**1) XSS (Cross‑Site Scripting) nedir?**

XSS, bir saldırganın kötü amaçlı istemci‑tarafı kodu (çoğunlukla JavaScript) bir web sayfasına yerleştirip, o sayfayı ziyaret eden diğer kullanıcıların tarayıcılarında çalıştırmasına yol açan zaafiyettir. Amaç genelde oturum çalma, yönlendirme, içerik değiştirme veya kullanıcı bilgilerini okumaktır.

**2) XSS türleri (kısa)**

* **Stored (Persistent) XSS:** Kötü içerik sunucuda (ör. veritabanında) saklanır; sonrasında birçok kullanıcıya gösterilir.
* **Reflected (Non‑persistent) XSS:** Kötü içerik, kullanıcı isteğiyle (örn. URL parametresi, form) anlık olarak sayfaya yansır ve hemen çalışır.
* **DOM‑based XSS:** Zararlı veri tamamen istemci‑taraflı kodun (JavaScript) DOM üzerinde hatalı kullanımından kaynaklanır; sunucu hiçbir değişiklik yapmaz.

**3) Bu zaafiyet nasıl ortaya çıkar? (nedenleri)**

XSS genelde şu hatalardan kaynaklanır:

* **Girdi doğrulama/temizleme yok:** Kullanıcı verisi doğrudan sayfaya/HTML'e eko ediliyor.
* **Çıkış kaçış (output escaping) yapılmıyor:** Tarayıcıya gönderilen veri HTML, attribute, JavaScript veya URL bağlamına uygun şekilde kaçışlanmıyor.
* **İstemci‑taraflı tehlikeli API kullanımı:** innerHTML, document.write, eval, setInnerHTML vb. fonksiyonlarla ham veri DOM'a ekleniyor.
* **Güvensiz şablon kullanımı:** Şablon motorunda kullanıcı verisi otomatik kaçışlanmıyorsa.
* **Yanlış içerik güvenlik yapılandırması:** CSP (Content‑Security‑Policy) yok veya yanlış yapılandırılmış.
* **Güvenlikli çerez ayarlarının eksikliği:** HttpOnly, Secure, SameSite gibi ayarlar kullanılmıyor (XSS etkisini artırır).

**4) Zaafiyeti yetkili ve güvenli şekilde tespit etme (etik yöntemler)**

**Uyarı:** Güvenlik testleri yalnızca sizin kontrolünüzdeki sistemlerde veya açık izin alınmış sistemlerde yapılmalıdır. İzinsiz testler yasal sorumluluk doğurur.

Tespit yöntemleri (etik / savunma amaçlı):

1. **Kod incelemesi (Code review):**
   * Sunucudan gelen veya kullanıcı tarafından sağlanan verinin üretken (render) olduğu noktaları bulun.
   * Bu verinin hangi bağlamda kullanıldığını (HTML gövdesi, attribute, JavaScript, URL) tespit edin — bağlama göre kaçış gerekir.
2. **Güvenli input/output kontrolü:**
   * Formlar, arama kutuları, URL parametreleri, kullanıcı profili alanları, yorum alanları gibi tüm giriş noktalarını listeleyin.
   * Bu alanlardan gelen verinin sayfada nasıl kullanıldığına bakın.
3. **Manuel davranış testi (kontrollü):**
   * Test ortamında “HTML/JS içerebilecek” birkaç zararsız işaret (ör. <tag> içeren metin) girin ve sayfada bu metnin *işaret olarak mı* yoksa *düz metin* olarak mı görüntülendiğini kontrol edin.
   * Eğer HTML/JS içeriği olduğu gibi yorumlanıp çalıştırılıyorsa zaafiyet olabilir.
4. **Otomatik tarayıcılar ve araçlar:**
   * OWASP ZAP, Burp Suite (Community), Nikto gibi güvenlik tarayıcılarını yetkili test ortamında kullanın. Bu araçlar zayıf noktaları tespit etmede yardımcı olur.
   * Sonuçları elle doğrulayın; otomatik sonuçlar yanlış pozitif/negatif verebilir.
5. **DOM analiz araçları:**
   * Tarayıcı geliştirici araçlarıyla DOM değişikliklerini izleyin; istemci JS’in kullanıcı verisini nasıl kullandığını gözlemleyin.

**5) Nasıl önlenir? (geliştirici/pratik checklist)**

Aşağıdaki önlemler bir arada uygulandığında XSS riskini büyük oranda azaltır:

1. **Çıkışta uygun kaçış (output encoding) kullanın:**
   * HTML içeriği için HTML kaçış, HTML attribute için attribute kaçışı, JavaScript bağlamı için JS kaçışı, URL bağlamı için URL kaçışı uygulanmalı.
   * Popüler çerçeveler (örn. Twig, Blade, React, Angular) genelde otomatik kaçış sağlar — varsayılan davranışı öğrenin ve kapatmayın.
2. **Girdi doğrulama (input validation):**
   * Gerekli olduğu kadar girdi kabul edin (length, format, karakter seti). Ancak “input validation” XSS’in tek başına çözümü değildir; mutlaka output encoding ile beraber kullanılmalı.
3. **Güvenli şablon/çeşitleri tercih edin:**
   * Kullanılan şablon motorunun otomatik kaçış özelliğini aktif tutun. Ham HTML yerleştirmek (unescaped) gerekiyorsa çok dikkatli olun.
4. **İstemci‑tarafı tehlikeli API’lerden kaçının:**
   * innerHTML, document.write, eval gibi fonksiyonları kullanmaktan kaçının veya kullandıktan sonra kesinlikle güvenli veri sağlayın.
5. **Content Security Policy (CSP) uygulayın:**
   * Inline script'leri (aynı sayfadaki script) ve dış kaynakları sınırlandırarak XSS etkisini azaltır. CSP, savunma‑in‑depth (çok katmanlı güvenlik) sağlar.
6. **Çerez güvenlik ayarları:**
   * Oturum çerezlerini HttpOnly, Secure ve uygun SameSite ile ayarlayın.
7. **Kütüphane ve bağımlılıkları güncel tutun:**
   * Framework/kitlerdeki güvenlik yamalarını takip edin.
8. **Eğitim ve güvenlik politikaları:**
   * Geliştiricilere güvenli kod yazımı eğitimi verin; kod gözden geçirme süreçleri oluşturun.