**OWASP TOP 10**

OWASP Top 10, web uygulama güvenliğinde en yaygın ve kritik zafiyetleri belirleyen bir liste sunar. Bu liste, yazılım geliştiricilerine ve siber güvenlikçilere rehberlik ederek yardımcı olur. OWASP Top 10 düzenli olarak güncellenmektedir.

**A01: Broken Access Control**

**Zafiyet Nedir?**

Broken Access Control, kullanıcıların yetkileri dışında işlemler yapmasına izin veren bir güvenlik açığıdır.

**Neden Kaynaklanır?**

Genellikle hatalı yapılan erişim kontrollerinden veya eksik yetkilendirme denetimlerinden kaynaklanır.

**Türleri:**

1. **Vertical Privilege Escalation:** Daha yüksek yetkili işlemler yapılabilir.
2. **Horizontal Privilege Escalation:** Diğer kullanıcıların verilerine erişim sağlanabilir.

**Örnek Kod:**

<?php

if($\_SESSION['role'] {

echo "Hoş geldiniz, kullanıcı!";

}

?>

**Nasıl Önlenir?**

Sıkı erişim kontrolleri uygulanmalı ve düzenli güvenlik testleri yapılmalıdır. Kullanıcılara yalnızca ihtiyaçları olan yetkiler verilmelidir.

**A02: Cryptographic Failures**

**Zafiyet Nedir?**

Cryptographic Failures, verilerin yetersiz şifrelenmesi veya hiç şifrelenmemesi sonucu oluşan güvenlik zafiyetleridir. Bu zafiyetler, hassas verilerin yetkisiz kişiler tarafından ele geçirilmesine yol açabilir.

Bu zafiyet genellikle zayıf veya hatalı şifreleme yöntemlerinin kullanılması ve şifreleme anahtarlarının kötü yönetimi ya da verilerin açık olarak saklanması gibi nedenlerden kaynaklanır.

**Türleri:**

1. **Zayıf Şifreleme Algoritmaları:** MD5 gibi artık güvenli olmayan algoritmaların kullanılması.
2. **Şifresiz Veri Saklama:** Hassas verilerin hiç şifrelenmeden depolanması.

**Örnek Kod:**

<?php

//Güvensiz bir şekilde parola saklama örneği:

$password = "user\_password";

file\_put\_contents("passwords.txt", $password);

// MD5 gibi zayıf algoritma ile şifreleme:

$hashed\_password = md5($password);

?>

**Nasıl Önlenir?**  
Güçlü şifreleme algoritmaları kullanılmalı ve önemli veriler mutlaka şifrelenmelidir. Şifreleme anahtarları güvenli bir şekilde saklanmalıdır.

**A03: Injection**

**Zafiyet Nedir?**  
Injection, kullanıcının girdileri üzerinden uygulamaya zararlı komutlar enjekte edilmesiyle ortaya çıkan bir güvenlik zafiyetidir. Bu zafiyet, veritabanı sorguları, işletim sistemi komutları gibi kritik işlemlerin yetkisiz bir şekilde çalıştırılmasına neden olabilir.

**Neden Kaynaklanır?**  
Bu zafiyet, kullanıcının girdilerinin doğrulanmadan veya güvenli hale getirilmeden doğrudan uygulama içerisinde kullanılmasından kaynaklanır.

**Türleri:**

1. **SQL Injection:** Zararlı SQL sorgularının enjekte edilmesi.
2. **Command Injection:** Zararlı sistem komutlarının çalıştırılması.

**Örnek Kod:**

<?php

// Bu kodda SQL kodlarını enjekte ederek veritabanındaki diğer verilere erişime olanak tanır.

$kullaniciadi = $\_GET[' kullaniciadi '];

$sorgu = "SELECT \* FROM kullanıcılar WHERE kullaniciadi = '$kullaniciadi '";

$cikti = mysqli\_query($baglanti, $sorgu);

?>

**Nasıl Önlenir?**

Kullanıcı girdileri her zaman doğrulanmalı, kontrol edilmeli ve güvenli hale getirilmelidir. Parametreli sorgular ve hazırlıklı ifadeler (örn: prepared statements) kullanılmalıdır.

**A04: Insecure Design**

**Zafiyet Nedir?**  
Insecure Design, uygulamanın kodlama aşamasında kod güvenliğinin yeterince dikkate alınmaması sonucu ortaya çıkan zafiyetlerdir. Bu, güvenlik açıklarının kod temelinde yer aldığı anlamına gelir.

**Neden Kaynaklanır?**  
Bu zafiyet, güvenlik önlemlerinin alınmaması, tehdit tespitinin yapılmaması veya güvenlik kontrollerinin kodlama aşamasında göz ardı edilmesinden kaynaklanır.

**Türleri:**

1. **Yetersiz Yetkilendirme Kontrolleri:** Kullanıcıların yetkileri dışında işlemler yapabilmesi.
2. **Güvensiz Veri Akışı:** Verilerin şifrelenmeden aktarılması veya depolanması.

**Örnek Kod:**

<?php

// Bu kodda yetkilendirme kontrolü yapılmadan kullanıcı silme işlemi gerçekleştirilir, bu da kodlama aşamasında güvenliğin ihmal edildiğini gösterir.

function deleteUser($userId) {

global $conn;

$query = "DELETE FROM users WHERE id = $userId";

mysqli\_query($conn, $query);

}

?>

**Nasıl Önlenir?**  
Güvenlik tehditleri kodlama aşamasında belirlenmeli, tehdit önlemleri alınmalı ve güvenlik kontrolleri kodlamanın bir parçası olmalıdır.

**A05: Security Misconfiguration**

**Zafiyet Nedir?**  
Security Misconfiguration, uygulamanın veya sunucuların yanlış yapılandırılması sonucu oluşan bir güvenlik zafiyetidir. Bu tür zafiyetler, saldırganların güvenlik açıklarını kolayca bulup istismar etmelerine neden olabilir.

**Neden Kaynaklanır?**  
Bu zafiyet genellikle varsayılan ayarların değiştirilmemesi, gereksiz özelliklerin etkinleştirilmesi veya güvenlik yamalarının eklenmemesi gibi nedenlerden kaynaklanır.

**Türleri:**

1. **Varsayılan Ayarların Kullanılması:** Uygulamanın varsayılan kullanıcı adı ve şifre ile bırakılması.
2. **Gereksiz Servislerin Açık Bırakılması:** Kullanılmayan servislerin aktif olması.

**Örnek Kod:**

<?php

// Bu kod, uygulamanın kodlama aşamasında hata mesajlarını detaylı bir şekilde göstermesine neden olur, bu da saldırganlara sistem hakkında bilgi vererek hatalardan oluşabilecek zaafiyetleri bulmalarına olanak tanır.

ini\_set('display\_errors', 1);

error\_reporting(E\_ALL);

?>

**Nasıl Önlenir?**  
Varsayılan ayarlar değiştirilmelidir. Gereksiz özellikler ve servisler devre dışı bırakılmalı, hata mesajları kullanıcıya gösterilmemelidir. Güvenlik yamaları düzenli olarak eklenmelidir.

**A06: Vulnerable and Outdated Components**

**Zafiyet Nedir?**  
Vulnerable and Outdated Components, uygulamanın eski veya bilinen güvenlik açıklarına sahip bileşenler kullanması durumunda oluşan bir güvenlik zafiyetidir. Bu tür zafiyetler, saldırganların bilinen güvenlik açıklarından yararlanmasına olanak tanır.

**Neden Kaynaklanır?**  
Bu zafiyet, kullanılan kütüphane, framework veya yazılım bileşenlerinin güncellenmemesi yada güvenlik yamalarının eklenmemesi nedeniyle ortaya çıkar.

**Türleri:**

1. **Eski Kütüphanelerin Kullanılması:** Güvenlik güncellemeleri yapılmamış kütüphanelerin kullanılması.
2. **Güvenlik Açıkları Olan Modüllerin Kullanılması:** Bilinen güvenlik açıklarına sahip bileşenlerin kullanımı.

**Örnek Kod (Açığın Olduğu Durum):**

<?php

// Bu kod güvenlik açıkları bilinen eski bir kütüphaneyi kullanmaktadır, bu da saldırganların bu açıkları istismar etmesine neden olabilir.

require 'old\_library.php';

$oldLibrary = new OldLibrary();

$oldLibrary->execute();

?>

**Nasıl Önlenir?**  
Kullanılan bileşenler ve kütüphaneler düzenli olarak güncellenmeli ve güvenlik yamaları uygulanmalıdır. Eski veya güvenliği sağlanamayan bileşenlerden kaçınılmalıdır.

**A07: Identification and Authentication Failures**

**Zafiyet Nedir?**  
Identification and Authentication Failures, kullanıcı kimlik doğrulama süreçlerinde meydana gelen güvenlik açıklarıdır. Bu tür zafiyetler, kimlik doğrulamanın atlatılmasına veya saldırganların yetkisiz erişim sağlamasına neden olabilir.

**Neden Kaynaklanır?**  
Bu zafiyet genellikle zayıf parolalar, oturum yönetimindeki hatalar veya çok faktörlü kimlik doğrulamanın eksik olmasından kaynaklanır.

**Türleri:**

1. **Zayıf Parola Politikaları:** Kullanıcıların kolay tahmin edilebilir parolalar seçmesine izin verilmesi.
2. **Oturum Yönetim Hataları:** Oturumların güvenli bir şekilde sonlandırılmaması veya çalınması.

**Örnek Kod:**

<?php

// Bu kod zayıf kimlik doğrulama ve şifreleme teknikleri kullanıyor, buda saldırganların kaba kuvvet saldırıları (bruteforce) ile kimlik doğrulama süreçlerini atlatmasına yol açabilir.

$username = $\_POST['username'];

$password = $\_POST['password'];

$query = "SELECT \* FROM users WHERE username = '$username' AND password = '$password'";

$result = mysqli\_query($conn, $query);

if(mysqli\_num\_rows($result) > 0) {

echo "Giriş başarılı!";

} else {

echo "Giriş başarısız!";

}

?>

**Nasıl Önlenir?**  
Güçlü parola politikaları uygulanmalı, oturumlar güvenli bir şekilde yönetilmeli ve mümkünse çok faktörlü kimlik doğrulama kullanılmalıdır. Parolalar güvenli bir şekilde şifrelenmeli ve kimlik doğrulama süreçleri sıkı bir şekilde denetlenmelidir.

**A08: Software and Data Integrity Failures**

**Zafiyet Nedir?**  
Software and Data Integrity Failures, yazılım veya verilerin korunamaması durumunda ortaya çıkan güvenlik zafiyetleridir. Bu, kötü niyetli saldırıların veya hataların sistemdeki yazılımı veya verileri bozmasına olanak tanır.

**Neden Kaynaklanır?**  
Bu zafiyet, yazılım güncellemelerinin güvenli bir şekilde doğrulanmaması, verilerin manipülasyona açık olması veya kod imzalama gibi güvenlik önlemlerinin eksik olmasından kaynaklanır.

**Türleri:**

1. **Güvenli Olmayan Yazılım Güncellemeleri:** Güncellemelerin kaynağının doğrulanmaması.
2. **Manipülasyona Açık Veritabanları:** Veritabanlarının yeterince korunmaması.

**Örnek Kod (Açığın Olduğu Durum):**

<?php

// Bu kod yazılım güncellemelerini indirip kurarken herhangi bir kontrol yapmaz, bu da zararlı bir güncellemenin sisteme entegre edilmesine neden olabilir.

$updateUrl = "http://example.com/update.zip";

file\_put\_contents("update.zip", file\_get\_contents($updateUrl));

shell\_exec("unzip update.zip -d /app/");

?>

**Nasıl Önlenir?**  
Yazılım güncellemeleri ve veri işlemleri sırasında bütünlük kontrolleri yapılmalı, dijital imzalar kullanılmalı ve güvenilir kaynaklardan gelen güncellemeler uygulanmalıdır. Veritabanı ve diğer veri depolama alanları güvenli hale getirilmelidir.

**A09: Security Logging and Monitoring Failures**

**Zafiyet Nedir?**  
Bu zafiyet, güvenlik olaylarının yeterince kaydedilmemesi veya izlenmemesi durumunda ortaya çıkar. Bu durum saldırıların fark edilmemesine ve zamanında müdahale edilememesine yol açabilir.

**Neden Kaynaklanır?**  
Loglama yapılmaması, yetersiz izleme mekanizmaları veya logların düzenli olarak analiz edilmemesinden kaynaklanır.

**Türleri:**

1. **Yetersiz Loglama:** Güvenlik olaylarının veya kullanıcı faaliyetlerinin yeterince kaydedilmemesi.
2. **Eksik İzleme:** Güvenlik loglarının izlenmemesi veya geç analiz edilmesi.

**Örnek Kod:**

<?php

// Bu Kodda Giriş denemeleri loglanmıyor, güvenlik olayları izlenmiyor

if(mysqli\_num\_rows($result) > 0) {

echo "Giriş başarılı!";

} else {

echo "Giriş başarısız!";

}

?>

**Nasıl Önlenir?**  
Güvenlik olayları düzenli olarak loglanmalı ve bu loglar aktif olarak izlenmeli, analiz edilmelidir.

**A10: Server-Side Request Forgery (SSRF)**

**Zafiyet Nedir?**  
Server-Side Request Forgery (SSRF) saldırganın hedef sunucuya kendi kontrolünde olan istekler göndermesine ve bu istekler aracılığıyla sunucu üzerindeki veya sunucunun bağlı olduğu diğer sistemlerdeki kaynaklara yetkisiz erişim sağlamasına olanak tanıyan bir güvenlik zafiyetidir. Bu zafiyet sunucunun diğer iç ağlara, hizmetlere veya dosyalara erişmesine de yol açabilir.

**Neden Kaynaklanır?**  
Bu zafiyet genellikle sunucunun kullanıcı tarafından sağlanan URL veya diğer kaynakları doğrulamadan işlem yapmasından kaynaklanır. Sunucu, gelen istekleri yeterince filtrelemediği veya sınırlandırmadığı için saldırgan sunucuyu kötü niyetli bir şekilde kullanabilir.

**Türleri:**

1. **Basit SSRF:** Saldırganın, sunucunun belirtilen bir URL'ye istek göndermesini sağlaması. Örneğin, saldırgan kendi kontrolündeki bir sunucuya istek göndertebilir.
2. **Blind SSRF:** Saldırganın isteğin sonuçlarını doğrudan görmediği, ancak sunucunun başka bir sisteme istek gönderdiği senaryolar. Bu tür saldırılar, sonuçların doğrudan gözlemlenemediği, ancak etkilerinin başka yollarla ölçülebildiği durumlarda gerçekleşir.

**Örnek Kod:**

<?php

// Kullanıcının sağladığı URL'ye doğrudan istek gönderiliyor

$targetUrl = $\_GET['url'];

$response = file\_get\_contents($targetUrl);

echo $response;

?>

Bu kod, kullanıcıdan gelen URL'yi doğrulamadan işleyip sunucunun bu URL'ye istek göndermesine neden olur. Bu, saldırganın sunucuyu, yetkisiz kaynaklara erişmek veya başka sistemlere saldırmak için kullanmasına olanak tanır.

**Nasıl Önlenir?**  
SSRF zafiyetini önlemek için şu önlemler alınmalıdır:

* **Girdi Doğrulama:** Kullanıcıdan gelen URL'ler sıkı bir şekilde doğrulanmalı ve sadece belirli, güvenilir kaynaklara istek gönderilmesine izin verilmelidir.
* **Beyaz Liste:** Güvenilir kaynaklar için bir beyaz liste oluşturulmalı ve sadece bu listedeki URL'lere erişim sağlanmalıdır.
* **İstekleri Sınırlandırma:** İç ağlara veya yerel dosya sistemine yapılan istekler sınırlandırılmalı veya tamamen engellenmelidir.
* **HTTP Metotlarının Kısıtlanması:** Yalnızca gerekli HTTP metotlarına (örneğin GET) izin verilmeli ve diğer metotlar devre dışı bırakılmalıdır.
* **Güvenlik Duvarı ve Ağ Kontrolleri:** İç ağlara yönelik erişimleri kısıtlayacak güvenlik duvarı kuralları uygulanmalıdır.