Saldırganlar, kullanıcıların kimlik bilgilerini çalmak veya kötü amaçlı yazılım yüklemek için bu yönlendirmeleri kullanabilir.
   - **Örnek:** Bir kullanıcı, güvenilir görünen ancak kötü amaçlı bir siteye yönlendirilir ve bu sitede kimlik bilgilerini girerse, saldırgan bu bilgileri çalabilir.

Nasıl Önlenir :

1. **Güçlü Şifre Politikaları:**
   - Kullanıcıların güçlü, karmaşık şifreler seçmesini zorunlu kılın. Şifre uzunluğu ve karmaşıklığı için belirli standartlar belirleyin.
2. **Çok Faktörlü Kimlik Doğrulama (MFA):**
   - Kimlik doğrulama sürecine ek bir güvenlik katmanı eklemek için MFA uygulayın. SMS, e-posta doğrulama, biyometrik veriler gibi yöntemler kullanılabilir.
3. **Güvenli Şifre Saklama:**
   - Şifreleri güvenli bir şekilde saklamak için güçlü şifreleme yöntemleri kullanın, örneğin bcrypt veya Argon2.
4. **Kimlik Bilgilerini Koruma:**
   - Kimlik bilgilerini güvenli bir şekilde saklayın ve veri sızıntılarına karşı önlemler alın.
5. **Oturum Yönetimi Politikaları:**
   - Oturum sürelerini sınırlayın ve kullanıcılar oturum kapatma işlemi yapmadığında oturumları otomatik olarak kapatın.
6. **Kimlik Doğrulama Mekanizmalarının Test Edilmesi:**
   - Kimlik doğrulama ve yetkilendirme mekanizmalarını düzenli olarak test edin ve güvenlik açıklarını giderin.

Software and Data Integrity Failures Nedir?

Software and Data Integrity Failures, yazılım veya verinin yetkisiz veya beklenmedik şekilde değiştirilmesi durumunda ortaya çıkan güvenlik açıklarını ifade eder. Bu tür güvenlik açıkları, kötü niyetli kişilerin sistem üzerinde kontrol sahibi olmasına veya hassas bilgileri ele geçirmesine olanak tanır.

Örnekler ve Gerçek Hayattan Senaryolar

1. **SolarWinds Orion Saldırısı:**
    - **Açıklama:** SolarWinds, büyük bir IT yönetim yazılımı sağlayıcısıdır. Aralık 2020'de, SolarWinds'in Orion platformuna yapılan bir tedarik zinciri saldırısı keşfedildi. Saldırganlar, SolarWinds'in güncelleme mekanizmasını manipüle ederek zararlı bir güncelleme sundular. Bu güncelleme, birçok önemli kuruma yayıldı ve ciddi güvenlik ihlallerine neden oldu.
    - **Sonuç:** Binlerce kurum etkilendi, saldırganlar uzun süre fark edilmeden sistemlere erişim sağladı.
    - **Önleme:** Yazılım güncellemelerinin bütünlük kontrolleri ile doğrulanması ve dijital imzaların kullanılması.
2. **Equifax Veri İhlali:**
    - **Açıklama:** 2017'de gerçekleşen bu olayda, Equifax'ın web uygulamasındaki bir güvenlik açığı nedeniyle milyonlarca kişiye ait hassas bilgi çalındı.
    - **Sonuç:** Yaklaşık 147 milyon Amerikalının sosyal güvenlik numaraları, doğum tarihleri ve adresleri çalındı.
    - **Önleme:** Yazılım güncellemelerinin ve yamaların düzenli olarak uygulanması, güvenlik açığı tarama araçlarının kullanılması.
3. **NotPetya Fidye Yazılımı:**
    - **Açıklama:** 2017'de NotPetya adlı fidye yazılımı, bir muhasebe yazılımının güncelleme mekanizmasını kullanarak yayıldı. Bu yazılım, özellikle Ukrayna'daki birçok kurumu hedef aldı.
    - **Sonuç:** Global ölçekte büyük zararlar verdi ve birçok büyük şirketin operasyonlarını durdurdu.
    - **Önleme:** Yazılım tedarik zinciri güvenliğinin sağlanması, sistemlerin yedeklenmesi ve güncellemelerin doğrulanması.

Önleme Yöntemleri ve Güvenlik Tedbirleri

1. **Dijital İmzalar:**
    - **Açıklama:** Yazılım veya verinin beklenen kaynaktan geldiğini ve değiştirilmediğini doğrulamak için dijital imzalar kullanılır.
    - **Örnek:** Bir yazılım güncellemesi indirildiğinde, bu güncellemenin güvenilir bir kaynaktan geldiğini doğrulamak için dijital imza kontrolü yapılmalıdır.
2. **Güvenilir Depolar:**

- **Açıklama:** Kütüphaneler ve bağımlılıkların güvenilir depolardan alınması gerekir. Daha yüksek risk profiline sahip durumlar için, güvenilir bir iç depo kullanılabilir.
- **Örnek:** Yazılım geliştirme sürecinde kullanılan tüm kütüphanelerin güvenilir kaynaklardan indirildiğini doğrulamak için araçlar (örneğin, OWASP Dependency-Check) kullanmak.

3. **Kod ve Konfigürasyon Değişikliklerinin İncelenmesi:**
    - **Açıklama:** Yazılım hattına kötü niyetli kod veya yapılandırma eklenme riskini en aza indirmek için bir inceleme süreci oluşturulmalıdır.
    - **Örnek:** Kod incelemeleri ve otomatik testlerin yapılması, kod değişikliklerinin doğrulanmasını sağlar.
4. **CI/CD Pipeline Güvenliği:**
    - **Açıklama:** Sürekli entegrasyon ve sürekli dağıtıma (CI/CD) uygun şekilde ayrıldığından, yapılandırıldığından ve erişim kontrolüne sahip olduğundan emin olunmalıdır.
    - **Örnek:** CI/CD süreçlerinin sadece yetkili kişiler tarafından erişilebilir olmasını sağlamak ve tüm değişikliklerin izlenebilir olmasını sağlamak.
5. **Serileştirilmiş Verilerin Güvenliği:**
    - **Açıklama:** İmzalanmamış veya şifrelenmemiş serileştirilmiş verilerin, bir tür bütünlük kontrolü veya dijital imza olmadan güvenilmeyen istemcilere gönderilmemesi.
    - **Örnek:** Web uygulamalarında JSON Web Tokens (JWT) kullanırken, token'ların güvenli bir şekilde imzalandığından emin olmak.

Software and Data Integrity Failures Çözüm Önerileri ve En İyi Uygulamalar

1. **Katmanlı Güvenlik Yaklaşımı:**
    - **Açıklama:** Tek bir güvenlik önlemi yeterli olmayabilir. Bu nedenle, katmanlı bir güvenlik yaklaşımı benimsenmelidir.
    - **Örnek:** Ağ güvenlik duvarları, uygulama güvenlik duvarları (WAF), güvenli kodlama uygulamaları ve güvenlik taramaları gibi çeşitli güvenlik önlemleri birlikte kullanılmalıdır.
2. **Güvenlik Eğitimleri:**
    - **Açıklama:** Tüm geliştiricilerin ve ilgili personelin güvenlik konusunda eğitim alması sağlanmalıdır.
    - **Örnek:** OWASP eğitimleri, güvenlik konferansları ve çevrimiçi kurslar.
3. **Sürekli İzleme ve Denetim:**
    - **Açıklama:** Güvenlik kontrollerinin etkinliğinin sürekli izlenmesi ve düzenli denetimlerin yapılması.
    - **Örnek:** Güvenlik bilgi ve olay yönetimi (SIEM) araçları kullanarak güvenlik olaylarını izlemek ve analiz etmek.

Security Logging and Monitoring Failures Nedir ?

Saldırganlar saldırıya geçmeden önce uzun süreler boyunca vakitlerini sistem hakkında bilgiler elde etmeye ayırırlar. Bilgi elde etme sürecinde pasif veya aktif yöntemler kullanılabilir. Aktif yöntemlerin pasif yöntemlerden farkının karşı tarafla etkileşime geçilmesi olduğunu biliyorduk. Etkileşime geçmek aslında artık karşı taraf tarafından fark edilebilir olmak demektir. Hedef sistemde loglama ve monitoring işlemleri düzgünce yapıldığında saldırganların bilgi toplama süreçleri veya saldırılar erken evrede fark edilip önlenebilirler. Bunun için gerekli her şeyin loglandığından ve monitoring işlemlerinin yapıldığından emin olunmalıdır.

Loglama ve izleme süreçlerindeki eksiklikler OWASP TOP listesinde 9 numarada yer alan Security Logging and Monitoring Failures veya Güvenlik İzleme ve Loglama Hataları'na sebep olur. Loglanması gereken verilerin düzenli olarak loglanmaması şüpheli davranışların tespitini zorlaştırır. Bilgi toplama aşamalarında olduğu gibi brute-force veya dos saldırıları gibi zaman alan ve sistemde çok sayıda log bırakan saldırılar bu adımların atlanması halinde kolaylıkla başarıya ulaşabilir.

Zafiyetin Nedenleri:

1. **Eksik veya Yetersiz Günlükleme:**
    - Güvenlik olaylarını kaydeden günlükleme sistemlerinin olmaması veya yetersiz olması. Örneğin, sadece kritik olayları değil, tüm güvenlik olaylarını kaydeden sistemlerin eksikliği.
2. **Günlük Verilerinin Saklanmaması:**
    - Günlüklerin yeterli süre boyunca saklanmaması veya arşivlenmemesi. Bu, geçmiş olayların analiz edilmesini zorlaştırır.
3. **Yetersiz İzleme ve Uyarı Sistemleri:**
    - Güvenlik olaylarını izleyen ve bu olaylara karşı uyarı veren sistemlerin yetersiz olması. Örneğin, izleme sistemleri yalnızca belirli olaylara tepki verir ve diğer önemli olayları göz ardı eder.
4. **Yanlış Yapılandırılmış Günlükleme Sistemleri:**
    - Günlükleme sistemlerinin yanlış yapılandırılması, kritik bilgilerin kaydedilmemesi veya yanlış şekilde saklanması.
5. **İzleme Araçlarının Güncellenmemesi:**
    - İzleme ve günlükleme araçlarının güncel olmaması veya güvenlik tehditlerini tespit edememesi.

Nasıl Önlenir ?

1. **Kapsamlı Günlükleme:**
   - Tüm güvenlik olaylarının, kritik erişim ve veri değişikliklerinin, hata durumlarının ve yetkisiz giriş girişimlerinin kayıt altına alındığı kapsamlı bir günlükleme stratejisi oluşturun.
2. **Düzenli İzleme ve Uyarı Sistemleri:**
   - Güvenlik olaylarını sürekli izleyen ve anormal durumları veya potansiyel tehditleri tespit eden uyarı sistemleri kullanın. Bu sistemler, olayları gerçek zamanlı olarak izlemeli ve gerekli bildirimleri sağlamalıdır.
3. **Günlük Verilerinin Saklanması:**
   - Günlük verilerini yeterli süre boyunca saklayarak, geçmiş olayları analiz edebilmek için uzun vadeli arşivleme çözümleri kullanın. Yasal ve düzenleyici gerekliliklere uygun olarak veri saklama sürelerini belirleyin.
4. **Günlükleme Araçlarının Güncel Olması:**
   - Günlükleme ve izleme araçlarınızı düzenli olarak güncelleyerek, yeni güvenlik tehditlerine karşı etkili bir şekilde korunmasını sağlayın.
5. **Otomatik Olay Yanıtı:**
   - Güvenlik olaylarına otomatik yanıt veren sistemler kurarak, tehditlere hızlı bir şekilde müdahale edin. Örneğin, belirli bir türdeki anormal faaliyet tespit edildiğinde otomatik olarak uyarı veya engelleme mekanizmaları çalıştırılabilir.
6. **Düzenli Denetimler ve Testler:**
   - Günlükleme ve izleme sistemlerinizi düzenli olarak denetleyin ve test edin. Sistemlerin etkinliğini ve doğruluğunu kontrol edin ve eksiklikleri giderin.

Server-Side Request Forgery (SSRF) Nedir?

SSRF, bir web uygulamasının veya API'nin, dışardan gelen verileri (örneğin, bir URL) işleyip bu URL'ye istek göndermesine olanak tanıdığı durumlarda ortaya çıkar. Eğer uygulama bu istekleri yeterince doğrulamadan gönderirse, saldırgan bu mekanizmayı kullanarak sunucunun erişim iznine sahip olduğu sistemlere istekler gönderebilir.

Örnek Senaryo :

Uygulama Özelliği:
Bir e-ticaret web uygulaması, kullanıcıların dış web hizmetlerinden veri çekmesini sağlayan bir özelliğe sahiptir. Bu özellik, kullanıcıların "Harici URL'yi Getir" adında bir form

aracılığıyla bir URL girip bu URL'ye HTTP istekleri göndermesine olanak tanır. Kullanıcılar, bu özelliği kullanarak belirli bir URL'den veri alabilirler.

İlgili Özellik:

- **Form Alanı:** Kullanıcılar, "Harici URL" adı verilen bir form alanına URL girerler.
- **İşlem:** Kullanıcı "Getir" butonuna tıkladığında, uygulama sunucu tarafında belirtilen URL'ye bir HTTP GET isteği gönderir.
- **Sonuç:** Sunucu, gelen yanıtı alır ve kullanıcıya gösterir.

Güvenlik Açığı:

Uygulama, kullanıcıların iç IP adreslerine veya yerel ağdaki hizmetlere istek göndermesine izin vermektedir. Uygulamanın içeriği doğrulama veya filtreleme özelliği yoktur.

Saldırı Senaryosu:

1. **Saldırganın Aksiyonları:**
    - Saldırgan, e-ticaret sitesinde yer alan "Harici URL'yi Getir" formunu kullanarak URL alanına özel bir IP adresi girer. Örneğin:
        - http://localhost:8000/admin (sunucunun yerel yönetim paneli)
        - http://192.168.1.10:3306 (yerel ağdaki bir veritabanı sunucusu)
        - http://127.0.0.1:9000/secret (sunucunun yerel makinesindeki gizli bir hizmet)
2. **İstek Gönderimi:**
    - Saldırgan, "Getir" butonuna tıklayarak sunucunun, belirttiği iç IP adresine istek göndermesini sağlar.
3. **Sonuç:**
    - Sunucu, iç IP adresine olan isteği gerçekleştirir ve yanıtı alır. Eğer URL, bir yönetim paneli veya veritabanı sunucusuna işaret ediyorsa, sunucu bu servisten gelen yanıtı içerir. Saldırgan bu yanıtı görür ve iç sistemlere dair hassas bilgilere erişim sağlayabilir.
    - Örneğin, eğer URL bir yönetim panelini işaret ediyorsa, bu panelin kullanıcı arayüzüne veya hata mesajlarına erişebilir.

Potansiyel Sonuçlar:

- **İç Bilgilere Erişim:**
    - Saldırgan, iç ağdaki yönetim panelleri, veri tabanı sunucuları veya diğer hassas sistemlere erişim sağlayabilir.
- **Veri Çalınması:**
    - İç sistemlerden veri çalabilir. Örneğin, bir veritabanı sunucusundan veri çekebilir.

- **Kötüye Kullanım:**
    - İç sistemlerdeki zafiyetleri kullanarak daha fazla kötüye kullanım yapabilir.

Önleme ve Koruma Adımları:

1. **URL Doğrulama ve Temizleme:**
    - Kullanıcıların girilen URL'leri doğrulama ve yalnızca güvenilir ve izin verilen dış URL'lere yönlendirme yapmalarını sağlayın. İç IP adreslerine veya özel yerel adreslere olan istekleri engelleyin.
2. **Beyaz Liste (Whitelist) Kullanımı:**
    - Sadece belirli, güvenli harici URL'lere veya IP adreslerine istek yapılmasına izin verin. İç IP adresleri veya yerel ağ adreslerini engelleyen bir beyaz liste oluşturun.
3. **İç Erişim Kısıtlamaları:**
    - Sunucunun iç ağda bulunan sistemlere erişimini kısıtlayın. İç ağ kaynaklarına doğrudan erişim yerine proxy veya güvenli geçişler kullanın.
4. **Güvenlik Testleri ve Denetimler:**
    - SSRF açıklarını tespit etmek için düzenli güvenlik testleri ve denetimler gerçekleştirin. Uygulamanın güvenlik açıklarını belirleyin ve kapatın.
5. **Günlükleme ve İzleme:**
    - Uygulamanızdaki istekleri ve yanıtları günlükleyin ve izleyin. Şüpheli etkinlikleri tespit etmek için bu günlükleri analiz edin.