RAPOR BAŞLIĞI: OWASP Top 10 Araştırması

HAZIRLAYAN ADI SOYADI: Huriye Gökçen AÇIKGÖZ

AÇIKLAMASI: OWASP Top 10 hakkında genel bir araştırma ve her bir kategori hakkında kısa açıklamalar içeren bir rapor.

---

1) **Broken Access Control**:

a)Nedir?

"Broken Access Control", OWASP (Open Web Application Security Project) tarafından belirlenen en kritik web uygulama güvenlik açıkları arasında yer alır ve Türkçe'de "Bozuk Erişim Kontrolü" veya "Bozuk Yetkilendirme" olarak ifade edilebilir.

b)Neyden kaynaklanır?

"Broken Access Control" zafiyetleri, genellikle yanlış yapılandırılmış yetkilendirme politikaları, eksik güvenlik denetimleri ve yetersiz rol bazlı erişim kontrolü gibi nedenlerden kaynaklanır. Uygulamalarda erişim kontrollerinin doğru bir şekilde uygulanmaması, kullanıcıların yetkileri dışındaki alanlara veya verilere erişmesine yol açabilir. Ayrıca, doğrudan nesne referanslarının güvensiz bir şekilde işlenmesi, yetersiz girdi doğrulaması ve eski yazılımların kullanılması da bu zafiyetlerin oluşmasına sebep olabilir. Bu tür güvenlik açıkları, sistemin saldırılara karşı savunmasız kalmasına neden olur ve bu yüzden erişim kontrolü mekanizmalarının dikkatlice tasarlanması ve düzenli olarak test edilmesi önemlidir.

c) Türleri ve Kısa Açıklamaları:

Bu kategorinin altına, güvenlik açısından önemli olan şu durumlar girer:

1. IDOR (Insecure Direct Object References) - Güvensiz Doğrudan Nesne Referansları

- Kullanıcılar, URL veya parametreler aracılığıyla doğrudan belirli bir nesneye (örneğin, kullanıcı ID'si) erişim sağlayabilirler. Bu, yetkilendirme kontrolleri doğru uygulanmadığında, bir kullanıcının başka bir kullanıcının verilerine erişmesine olanak tanır.

2. Yetersiz Yetki Kontrolleri

- Kullanıcıların, rol veya yetkileri göz önüne alınmadan bazı işlemleri gerçekleştirebilmesine neden olan eksikliklerdir. Örneğin, bir normal kullanıcının admin paneline erişim sağlayabilmesi.

3. Zayıf Rol Bazlı Erişim Kontrolü (RBAC)

- Rollerin ve kullanıcıların yetkilerinin doğru bir şekilde tanımlanmadığı durumlar. Bu, bir kullanıcının olması gerekenden daha fazla yetkiye sahip olmasına neden olabilir.

4. Yatay Yetki Yükseltme (Horizontal Privilege Escalation)

- Bir kullanıcının, kendi düzeyinde ancak başkasına ait verilere veya işlemlere erişim sağlamasıdır. Örneğin, bir kullanıcı başka bir kullanıcının hesabına erişebilir.

5. Dikey Yetki Yükseltme (Vertical Privilege Escalation)

- Normal bir kullanıcının, yönetici gibi daha üst seviyede yetkilere sahip olması durumudur. Örneğin, bir kullanıcının admin paneline erişim sağlayabilmesi.

6. Erişim Kontrol Listelerinde (ACL) Yanlış Yapılandırmalar

- Erişim Kontrol Listelerinin (ACL) yanlış yapılandırılması, kullanıcıların yetkisiz alanlara erişim sağlamasına yol açabilir.

7. Zayıf URL Yetkilendirmesi

- Sadece URL'deki bir değişiklikle yetkisiz işlemlerin gerçekleştirilebilmesi. Örneğin, /admin/delete?id=1 gibi bir URL'ye doğrudan erişim sağlayarak veri silme işlemi yapılabilir.

8. Çok Faktörlü Kimlik Doğrulamanın Atlatılması

- Çok faktörlü kimlik doğrulama (MFA) gibi güvenlik önlemlerinin atlanması veya baypas edilmesiyle erişim sağlanmasıdır.

9. Kapsam Dışı Erişim Kontrolleri

- Uygulamada belirli bir sayfa veya işlev için erişim kontrolünün tamamen atlanması. Bu durum, gizli veya özel verilerin açığa çıkmasına neden olabilir.

10. Zayıf API Erişim Kontrolü: API'ler üzerinden yapılan işlemlerde, yetkilendirme eksiklikleri nedeniyle yetkisiz erişim sağlanabilmesi. API uç noktalarının (endpoints) doğru bir şekilde korunmaması bu soruna yol açabilir.

d)Nasıl önlenir?

"Broken Access Control" zafiyetlerini önlemenin bazı temel yolları:

1. Rol Bazlı Erişim Kontrolü (RBAC) Uygulama

- Kullanıcıların yetkilerini ve rollerini net bir şekilde tanımlayın ve sadece ihtiyaç duydukları minimum yetkileri verin. Gereksiz yetkilendirmelerden kaçının.

2. Erişim Kontrol Denetimlerini Zorunlu Hale Getirme

- Tüm önemli işlevler ve kaynaklar için erişim kontrolü denetimlerini uygulayın. Yetkilendirme kontrollerinin eksiksiz ve doğru bir şekilde uygulandığından emin olun.

3. Kullanıcı Girdilerini Doğrulama

- Kullanıcı girdilerini dikkatlice doğrulayarak, kimlik doğrulama ve yetkilendirme süreçlerini güvence altına alın. Parametre manipülasyonuna karşı koruma sağlamak için input validation (girdi doğrulama) kullanın.

4. En Az Yetki Prensibi (Principle of Least Privilege)

- Kullanıcılara yalnızca işlerini yapmaları için gerekli olan en düşük yetkiyi verin. Bu, yanlış yapılandırmalardan kaynaklanan yetki yükseltme riskini azaltır.

5. Doğrudan Nesne Referanslarını (IDOR) Güvenli Hale Getirme

- Doğrudan nesne referansları yerine rastgele oluşturulmuş ve tahmin edilemez kimlikler kullanın. Nesnelere erişim sağlanmadan önce yetkilendirme kontrollerinin yapıldığından emin olun.

6. Düzenli Güvenlik Testleri ve Denetimleri Yapma

- Uygulamanızın düzenli olarak güvenlik testlerinden geçirilmesini sağlayın. Penetrasyon testleri ve kod incelemeleri, erişim kontrolü zafiyetlerini tespit etmede etkili olabilir.

7. Güvenli Varsayılan Yapılandırmalar Kullanma

- Erişim kontrol listeleri (ACL) ve diğer güvenlik ayarlarını güvenli varsayılan ayarlara göre yapılandırın. Yanlış yapılandırmaların önüne geçmek için yapılandırma yönetimi araçları kullanın.

8. Güncel Yazılım ve Kütüphaneler Kullanma

- Kullandığınız yazılım ve kütüphanelerin güncel olduğundan emin olun. Bilinen güvenlik açıklarını gidermek için düzenli olarak güncellemeler uygulayın.

9. Çok Faktörlü Kimlik Doğrulama (MFA)

- Özellikle kritik işlemler için çok faktörlü kimlik doğrulama (MFA) kullanarak, yetkisiz erişimi zorlaştırın.

Bu önlemler, Broken Access Control zafiyetlerini önemli ölçüde azaltmaya yardımcı olur ve uygulamalarımızın güvenliğini artırır.

2) **Cryptographic Failures** :

a) Nedir?

Kriptografi, dijital dünyada veri güvenliğini sağlamak için temel bir unsurdur. Kriptografik yöntemler, verileri şifreleyerek yetkisiz erişimlerden korur, veri bütünlüğünü sağlar ve güvenli iletişim kanalları oluşturur. Ancak, kriptografik yöntemlerin yanlış kullanımı veya yapılandırılması, güvenlik açıklarına neden olabilir. Bu tür hatalar "kriptografik hatalar" olarak adlandırılır.

b)Neyden Kaynaklanır?

Kriptografik Hataların Kaynakları:

1. **Zayıf veya Eskimiş Şifreleme Algoritmalarının Kullanılması:**

- **Eski Algoritmalar:** Zamanla, belirli kriptografik algoritmaların güvenliği zayıflar. Örneğin, MD5 ve SHA-1 gibi algoritmalar, günümüz teknolojisiyle kolayca kırılabilir. Bu nedenle, eski ve güvenli olmayan algoritmaların kullanımı, verilerin ele geçirilmesine yol açabilir.
- **Zayıf Algoritmalar:** Yetersiz uzunlukta anahtarlar kullanan veya kötü tasarlanmış algoritmalar, brute-force (kaba kuvvet) saldırıları gibi yöntemlerle kolayca kırılabilir.

2. **Yanlış Anahtar Yönetimi:**

- **Anahtarların Güvensiz Saklanması:** Şifreleme anahtarlarının güvenli olmayan bir ortamda saklanması, bu anahtarlara yetkisiz erişimi kolaylaştırır. Anahtarların doğru bir şekilde saklanmaması, sistemin güvenliğini ciddi şekilde tehlikeye atar.
- **Anahtarların Yenilenmemesi:** Anahtarların düzenli olarak değiştirilmemesi, saldırganların bu anahtarlara erişme olasılığını artırır. Ayrıca, aynı anahtarın birden fazla işlemde kullanılması da güvenliği zayıflatır.

3. **Zayıf Parola Politikaları:**

- Güçlü ve karmaşık parolalar kullanılmadığında, sistemler brute-force saldırılarına karşı savunmasız hale gelir. Parolaların zayıf olması, yetkisiz erişimlere kapı aralar.

4. **Hatalı Sertifika Yönetimi:**

- **Geçersiz veya Süresi Dolmuş Sertifikalar:** Güvenlik sertifikalarının süresi dolduğunda veya geçersiz hale geldiğinde, sistemler yetkisiz erişimlere ve man-in-the-middle (ortadaki adam) saldırılarına karşı savunmasız hale gelir.
- **Yanlış Sertifika Kullanımı:** Yanlış veya sahte sertifikaların kullanılması, kullanıcıların kimlik doğrulamasını geçersiz kılabilir ve saldırganların sisteme erişimini kolaylaştırabilir.

5. **Eksik Şifreleme Uygulamaları:**

- **Tüm Verilerin Şifrelenmemesi:** Bazı durumlarda, hassas verilerin sadece bir kısmının şifrelenmesi veya verilerin şifreleme sırasında yeterince korunmaması, saldırganların bu verilere erişmesine yol açabilir.
- **Güvenli Olmayan Şifreleme Yöntemlerinin Kullanılması:** Yanlış uygulanan şifreleme yöntemleri, verilerin şifre çözme işlemini kolaylaştırabilir.

c)Nasıl önlenir?

1) **Güncel ve Güçlü Şifreleme Algoritmaları Kullanmak**
   - **Güvenli Algoritmalar:** MD5 ve SHA-1 gibi eski ve zayıf şifreleme algoritmaları yerine, AES-256 ve SHA-256 gibi modern, güçlü algoritmalar kullanılmalıdır. Bu algoritmalar, güncel tehditlere karşı daha güvenlidir.
   - **Algoritma Güncellemeleri:** Kullanılan şifreleme algoritmaları düzenli olarak gözden geçirilmeli ve gerekli durumlarda daha güvenli seçeneklerle değiştirilmelidir.
2) **Güvenli Anahtar Yönetimi Uygulamak**
   - **Anahtar Saklama:** Şifreleme anahtarları, güvenli ortamlar veya donanım güvenlik modülleri (HSM) gibi güvenli çözümlerle saklanmalıdır. Anahtarların erişimi yalnızca yetkili kişilerle sınırlandırılmalıdır.
   - **Anahtar Döngüsü:** Şifreleme anahtarları düzenli olarak yenilenmeli ve aynı anahtarın tekrar kullanılması önlenmelidir. Anahtarların belirli aralıklarla değiştirilmesi, güvenlik açıklarının minimize edilmesine yardımcı olur.
3) **Güçlü Parola Politikaları Geliştirmek**
   - **Karmaşık Parolalar:** Kullanıcılar, güçlü, karmaşık ve benzersiz parolalar oluşturmaya teşvik edilmelidir. Parolaların minimum uzunlukta olması, büyük/küçük harfler, sayılar ve özel karakterler içermesi sağlanmalıdır.
   - **İki Faktörlü Kimlik Doğrulama (2FA):** Parola güvenliğini artırmak için iki faktörlü kimlik doğrulama yöntemleri uygulanmalıdır.
4) **Güvenilir Sertifika Yönetimi Sağlamak**
   - **Sertifika Geçerliliği:** Güvenlik sertifikaları düzenli olarak kontrol edilmeli ve süresi dolmuş sertifikalar hemen yenilenmelidir. Sahte sertifikaların kullanımını önlemek için sertifikaların doğruluğu sıkı bir şekilde denetlenmelidir.
   - **Doğru Sertifika Kullanımı:** Sadece güvenilir sertifika otoriteleri (CA) tarafından verilen sertifikalar kullanılmalı ve sertifikaların doğru yapılandırıldığından emin olunmalıdır.
5) **Verilerin Güvenli Şifrelenmesini Sağlamak**

- **Tam Şifreleme:** Tüm hassas veriler, veri taşıma sırasında (in transit) ve depolama halinde (at rest) şifrelenmelidir. Bu, verilerin yetkisiz erişime karşı korunmasını sağlar.
- **Uygulama Düzeyinde Şifreleme:** Şifreleme ve şifre çözme işlemleri yalnızca gerekli olduğu zaman ve güvenli koşullarda gerçekleştirilmelidir.

6) **Düzenli Güvenlik Denetimleri Yapmak**
   - **Güvenlik Denetimleri:** Kriptografik uygulamalar ve politikalar düzenli olarak denetlenmeli ve olası zafiyetler tespit edilmelidir. Bu denetimler, güvenlik açıklarının erken tespit edilip giderilmesine olanak tanır.
   - **Penetrasyon Testleri:** Sistemler, olası saldırılara karşı düzenli olarak test edilmeli (penetrasyon testleri) ve kriptografik hatalar tespit edildiğinde hemen düzeltilmelidir.
7) **Eğitim ve Farkındalığı Artırmak**
   - **Güvenlik Eğitimi:** Yazılım geliştiriciler, sistem yöneticileri ve diğer ilgili personellere düzenli olarak kriptografi ve güvenlik konularında eğitim verilmelidir. Eğitimler, kriptografik hataların önlenmesine yönelik farkındalığı artırır ve insan hatasını minimize eder.

Sonuç olarak kriptografik hataların önlenmesi, yalnızca teknik önlemlerle sınırlı kalmamalı, aynı zamanda organizasyon genelinde bir güvenlik kültürü oluşturulmalıdır. Güvenli şifreleme yöntemlerinin doğru uygulanması, anahtar yönetiminin titizlikle yapılması, düzenli denetimler ve sürekli eğitimlerle desteklenmelidir. Bu şekilde kriptografik haalrı önleyebiliriz.

3) **Injection**:

a)Nedir?

Injection (enjeksiyon) saldırıları, bir saldırganın, hedef sistemin veri işleme mekanizmasına, genellikle kullanıcı girdisi aracılığıyla zararlı komutlar enjekte ederek sistemin normal işleyişini bozmaya çalıştığı güvenlik açıklarıdır. Bu saldırılar, veritabanı, işletim sistemi komutları, LDAP sorguları, XML işleme, HTTP başlıkları gibi çeşitli alanlarda gerçekleşebilir. Injection saldırıları, güvenlik açıklarının en kritik ve yaygın türlerinden biridir.

b)Neyden Kaynaklanır?

Injection saldırıları, genellikle kullanıcı girdisinin yeterince doğrulanmaması ve filtrelenmemesi sonucu ortaya çıkar. Bir sistem, kullanıcıdan aldığı girdiyi doğrudan bir komut veya sorgu içinde kullanıyorsa, saldırganlar bu girdiye zararlı içerikler ekleyerek sistemi manipüle edebilir. Bu güvenlik açığı, zayıf kodlama uygulamalarından, güvenlik farkındalığının eksikliğinden veya doğru güvenlik kontrollerinin uygulanmamasından kaynaklanır.

c) Türleri ve Kısa Açıklamaları:

I. **SQL Injection (SQL Enjeksiyonu):**

Tanım: SQL Injection, bir saldırganın bir SQL sorgusuna zararlı SQL komutları enjekte ederek veritabanını manipüle etmesine olanak tanıyan bir güvenlik açığıdır. Bu tür saldırılar, saldırganların hassas verileri görüntülemesine, değiştirmesine veya veritabanı sunucusunu ele geçirmesine neden olabilir

Örnek Kod:

```
$username = $_POST['username'];
$query = "SELECT * FROM users WHERE username =
'$username'";$result = mysqli_query($conn, $query);
```

Saldırı Senaryosu:

Kullanıcı adı yerine şu değer girildiğinde: ' OR '1'='1', sorgu şu şekilde çalışır.

SELECT * FROM users WHERE username = '' OR '1'='1';

Bu sorgu, veritabanındaki tüm kullanıcıları döndürür, çünkü '1'='1' her zaman doğru döner.

II. **Command Injection (Komut Enjeksiyonu):**

Tanım: Komut enjeksiyonu, bir saldırganın, sistem komutlarını çalıştıran bir uygulamanın bu komutlara zararlı içerik enjekte etmesine olanak tanıyan bir güvenlik açığıdır. Bu saldırı, saldırganların sunucuda keyfi komutlar çalıştırmasını sağlar.

Örnek Kod:

```
$filename = $_POST['filename'];
system("cat " . $filename);
```

Saldırı Senaryosu: Dosya adı yerine şu değer girildiğinde: ; rm -rf /, sistem komutları şu şekilde çalışır:

cat ; rm -rf /

Bu komut, sunucudaki tüm dosyaları siler.

III. **LDAP Injection (LDAP Enjeksiyonu):**

Tanım: LDAP Injection, bir saldırganın LDAP sorgularına zararlı kod enjekte etmesine olanak tanır. Bu, LDAP dizin sunucularına yönelik saldırılarda kullanılır ve hassas verilerin açığa çıkmasına neden olabilir.

Örnek Kod:

```
$username = $_POST['username'];
$ldap_query = "(uid=$username)";
$result = ldap_search($ldap_conn, "dc=example,dc=com", $ldap_query);
```

Saldırı Senaryosu: Kullanıcı adı yerine şu değer girildiğinde: *)(|(uid=*, LDAP sorgusu şu şekilde çalışır:

(uid=*)(|(uid=*)

Bu sorgu, dizindeki tüm kullanıcıları döndürür.

IV. **XML Injection (XML Enjeksiyonu):**

Tanım: XML Injection, bir saldırganın XML verilerine veya XML tabanlı uygulamalara zararlı içerik enjekte etmesine olanak tanır. Bu tür saldırılar, XML verilerinin işlenmesi sırasında gerçekleşir ve XML tabanlı sistemlerin manipüle edilmesine neden olabilir.

Örnek Kod:

```
<user>
<name>{$_POST['name']}</name>
</user>
```

Saldırı Senaryosu: Kullanıcı adı yerine şu değer girildiğinde: <admin><role>admin</role></admin>, XML şu şekilde çalışır:

```
<user>
<name><admin><role>admin</role></admin></name>
</user>
```

Bu, XML işleyiciye sahte veriler sağlar.

d)Nasıl Önlenir?

i. **Parametreli Sorgular (Prepared Statements):**

Veritabanı sorgularında kullanıcı girdilerini doğrudan kullanmaktan kaçının. Bunun yerine, parametreli sorgular kullanarak kullanıcı girdilerini güvenli bir şekilde işleyin.

Örnek:

```
$stmt = $conn->prepare("SELECT * FROM users WHERE username = ?");
$stmt->bind_param("s", $username);
$stmt->execute();
```

ii. **Girdi Doğrulama:**

Tüm kullanıcı girdilerini, özellikle de tehlikeli karakterleri filtreleyin ve doğrulayın. Sadece beklenen veri türlerini kabul edin.

Örnek:

```
$username = filter_input(INPUT_POST, 'username', FILTER_SANITIZE_STRING);
```

iii. **Çıkış Kaçırma (Output Escaping):**

Kullanıcı girdilerini herhangi bir sistem komutuna veya veri işlemeye dahil etmeden önce, bu girdileri kaçırarak güvenli hale getirin.

Örnek:

```
$safe_string = htmlspecialchars($user_input, ENT_QUOTES, 'UTF-8');
```

iv. **Güvenli Kodlama Uygulamaları:**

Kod yazarken, injection saldırılarına karşı duyarlı olabilecek kod bölgelerini tanımlayın ve bu bölgeleri güvenli hale getirin.

Eğitim: Geliştirici ekiplerin güvenli kodlama konusunda eğitilmesi ve farkındalıklarının artırılması.

v. **Güvenlik Duvarları ve WAF (Web Application Firewall):**

Web uygulamaları için güvenlik duvarları kullanarak, kötü niyetli saldırıların tespit edilmesini ve engellenmesini sağlayın.

4) **Insecure Design**:

a)Nedir?

Insecure Design" (Güvensiz Tasarım), bir sistemin, uygulamanın veya yazılımın tasarım aşamasında güvenliğin yeterince dikkate alınmaması sonucu ortaya çıkan güvenlik açıklarıdır. Bu tür güvenlik açıkları, genellikle bir saldırı gerçekleştirilmeden önce tasarım sürecinde meydana gelir ve bu durum, güvenliğin birincil öncelik olmadığı veya yetersiz güvenlik önlemlerinin alındığı anlamına gelir.

b)Neden Olur?

Insecure Design, çeşitli nedenlerden dolayı ortaya çıkar ve genellikle sistem veya yazılımın tasarım aşamasında güvenliğin yeterince dikkate alınmamasından kaynaklanır. Bu durumun en temel nedenlerinden biri, tasarım süreçlerinde güvenlik bilincinin eksik olmasıdır. Geliştirici ekipler, genellikle güvenlik konularına yeterince odaklanmaz ve bu da tasarımda ciddi güvenlik açıklarının oluşmasına yol açar. Bir diğer neden ise yetersiz tehdit modelleme ve risk değerlendirmesidir; bu süreçlerde, potansiyel saldırı vektörleri yeterince analiz

edilmediğinde, sistemin zayıf noktaları gözden kaçabilir. Hızlı pazara çıkma baskısı da güvensiz tasarımın oluşmasına katkıda bulunur, çünkü zaman kısıtlamaları nedeniyle güvenlik önlemleri göz ardı edilebilir veya yeterince uygulanmayabilir. Ayrıca, güvenlik standartlarına uyulmaması veya en iyi uygulamaların takip edilmemesi, tasarım aşamasında güvenlik açıklarının ortaya çıkmasının bir başka önemli sebebidir. Bu faktörlerin birleşimi, tasarım sürecinde güvenliğin ikinci plana atılmasına ve güvenlik zafiyetlerinin sistematik olarak meydana gelmesine neden olur.

c) Türleri ve Kısaca Açıklamaları:

Zayıf Kimlik Doğrulama ve Yetkilendirme:

Kullanıcıların kimlik doğrulama ve yetkilendirme süreçlerinin yetersiz veya eksik tasarlandığı durumlardır. Bu, örneğin zayıf parola gereksinimleri veya yetkilendirme kontrollerinin etkisiz olduğu durumları içerir.

Eksik Girdi Doğrulama:

Kullanıcı girdilerinin yeterince doğrulanmadığı durumlarda ortaya çıkar. Bu eksiklik, saldırganların zararlı girdilerle sistemi manipüle etmesine imkan tanır.

Yanlış Konfigürasyonlar:

Sistemlerin veya uygulamaların güvenli bir şekilde yapılandırılmaması sonucu ortaya çıkan güvenlik açıklarını ifade eder. Bu, örneğin varsayılan ayarların değiştirilmemesi gibi durumları kapsar.

Güvensiz Veri Depolama:

Hassas verilerin güvensiz bir şekilde depolandığı, şifrelenmeden saklandığı veya kolayca erişilebilir olduğu durumlardır. Bu tür güvenlik açıkları, verilerin kötü niyetli kişiler tarafından ele geçirilmesine neden olabilir.

Zayıf Oturum Yönetimi:

Kullanıcı oturumlarının güvensiz bir şekilde yönetildiği durumlardır. Bu, oturum kimliklerinin (session IDs) güvensiz bir şekilde saklanması veya iletilmesi gibi sorunları içerir ve saldırganların oturumları ele geçirmesine yol açabilir.

d) Nasıl Önleriz?

- Güvenlik Bilincinin Artırılması:
  - Tasarım sürecine dahil olan tüm ekiplerin güvenlik konusunda eğitilmesi ve farkındalıklarının artırılması gereklidir. Güvenliğin tasarımın her aşamasında göz önünde bulundurulması sağlanmalıdır.

- Tehdit Modelleme ve Risk Değerlendirmesi:
  - Tasarım sürecinin erken aşamalarında tehdit modelleme ve risk değerlendirmesi yapılarak, potansiyel güvenlik açıkları önceden tespit edilmeli ve tasarım buna göre uyarlanmalıdır.
- Güvenlik Standartlarına ve En İyi Uygulamalara Uyum:
  - Güvenlik standartları ve en iyi uygulamalar, tasarım sürecinde rehber olarak kullanılmalıdır. Bu, yazılım geliştirme süreçlerinde güvenliğin sistematik bir şekilde ele alınmasını sağlar.
- Düzenli Güvenlik Denetimleri ve Testleri:
  - Tasarım sürecinin her aşamasında düzenli güvenlik denetimleri yapılmalı ve olası güvenlik açıkları test edilmelidir. Bu, tasarımın güvenli bir şekilde ilerlediğini doğrulamak için önemlidir.
- Güvenli Kodlama Prensiplerinin Benimsenmesi:
  - Güvenli kodlama prensipleri, tasarım aşamasında dikkate alınmalı ve kodlama sırasında bu prensiplerin uygulanması sağlanmalıdır. Bu, yazılımın tasarımından itibaren güvenli olmasını garanti eder.

5) **Security Misconfiguration** :

a) Zafiyet Nedir?

Security Misconfiguration (Güvenlik Yanlış Yapılandırması), sistemlerin, uygulamaların veya ağların güvenlik ayarlarının yanlış yapılandırılması sonucunda ortaya çıkan güvenlik zafiyetlerini ifade eder. Bu tür zafiyetler, bir sistemin beklenen güvenlik önlemlerini karşılayacak şekilde yapılandırılmaması nedeniyle oluşur ve saldırganların sistemlere izinsiz erişim sağlamasına, veri hırsızlığına veya hizmet kesintilerine yol açabilir. Güvenlik yanlış yapılandırmaları, genellikle web sunucuları, veritabanları, ağ cihazları, işletim sistemleri ve uygulamalarda yaygın olarak görülür.

b) Neden Kaynaklanır?

Security Misconfiguration, birkaç temel nedenden dolayı ortaya çıkabilir:

1. **Varsayılan Ayarların Değiştirilmemesi:**
   - Birçok yazılım veya sistem varsayılan olarak zayıf güvenlik ayarlarıyla gelir. Bu ayarların değiştirilmemesi, saldırganlar için kolay hedefler yaratır.
2. **Güncellemelerin İhmal Edilmesi:**
   - Güvenlik yamalarının veya yazılım güncellemelerinin zamanında uygulanmaması, bilinen zafiyetlerin sömürülmesine yol açabilir.
3. **Aşırı Yetki Verilmesi:**
   - Sistem bileşenlerine veya kullanıcılara gereğinden fazla yetki verilmesi, kötü niyetli işlemlerin gerçekleştirilmesine olanak tanır.
4. **Güvenlik Kontrollerinin Yanlış Yapılandırılması:**

- Güvenlik duvarları, kimlik doğrulama sistemleri ve diğer güvenlik kontrollerinin yanlış yapılandırılması, güvenlik açıklarına neden olabilir.

5. **Gizli Bilgilerin Yanlış Yönetimi:**
   - Gizli anahtarların, şifrelerin veya diğer hassas bilgilerin yanlış veya güvensiz bir şekilde saklanması, sistem güvenliğini tehlikeye atar.

c) Örnek:

Bir web sunucusunda varsayılan parolaların değiştirilmemesi sonucu oluşabilecek bir güvenlik açığına dair basit bir PHP kodu örneği:

Varsayılan admin hesabı ve parolası

```
$username = "admin";
$password = "admin"; -Varsayılan parola
```

Giriş kontrolü

```
if ($_POST['username'] == $username && $_POST['password'] == $password) {
    echo "Giriş Başarılı!";
} else {
    echo "Hatalı Kullanıcı Adı veya Parola!";
}
```

Bu örnekte, varsayılan kullanıcı adı ve parolanın değiştirilmemesi, saldırganların kolayca erişim sağlamasına yol açabilir.

d) Nasıl Önlenir?

1. **Güvenli Varsayılanlar Belirleme:**
   - Varsayılan ayarları güvenlik odaklı olarak değiştirin. Varsayılan parolaları ve kullanıcı adlarını mutlaka değiştirin.
2. **Düzenli Güncellemeler ve Yamalar:**
   - Sistemlerinizi ve yazılımlarınızı düzenli olarak güncelleyin ve güvenlik yamalarını zamanında uygulayın.
3. **İhtiyaç Fazlası Yetkilerin Kaldırılması:**
   - Kullanıcılara ve sistem bileşenlerine sadece ihtiyaç duydukları yetkileri verin. Gereksiz yetkileri kaldırın.
4. **Güvenlik Yapılandırma Yönetimi:**
   - Güvenlik duvarı, kimlik doğrulama sistemleri ve diğer güvenlik kontrollerinin doğru yapılandırıldığından emin olun. Gerektiğinde profesyonel güvenlik denetimleri yapın.
5. **Gizli Bilgilerin Güvenli Yönetimi:**
   - Gizli bilgilerin güvenli bir şekilde saklanmasını sağlayın ve bu bilgilerin yalnızca yetkili kişiler tarafından erişilebilir olduğundan emin olun.
6. **Düzenli Güvenlik Denetimleri:**
   - Sistemlerinizi düzenli olarak güvenlik taramalarından geçirin ve yapılandırmaların doğru olduğundan emin olun. Bu, potansiyel zafiyetlerin önceden tespit edilmesine yardımcı olur.

6)Vulnerable and Outdated Components

a)Zafiyet Nedir?

Vulnerable and Outdated Components (Savunmasız ve Güncel Olmayan Bileşenler), bir sistemde kullanılan yazılım veya donanım bileşenlerinin güncel olmayan sürümleri veya bilinen güvenlik açıklarına sahip sürümleri içermesi durumunda oluşan güvenlik zafiyetlerini ifade eder. Bu bileşenler, güncel güvenlik yamaları uygulanmadığı veya yazılım sürümleri yükseltilmediği için saldırganlar tarafından kolayca sömürülebilecek zayıflıklar barındırır. Özellikle üçüncü parti kütüphaneler, eklentiler veya modüller gibi dış kaynaklı bileşenler, sık güncellenmediğinde ciddi güvenlik riskleri yaratabilir.

b) Neden Kaynaklanır?

Vulnerable and Outdated Components zafiyeti, çeşitli nedenlerden kaynaklanabilir:

1. **Güncellemelerin İhmal Edilmesi:**
    - Sistem yöneticileri veya geliştiriciler tarafından güvenlik yamalarının ve güncellemelerin düzenli olarak uygulanmaması, bileşenlerin savunmasız kalmasına neden olur.
2. **Desteklenmeyen veya Terk Edilmiş Yazılımlar:**
    - Destek süresi dolmuş veya geliştirici tarafından terk edilmiş yazılımlar kullanmak, bu yazılımların güncellemelerini alamadığı için güvenlik açıkları oluşturabilir.
3. **Üçüncü Parti Bileşenlerin Kullanımı:**
    - Üçüncü parti kütüphane veya modüllerin güvenlik güncellemeleri takip edilmediğinde, bu bileşenler potansiyel tehditlere açık hale gelir.
4. **Güvenlik Testlerinin Yetersizliği:**
    - Kullanılan bileşenlerin güvenlik açıklarının tespit edilmesi için yeterli güvenlik testlerinin yapılmaması, mevcut zafiyetlerin fark edilmeden kalmasına neden olabilir.
5. **Uyumluluk Endişeleri:**
    - Bileşenlerin güncellenmesinin diğer sistemlerle veya yazılımlarla uyumsuzluk yaratabileceği endişesi, güncellemelerin ertelenmesine veya ihmal edilmesine yol açabilir.

c) Örnek

Kod Örneği: Savunmasız bir üçüncü parti kütüphanenin kullanımına dair bir JavaScript örneği:

```
Eski bir jQuery sürümü kullanılıyor
<script src="https://code.jquery.com/jquery-1.12.4.min.js"></script>
```

```
Güvenlik açığı barındıran eski bir jQuery sürümü
$(document).ready(function(){
    $("#example").click(function(){
            alert("Güvenlik açığına açık kod!");
     });
});
```

Bu örnekte, eski bir jQuery sürümünün kullanılması, bilinen güvenlik açıklarına sahip olabileceği için risk taşır.

d) Nasıl Önlenir?

1. **Düzenli Güncellemeler ve Yama Yönetimi:**
    - Yazılım ve donanım bileşenlerinin en son sürümlerinin kullanıldığından emin olun. Güvenlik güncellemelerini ve yamalarını düzenli olarak uygulayın.
2. **Desteklenmeyen Yazılımlardan Kaçınma:**
    - Üretici tarafından artık desteklenmeyen yazılım ve donanımları kullanmaktan kaçının. Alternatif, desteklenen çözümler arayın.
3. **Üçüncü Parti Bileşenlerin Takibi:**
    - Üçüncü parti kütüphane ve modüllerin güvenlik güncellemelerini düzenli olarak takip edin. Güvenlik açıkları bulunan bileşenleri kullanmaktan kaçının.
4. **Güvenlik Testlerinin Uygulanması:**
    - Kullandığınız bileşenlerin güvenlik açıklarını tespit etmek için düzenli güvenlik testleri yapın. Bu, mevcut veya yeni zafiyetleri önceden tespit etmeye yardımcı olur.
5. **Uyumluluk Testleri ve Güncelleme Stratejileri:**
    - Güncellemelerin sistemle uyumlu olup olmadığını test edin ve güncellemelerin sistem performansını nasıl etkilediğini değerlendirin. Gerektiğinde uyumluluk sorunlarını çözmek için plan yapın.
6. **Terk Edilmiş Kütüphanelerin Kullanımından Kaçınma:**

- Terk edilmiş veya güncellenmeyen kütüphaneleri kullanmaktan kaçının. Bu tür kütüphanelerin yerine aktif olarak geliştirilen ve desteklenen alternatifler bulun.

7)Identification and Authentication Failures

a) Zafiyet Nedir?

Identification and Authentication Failures (Kimlik Tespit ve Doğrulama Hataları), bir sistemin kullanıcıların kimliğini doğru bir şekilde tanımlayamaması veya kimlik doğrulama süreçlerinde hatalar yapması sonucunda ortaya çıkan güvenlik zafiyetlerini ifade eder. Bu zafiyetler, yetkisiz kullanıcıların sisteme erişmesine, kimlik bilgilerini ele geçirmesine veya kimlik doğrulama sürecini atlamasına neden olabilir. Kimlik tespit ve doğrulama, güvenlik sistemlerinin temel taşlarından biridir; bu süreçlerdeki hatalar, sistemin tamamının güvenliğini tehlikeye atabilir.

b) Neden Kaynaklanır?

Identification and Authentication Failures zafiyetleri birkaç temel nedenden kaynaklanabilir:

1. **Zayıf Parola Politikaları:**
    - Kullanıcıların basit veya yaygın olarak bilinen parolalar kullanmasına izin veren zayıf parola politikaları, kimlik doğrulama süreçlerini tehlikeye atar.
2. **Çok Faktörlü Kimlik Doğrulamanın (MFA) Eksikliği:**
    - MFA kullanmayan sistemler, tek faktörlü kimlik doğrulamanın zayıflıklarından etkilenebilir ve bu durum, kimlik bilgisi hırsızlığı gibi saldırılara açık hale gelir.
3. **Parola Yönetimindeki Hatalar:**
    - Parolaların düz metin olarak saklanması, yeniden kullanılabilir olması veya yeterince karmaşık olmaması, kimlik doğrulama zafiyetlerine yol açabilir.
4. **Oturum Yönetimi Hataları:**
    - Kullanıcı oturumlarının güvenli bir şekilde yönetilememesi, oturum bilgilerini ele geçiren saldırganların yetkisiz erişim sağlamasına neden olabilir.
5. **Eksik veya Hatalı Girdi Doğrulama:**

- Kimlik doğrulama süreçlerinde kullanıcı girdilerinin yeterince doğrulanmaması, saldırganların sahte kimlik bilgileriyle sisteme giriş yapmasına imkan tanır

c) Örnek :

Kod Örneği: Parolaların düz metin olarak saklanmasının bir örneği:

```
Parola düz metin olarak saklanıyor
$password = $_POST['password'];
Veritabanına düz metin olarak kaydediliyor
$query = "INSERT INTO users (username, password) VALUES ('".$username."',
'".$password."')";
mysqli_query($conn, $query);
```

Bu örnekte, parolalar düz metin olarak saklanıyor, bu da bir veri ihlali durumunda kullanıcıların parolalarının kolayca ele geçirilmesine neden olabilir.

d) Nasıl Önlenir?

1. **Güçlü Parola Politikaları Uygulama:**
   - Kullanıcıların karmaşık ve tahmin edilmesi zor parolalar kullanmalarını zorunlu kılın. Minimum uzunluk, büyük-küçük harf, sayı ve özel karakter içermesi gibi gereksinimler belirleyin.
2. **Çok Faktörlü Kimlik Doğrulama (MFA) Kullanımı:**
   - Kimlik doğrulama süreçlerine ikinci bir doğrulama faktörü (örneğin, SMS kodu, biyometrik veri) ekleyerek güvenliği artırın.
3. **Parola Yönetimini Güçlendirme:**
   - Parolaları şifreli veya hashed formatta saklayın. Güçlü hashing algoritmaları (örneğin, bcrypt) kullanarak parolaların güvenliğini sağlayın.
4. **Oturum Yönetimini İyileştirme:**
   - Oturum kimliklerini şifreleyerek saklayın ve iletin. Oturum sürelerini sınırlayın ve oturum kapandıktan sonra geçersiz hale getirin.
5. **Parola Yeniden Kullanımını Engelleme:**
   - Kullanıcıların aynı parolayı birden fazla hesapta kullanmasını önlemek için politika oluşturun. Kullanıcıların parolalarını düzenli aralıklarla güncellemelerini zorunlu kılın.
6. **Girdi Doğrulama ve Sanitizasyon:**
   - Kimlik doğrulama işlemlerinde kullanıcı girdilerini doğrulayarak sahte girişlerin önüne geçin. Girdi sanitizasyonu uygulayarak saldırıları engelleyin.

8) **Software and Data Integrity Failures** :

a) Zafiyet Nedir?

Software and Data Integrity Failures (Yazılım ve Veri Bütünlüğü Hataları), yazılımın veya verilerin yetkisiz kişiler tarafından değiştirilmesi, manipüle edilmesi veya üzerinde oynanması sonucunda ortaya çıkan güvenlik zafiyetlerini ifade eder. Bu tür zafiyetler, sistemlerin beklenmedik şekilde davranmasına, kritik verilerin bozulmasına veya kötü amaçlı yazılımların sisteme sızmasına yol açabilir. Özellikle güvenlik güncellemeleri, kod entegrasyonu, veritabanı işlemleri gibi hassas işlemler sırasında yazılım ve veri bütünlüğünün sağlanamaması büyük riskler oluşturur.

b) Neden Kaynaklanır?

Software and Data Integrity Failures zafiyetleri şu nedenlerden kaynaklanabilir:

1. **Güvenli Olmayan Yazılım Güncellemeleri:**
    - Yazılım güncellemelerinin veya yamalarının güvenli olmayan kanallar üzerinden uygulanması, saldırganların bu süreçlere müdahale ederek zararlı yazılımlar veya kötü niyetli kodlar yerleştirmesine imkan tanır.
2. **Yetersiz Kod İmzalama:**
    - Kod imzalama süreçlerinin uygulanmaması veya yetersiz uygulanması, yetkisiz kişiler tarafından kodun değiştirilmesine ve sistemde çalıştırılmasına yol açabilir.
3. **Zayıf Veritabanı Yönetimi:**
    - Veritabanı işlemlerinde yeterli güvenlik önlemlerinin alınmaması, veri bütünlüğünün bozulmasına veya verilerin izinsiz olarak değiştirilmesine neden olabilir.
4. **Tedarik Zinciri Güvenliği Eksiklikleri:**
    - Üçüncü taraf yazılım bileşenleri veya dış kaynaklı kütüphanelerin güvenlik süreçlerinden geçmemesi, bu bileşenlerin zararlı kodlar içermesine veya kötü amaçlı olarak değiştirilmesine yol açabilir.
5. **Eksik Girdi Doğrulama ve Sanitizasyon:**
    - Kullanıcı girdilerinin yeterince doğrulanmaması, saldırganların zararlı verilerle sistemdeki veri bütünlüğünü bozmasına olanak tanır.

c) Örnek Kod:

- **Kod Örneği:** Güvenli olmayan bir yazılım güncelleme sürecinin örneği:

  Yazılım güncellemesi güvenli olmayan bir kaynaktan indiriliyor

  ```
  wget http://untrusted-source.com/update.sh
  ```

İndirilen güncelleme direkt olarak çalıştırılıyor

```
bash update.sh
```

Bu örnekte, yazılım güncellemesi güvenli olmayan bir kaynaktan indiriliyor ve güvenlik kontrolleri yapılmadan çalıştırılıyor. Bu, zararlı kodların sisteme sızmasına yol açabilir.

d) Nasıl Önlenir?

1. **Güvenli Yazılım Güncellemeleri Sağlama:**
    - Yazılım güncellemelerini sadece güvenilir kaynaklardan indirin ve güncellemelerin bütünlüğünü doğrulamak için dijital imza kontrolleri uygulayın.
2. **Kod İmzalama Kullanımı:**
    - Yazılımın bütünlüğünü korumak ve yetkisiz müdahaleleri önlemek için güçlü kod imzalama süreçleri uygulayın. Tüm kodların imzalı olduğundan emin olun.
3. **Veritabanı Güvenliğini Artırma:**
    - Veritabanı işlemlerinde güçlü erişim kontrol mekanizmaları kullanın, veri bütünlüğünü korumak için denetim izlerini ve şifrelemeyi uygulayın.
4. **Tedarik Zinciri Güvenliğine Odaklanma:**
    - Üçüncü taraf yazılım bileşenlerinin güvenliğini sağlamak için güvenlik taramaları yapın ve sadece güvenilir kaynaklardan temin edilen bileşenleri kullanın.
5. **Girdi Doğrulama ve Sanitizasyon Uygulama:**
    - Veri bütünlüğünü korumak için kullanıcı girdilerini doğrulayan ve sanitasyon yapan mekanizmalar uygulayın. Bu, zararlı verilerin sisteme girmesini engeller.
6. **Güvenli Kodlama Uygulamaları Benimseme:**
    - Yazılım geliştirirken güvenli kodlama uygulamalarına dikkat edin, özellikle veri işleme ve depolama işlemlerinde güvenlik önlemleri alın.

9) **Security Logging and Monitoring Failures** :

a) Zafiyet Nedir?

Security Logging and Monitoring Failures (Güvenlik Günlükleme ve İzleme Hataları), bir sistemde gerçekleşen olayların veya saldırı girişimlerinin yeterince kaydedilmemesi veya izlenmemesi sonucunda ortaya çıkan güvenlik zafiyetlerini ifade eder. Bu zafiyetler, bir saldırı gerçekleştiğinde veya gerçekleşmek üzereyken sistem yöneticilerinin durumu fark edememesi ve zamanında müdahale edememesi riskini doğurur. Etkili bir güvenlik denetimi ve olaya müdahale süreci için olayların doğru bir şekilde günlüklenmesi ve sürekli izlenmesi kritik öneme sahiptir.

b. Neden Kaynaklanır?

Security Logging and Monitoring Failures zafiyetleri şu nedenlerden kaynaklanabilir:

1. **Yetersiz Günlükleme Yapılandırması:**
    - Sistemlerdeki olayların, hataların veya kullanıcı aktivitelerinin yeterince ayrıntılı bir şekilde kaydedilmemesi.
2. **Günlük Kayıtlarının Yetersiz Depolanması:**
    - Günlük dosyalarının uygun bir süre boyunca saklanmaması veya yedeklenmemesi, bu verilerin bir saldırı sonrası analiz edilememesine yol açabilir.
3. **İzleme Sistemlerinin Eksikliği:**
    - Sistemlerin veya ağların sürekli olarak izlenmemesi, şüpheli aktivitelerin fark edilmemesine neden olabilir.
4. **Günlük Analizinin Yapılmaması:**
    - Günlük dosyalarının analiz edilmemesi veya bu sürecin otomatikleştirilmemesi, potansiyel güvenlik tehditlerinin gözden kaçmasına neden olabilir.
5. **Yanıt Sürelerinin Yetersizliği:**
    - İzleme sırasında tespit edilen olaylara yeterince hızlı yanıt verilememesi, zararın artmasına neden olabilir.

c) Örnek

Yetersiz günlükleme örneği:

Kullanıcı oturum açma işlemi, ancak olay günlüklenmiyor

```
session_start();
```

Oturum başlatma kodları burada

Bu örnekte, oturum açma işlemi gerçekleştirilirken olayın günlüğe kaydedilmemesi, oturum açma ile ilgili güvenlik olaylarının izlenememesine neden olabilir.

d) Nasıl Önlenir?

1. **Detaylı Günlükleme Yapılandırması:**
    - Tüm kritik olayların ve kullanıcı işlemlerinin detaylı bir şekilde kaydedildiğinden emin olun. Günlükleme yapılandırmalarını düzenli olarak gözden geçirin ve güncelleyin.
2. **Günlük Kayıtlarının Güvenli Depolanması:**
    - Günlük dosyalarını güvenli bir ortamda saklayın ve yedekleyin. Bu dosyaların belirli bir süre boyunca tutulmasını sağlayın.
3. **Sürekli İzleme ve Alarm Sistemleri:**
    - Sistem ve ağ izleme araçları kullanarak sürekli izleme sağlayın. Şüpheli aktiviteleri anında tespit edebilmek için otomatik alarm sistemleri kurun.
4. **Otomatik Günlük Analiz Araçları Kullanma:**
    - Günlük dosyalarını analiz eden otomatik sistemler kullanarak, büyük veri hacimlerinde potansiyel tehditleri hızlıca tespit edin.
5. **Etkin Yanıt ve Müdahale Süreçleri Belirleme:**
    - İzleme sistemlerinden gelen alarmlara hızlı yanıt verebilmek için etkili bir olay müdahale planı oluşturun ve bu planı düzenli olarak test edin.
6. **Günlük Kayıtlarının Bütünlüğünü Sağlama:**
    - Günlük dosyalarının değiştirilmesini veya silinmesini önlemek için dijital imza veya hashing gibi teknikler kullanarak bütünlüğünü sağlayın.

10) **Server-Side Request Forgery** :

a. Zafiyet Nedir?

Server-Side Request Forgery (SSRF), bir saldırganın, hedef sistemde bulunan bir uygulamanın sunucu tarafında istek yapmasını sağladığı bir güvenlik zafiyetidir. Bu saldırı türünde, saldırgan hedef sunucuyu başka bir sunucuya veya servise istek göndermesi için manipüle eder. Bu, saldırganın sunucunun iç

ağındaki veya korumalı ortamlardaki kaynaklara erişmesine ve potansiyel olarak hassas bilgileri elde etmesine olanak tanır.

SSRF zafiyeti, özellikle güvenlik kontrollerinin yalnızca istemci tarafında uygulanması veya sunucunun yetkisiz bağlantılara karşı yeterince korunmadığı durumlarda ortaya çıkar. Bu zafiyet, hem dahili sistemleri hedef almak hem de harici hizmetleri kötüye kullanmak için kullanılabilir.

b) Neden Kaynaklanır?

SSRF zafiyeti genellikle şu nedenlerden kaynaklanır:

1. **Güvenli Olmayan URL İşleme:**
    - Kullanıcı tarafından girilen URL'lerin yeterince doğrulanmadan işlenmesi. Bu, saldırganların zararlı URL'leri sunucu tarafından işlenmesini sağlamasına yol açabilir.
2. **Ağ İzolasyon Eksikliği:**
    - İç ve dış ağlar arasında yeterli izolasyonun olmaması, sunucunun iç ağ kaynaklarına doğrudan erişmesine imkan tanır ve bu da SSRF saldırılarına karşı hassasiyet oluşturur.
3. **Yetersiz Erişim Kontrolleri:**
    - Sunucunun hangi URL'lere veya servislere erişebileceğine dair uygun erişim kontrollerinin olmaması, yetkisiz bağlantıların yapılmasına izin verir.
4. **Yetersiz Girdi Doğrulama:**
    - Sunucu tarafından işlenecek olan kullanıcı girdilerinin yeterince doğrulanmaması veya filtrelenmemesi. Bu durum, saldırganın zararlı girdilerle sunucu isteklerini manipüle etmesine olanak tanır.

c) Var ise Türleri ve Kısa Açıklamaları

1. **Basit SSRF:**
    - **Tanım:** Saldırganın, sunucunun herhangi bir URL'ye istek yapmasını sağladığı durum. Örneğin, bir dosya yükleme işlevi kullanılarak iç ağa yapılan istekler.
2. **Blind SSRF (Kör SSRF):**
    - **Tanım:** Saldırganın sunucu üzerinden yaptığı isteklerin sonuçlarını doğrudan göremediği, ancak sunucunun başka yerlere istek gönderdiği durumlardır. Saldırgan, isteklerin etkilerini dolaylı yollarla ölçmeye çalışır.

3. **Recursive SSRF:**
    - **Tanım:** Saldırganın, sunucunun aynı isteği tekrar tekrar yapmasını sağladığı durum. Bu, sunucu üzerinde yüksek yük oluşturabilir ve potansiyel olarak bir hizmet aksatma (DoS) saldırısına yol açabilir.

d) Örnek:

- **Kod Örneği:** Basit bir SSRF zafiyetinin örneği:
-

```
<?php
$url = $_GET['url'];
$response = file_get_contents($url);
echo $response;
?>
```

Bu kodda, kullanıcı tarafından sağlanan URL doğrulanmadan file_get_contents fonksiyonu ile sunucu tarafında işleniyor. Bu durum, saldırganın sunucunun iç ağında veya başka sunuculara istek yapmasını sağlayabilir.

e) Nasıl Önlenir?

1. **Girdi Doğrulama ve Filtreleme:**
    - Kullanıcı tarafından sağlanan tüm URL girişlerini sıkı bir şekilde doğrulayın ve filtreleyin. Sadece izin verilen domainlere veya IP adreslerine erişimi sağlayacak beyaz listeleme uygulayın.
2. **Ağ İzolasyonu Sağlama:**
    - İç ve dış ağlar arasında güçlü bir izolasyon sağlayın. Sunucunun yalnızca belirli, güvenilir kaynaklara erişebilmesini sağlayın.
3. **Erişim Kontrolleri Uygulama:**
    - Sunucunun sadece belirli servis veya IP adreslerine erişmesine izin verin. Erişim kontrol listeleri (ACL) kullanarak erişim haklarını sınırlayın.
4. **URL Yönlendirmelerini Engelleme:**
    - Sunucunun, kullanıcı tarafından sağlanan URL'lerde yönlendirmeleri takip etmesini engelleyin. Bu, saldırganların zararlı sitelere yönlendirme yapmasını önler.
5. **Güvenlik Duvarı ve WAF Kullanma:**
    - Güvenlik duvarları ve Web Application Firewall (WAF) kullanarak, potansiyel SSRF saldırılarını tespit edin ve engelleyin. Bu tür güvenlik çözümleri, şüpheli istekleri engelleyebilir.
6. **Günlükleme ve İzleme Uygulama:**

- Sunucunun yaptığı tüm dış istekleri izleyin ve günlükleyin. Anormal davranışları tespit etmek için bu günlükleri düzenli olarak analiz edin.

7. **İstek Zaman Aşımı ve Kaynak Sınırlamaları:**
    - Sunucunun dış kaynaklara yaptığı isteklerde zaman aşımı ve kaynak sınırlamaları uygulayın. Bu, Recursive SSRF gibi saldırıları sınırlayabilir.

Bu raporumda sizlere OWASP TOP 10'daki zafiyetlerin ne olduğunu, nasıl olduğunu, türleri varsa neler olduğunu ve nasıl önleyebileceğimizi anlattım.