Cross-Site Scripting (XSS) Zafiyeti: Detaylı Analiz

I. XSS Nedir? (Cross-Site Scripting)

XSS (Cross-Site Scripting), web uygulamalarında en yaygın karşılaşılan güvenlik açıklarından biridir. XSS saldırısı, siber saldırganların, güvenilir bir web sitesinin içeriğine kötü amaçlı istemci tarafı kodları (çoğunlukla **JavaScript**) enjekte ederek, bu siteyi ziyaret eden diğer kullanıcıların tarayıcılarında çalıştırması esasına dayanır.

Bu zafiyet, uygulamanın kendisini (sunucuyu) değil, doğrudan uygulamanın **kullanıcılarını** hedef alır. Kullanıcının tarayıcısı, enjekte edilen kodu sitenin meşru bir parçası sanar ve tam yetkiyle çalıştırır.

Saldırının Sonuçları (Ne Yapılabilir?)

Başarılı bir XSS saldırısı, saldırgana kurbanın tarayıcısı üzerinde tam kontrol yetkisi verir. Bu yetkiyle şunlar yapılabilir:

- **Oturum Çerezlerini (Session Cookies) Çalmak:** Saldırgan, kullanıcının oturum bilgilerini içeren çerezleri ele geçirerek, kullanıcının parolasını bilmeden onun hesabına giriş yapabilir (Session Hijacking).
- **Tuş Vuruşlarını Kaydetmek (Keylogging):** Kullanıcının sayfada girdiği tüm verileri (şifreler, kredi kartı bilgileri) kaydedip saldırgana göndermek.
- **Kullanıcıyı Kandırmak (Phishing):** Sayfa içeriğini dinamik olarak değiştirerek kullanıcıyı sahte formlara hassas bilgilerini girmesi için yönlendirmek.
- **İstenmeyen Eylemleri Gerçekleştirmek (CSRF):** Kurbanın kimliğiyle, onun izni olmadan para transferi, şifre değiştirme gibi eylemleri gerçekleştirmek.
- **Tarayıcıyı Kötü Amaçlı Yazılımla Enfekte Etmek.**

II. XSS Zafiyeti Nasıl Ortaya Çıkar? (Temel Sebep)

XSS zafiyetinin temel ve tek sebebi, bir web uygulamasının **güvenilir olmayan kullanıcı girdisini** yeterince doğrulamadan veya temizlemeden doğrudan bir web sayfasına veya tarayıcıdaki Document Object Model'e (DOM) dahil etmesidir.

Temel Senaryo:

1. **Girdi (Input):** Bir web uygulaması, bir kullanıcıdan veri kabul eder (arama çubuğu, yorum formu, profil adı vb.).

2. **Güvenlik Kontrolü Eksikliği:** Uygulama, kullanıcının girdiği metnin sadece düz metin olmasını beklemesine rağmen, bu girdiyi zararlı etiketler (`<script>`, `<img>`, `<svg>`, vb.) açısından kontrol etmez.
3. **Çıktı (Output):** Uygulama, temizlenmemiş bu girdiyi olduğu gibi HTML çıktı olarak kullanıcıya geri gönderir veya veritabanına kaydeder.
4. **Çalıştırma (Execution):** Kurban kullanıcının tarayıcısı bu HTML çıktısını aldığında, içindeki kötü amaçlı JavaScript etiketini meşru sitenin bir parçası olarak algılar ve anında çalıştırır.

III. XSS Saldırı Türleri

XSS zafiyetleri, kötü amaçlı kodun web uygulaması içinde nasıl ve nerede çalıştığına bağlı olarak üç ana kategoriye ayrılır:

1. Stored XSS (Kalıcı / Depolanmış XSS)

- **Çalışma Şekli:** Saldırganın kodu, sunucudaki bir veritabanına, dosya sistemine veya başka bir depolama alanına **kalıcı olarak** kaydedilir.
- **Senaryo:** Saldırgan, bir blogun yorumlar kısmına veya bir mesaj panosuna kötü amaçlı bir JavaScript kodu yerleştirir. Uygulama bu kodu olduğu gibi veritabanına kaydeder.
- **Etkisi:** Bu yorumun yer aldığı sayfayı daha sonra ziyaret eden **her kurbanın** tarayıcısında, veritabanından çekilen zararlı kod otomatik olarak çalışır. Tek bir saldırıyla çok sayıda kurban hedeflenebilir, bu da onu en tehlikeli XSS türü yapar.

2. Reflected XSS (Yansıtılmış XSS)

- **Çalışma Şekli:** Saldırganın kodu, sunucuya bir HTTP isteği (çoğunlukla URL parametresi) ile gönderilir ve sunucu bu kodu herhangi bir doğrulama yapmadan **hemen yanıt sayfasında yansıtır**. Kod kalıcı olarak saklanmaz.
- **Senaryo:** Saldırgan, kötü amaçlı kodu içeren bir URL oluşturur. (Örn: `http://siteadi.com/arama?query=<script>alert('XSS')</script>`). Saldırgan, sosyal mühendislik yoluyla kurbanı bu linke tıklamaya ikna eder.
- **Etkisi:** Kurban linke tıkladığında, sunucu `query` parametresindeki kodu alır ve arama sonuçları sayfasında yansıtır. Kurbanın tarayıcısı kodu hemen çalıştırır.

3. DOM-Based XSS (DOM Tabanlı XSS)

- **Çalışma Şekli:** Bu saldırı, sunucuya herhangi bir istek göndermeden, **doğrudan istemci tarafında (tarayıcıda)** gerçekleşir. Kötü amaçlı kod, sayfanın Document Object Model'inde (DOM) manipülasyon yaparak çalıştırılır.
- **Zafiyet Kaynağı:** Genellikle istemci tarafındaki JavaScript kodunun, URL'nin karma (#) bölümü gibi güvenilir olmayan bir kaynaktan veri alıp bunu güvenli olmayan bir şekilde (örneğin `document.write` veya `innerHTML` kullanarak) DOM'a dahil etmesiyle ortaya çıkar.
- **Etkisi:** Sunucu tarafında temizleme veya filtreleme yapılsa bile bu saldırı işe yarayabilir, çünkü saldırı yalnızca tarayıcının kendi içinde gerçekleşir. Bu, tespit edilmesini zorlaştıran bir özelliktir.

IV. XSS Zafiyetini Ortaya Çıkarma Yöntemleri (Test Edilmesi)

Bir web uygulamasında XSS zafiyetini ortaya çıkarmak (güvenlik testi yapmak) genellikle aşağıdaki adımları ve teknikleri içerir:

1. Manuel Test (Temel Kavram Kanıtı)

Bu aşamada, saldırgan veya güvenlik araştırmacısı, kullanıcının veri girdiği ve bu verinin sayfada görüntülendiği tüm alanlara basit test komut dosyaları (payload) enjekte etmeye çalışır.

- **Temel Payload:** Girdinin sayfada çalıştırılıp çalıştırılmadığını anlamanın en basit yolu `alert()` fonksiyonunu kullanmaktır.
  - **Payload Örneği:** `<script>alert('XSS Calisti')</script>`
  - **Test Adımları:** Bu kodu bir arama çubuğuna, yorum kutusuna veya URL parametresine girerek sayfanın davranışını gözlemlersiniz. Eğer ekranda bir pop-up pencere açılırsa, XSS zafiyeti başarıyla tespit edilmiş demektir.
- **Etiketleri Kırma:** Girdi alanının kısıtlandığı durumlarda, mevcut HTML etiketlerini kapatıp yeni etiket açmak için denemeler yapılır.
  - **Örnek Senaryo:** Girdi alanı `<input value="Girdi">` şeklinde HTML'e ekleniyorsa, saldırgan şu kodu girer: `"> <script>alert(1)</script>`
    - Bu, HTML'de şöyle görünür: `<input value=""> <script>alert(1)</script>">` ve JavaScript çalışır.
- **Etiket Alternatifleri:** `<script>` etiketinin filtrelendiği durumlarda, diğer HTML etiketlerinin olay işleyicileri (event handler) kullanılır.

- **Payload Örneği:** `<img src=x onerror=alert(1)>` (Resim yüklenemezse hata fırlat ve kodu çalıştır.)
- **Payload Örneği:** `<svg/onload=alert(1)>`

2. Otomatik Tarama Araçları

Büyük ve karmaşık uygulamalarda, XSS tespiti için otomatik araçlar kullanılır. Bu araçlar, uygulamanın yüzeyini tarar ve yüzlerce farklı XSS payload'unu otomatik olarak test eder:

- **Popüler Araçlar:** Burp Suite Scanner, OWASP ZAP, Acunetix, Nessus.
- **Çalışma Prensibi:** Tarayıcılar, uygulamanın tüm giriş noktalarına (formlar, URL parametreleri, HTTP başlıkları) çeşitli payload'lar gönderir ve yanıtı analiz ederek, gönderilen kodun istemci tarafında çalışıp çalışmadığını kontrol eder.

3. Gelişmiş Tespitleme (Blind XSS)

Kalıcı (Stored) XSS türünde, enjekte edilen kodun hemen görülmediği durumlar için (örneğin, bir kullanıcının yorumu sadece site yöneticisinin panelinde görüntüleniyorsa), **Blind XSS** araçları kullanılır.

- **Prensip:** Saldırgan, **özel bir yük (payload)** enjekte eder. Bu yük, çalıştırıldığı zaman (yani yönetici sayfayı açtığında), kurbanın bilgisayarından saldırganın kontrolündeki bir sunucuya gizlice veri gönderir ve saldırının başarılı olduğunu kanıtlar.

Özetle

XSS, **güvenilir olmayan verinin çıktıda işlenmesinden** kaynaklanan kritik bir zafiyettir. Korunmanın anahtarı, kullanıcılardan gelen **tüm girdileri** güvenilir olmayan veri olarak kabul etmek ve bunu sayfa çıktısına dahil etmeden önce **girdiyi doğrulamak (Input Validation)** ve **çıktıyı uygun şekilde kodlamak (Output Encoding)** (örneğin < karakterini `<` olarak değiştirmek) prensibine dayanır.