**SSTI ve CSTI Zafiyetleri**

**SSTI ve CSTI Nedir?**

Web uygulamaları geliştirilirken, kullanıcı verilerini dinamik olarak sayfalara yerleştirmek için şablon (template) motorları kullanılır. Bu şablon motorları, geliştiricilerin kodu daha temiz ve okunabilir yazmasına olanak tanır. Ancak şablon motorlarının yanlış kullanılması ciddi güvenlik açıklarına yol açabilir.

Bu güvenlik açıklarının başında **SSTI (Server Side Template Injection)** ve **CSTI (Client Side Template Injection)** gelir.

* **SSTI**, sunucu tarafında çalışan şablon motorlarının kullanıcıdan gelen verileri doğru şekilde filtrelememesi sonucu oluşan bir güvenlik zafiyetidir. Bir saldırgan, şablon motoru üzerinden sunucu üzerinde kod çalıştırabilir veya hassas bilgilere erişebilir.
* **CSTI** ise istemci (tarayıcı) tarafında çalışan şablon motorlarında oluşan bir açık türüdür. Saldırgan, tarayıcı içinde kötü amaçlı JavaScript kodları çalıştırarak kullanıcı bilgilerini ele geçirebilir veya saldırılar düzenleyebilir.

Bu iki zafiyet de, uygulamaların güvenliğini tehlikeye atan ciddi problemler yaratabilir.

**Neden Önemlidir?**

SSTI ve CSTI zafiyetleri, saldırganların hedef uygulamalarda kontrolü ele geçirmesine, hassas bilgilere erişmesine ve sistem üzerinde yetkisiz işlemler gerçekleştirmesine olanak tanır.

Özellikle SSTI açıkları, saldırganın sunucu tarafında doğrudan kod çalıştırabilmesine (Remote Code Execution - RCE) kadar ilerleyebilir. Bu durum, sistemin tamamen ele geçirilmesine ve verilerin sızdırılmasına yol açabilir.

CSTI zafiyetleri ise istemci tarafında kötü amaçlı JavaScript kodlarının çalıştırılmasına (Cross-Site Scripting - XSS gibi) sebep olarak kullanıcıların oturum bilgilerinin çalınmasına, kimlik sahtekarlığına veya çeşitli sosyal mühendislik saldırılarına zemin hazırlayabilir.

Gerçek dünyada, şablon enjeksiyonu açıkları, hem bireysel kullanıcılar hem de kurumlar için ciddi güvenlik riskleri oluşturmakta ve veri sızıntısı, hizmet kesintisi veya maddi kayıplar gibi sonuçlara neden olabilmektedir. Bu nedenle, SSTI ve CSTI zafiyetlerinin anlaşılması ve önlenmesi, güvenli yazılım geliştirme süreçlerinin kritik bir parçasıdır.

|  |  |  |
| --- | --- | --- |
| **Zafiyet** | **Risk Seviyesi** | **Potansiyel Sonuçlar** |
| SSTI | Çok Yüksek | Sunucu kontrolü, veri sızıntısı |
| CSTI | Yüksek | Oturum hırsızlığı, XSS saldırıları |

**SSTI (Server Side Templete Injection) Nedir?**

SSTI (Server-Side Template Injection), bir web uygulamasının kullanıcıdan aldığı verileri sunucu tarafında çalışan bir şablon motoru içerisine doğrudan ve güvenlik önlemi alınmadan yerleştirmesi sonucu oluşan bir güvenlik zafiyetidir.

Şablon motorları, dinamik içerik üretmek için geliştirilmiş araçlardır ve Python (Jinja2), PHP (Smarty), Ruby (ERB) gibi birçok dilde yaygın olarak kullanılmaktadır. Eğer kullanıcı girdisi şablonun bir parçası haline getirilirken doğru şekilde filtrelenmezse, saldırganlar kendi kodlarını bu şablonlara enjekte edebilir.

Bu durum, şablon motorunun doğrudan saldırgan tarafından manipüle edilmesine ve sunucu üzerinde kötü amaçlı komutlar çalıştırılmasına (Remote Code Execution - RCE) kadar varabilecek ciddi güvenlik risklerine yol açar.

**Benzer Durum Örneği:**

Bir Flask uygulamasında kullanıcıdan alınan “name” değişkeni, doğrudan bir şablona gömülürse:

name = request.args.get('name')

render\_template\_string('Hello {{ ' + name + ' }}!')

Saldırgan, name parametresine şablon motorunun anlayacağı bir ifade ({{ 7\*7 }} gibi) göndererek, sunucuda işlem yapılmasını sağlayabilir. Sonuç olarak "Hello 49!" gibi bir çıktı üretilir. Daha ileri seviyelerde, sistemde komut çalıştırma gibi daha tehlikeli işlemler de gerçekleştirilebilir.

**SSTI'nin Ortaya Çıkabileceği Şablon Motorlarına Örnekler:**

* Python: Jinja2, Mako
* PHP: Smarty, Twig
* Java: Thymeleaf, FreeMarker
* Ruby: ERB, Liquid

SSTI zafiyetleri, çoğunlukla küçük bir ihmal sonucu ortaya çıkar ve saldırganlara çok yüksek seviyede yetki kazandırabileceği için son derece tehlikeli kabul edilir.

**CSTI (Client-Side Template Injection) Nedir?**

CSTI (Client-Side Template Injection), istemci tarafında (genellikle tarayıcıda) çalışan şablon motorlarının kullanıcıdan alınan verileri doğrudan ve yeterli kontrol olmadan işlemeleri sonucunda oluşan bir güvenlik zafiyetidir.

Modern web uygulamaları, dinamik içerik oluşturmak için sıklıkla client-side şablon motorları (örneğin AngularJS, Vue.js, Handlebars.js) kullanmaktadır. Bu motorlar, kullanıcı girdilerini işleyerek sayfa üzerinde dinamik olarak veri gösterimi sağlar. Ancak, kullanıcı verileri doğrudan şablon motoruna aktarılırken uygun güvenlik önlemleri alınmazsa, saldırganlar kendi şablon kodlarını enjekte ederek tarayıcıda kötü amaçlı JavaScript kodları çalıştırabilir.

**Benzer Durum Örneği:**

Bir web sayfasında kullanıcı adı doğrudan bir şablon ifadesi içine yerleştiriliyorsa:

<div>

{{username}}

</div>

Ve username değeri doğru şekilde filtrelenmiyorsa, bir saldırgan {{constructor.constructor('alert(1)')()}} gibi bir yük (payload) göndererek kullanıcının tarayıcısında zararlı JavaScript kodlarının çalışmasını sağlayabilir. Bu durum, klasik bir **XSS (Cross-Site Scripting)** saldırısına dönüşebilir.

**CSTI'nin Ortaya Çıkabileceği Şablon Motorlarına Örnekler:**

* AngularJS
* Vue.js
* Handlebars.js
* Mustache

CSTI zafiyetleri, saldırganlara kullanıcının oturum bilgilerini çalma, phishing saldırıları oluşturma veya kullanıcı tarayıcısında istenmeyen eylemler gerçekleştirme gibi imkanlar sunar. Bu nedenle, istemci tarafı şablon motorlarının kullanımında da kullanıcı verilerinin dikkatle işlenmesi kritik öneme sahiptir.

**SSTI Örnek Uygulama**

**Flask Uygulaması ile SSTI**

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Kullanıcı, URL'ye name adında bir parametre vererek uygulamaya veri gönderiyor.

Bu veri, doğrudan render\_template\_string() fonksiyonuna aktarılıyor.

Güvenlik kontrolü yapılmadığı için, gönderilen veri şablon motoru tarafından yorumlanıyor.

Uygulama çalıştıktan sonra URL de işlem execute edebiliyoruz. Örneğin:

http://127.0.0.1:5000/?name={{7\*7}}

ekran görüntüsü, multimedya yazılımı, yazılım, grafik yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**Daha İleri Saldırı Örneği (Tehlikeli Kullanım)**

Daha gelişmiş saldırılarla sistem komutlarına da erişilebilir. Örneğin Jinja2 tabanlı sistemlerde şu tür payload'lar kullanılabilir:

{{ config.items() }}

{{ ''.\_\_class\_\_.\_\_mro\_\_[2].\_\_subclasses\_\_() }}

Bu tür payload'lar kullanılarak sunucunun yapılandırma bilgileri veya hatta işletim sistemi üzerinde çalışan kritik nesnelere ulaşılabilir.

**Sonuç**

Bu basit örnek, kullanıcı girdisinin doğrudan şablona işlenmesinin nasıl ciddi güvenlik açıklarına yol açabileceğini göstermektedir.

Gerçek hayatta SSTI zafiyetlerinin kötüye kullanılması, veri sızıntısından sunucu ele geçirmeye kadar pek çok ciddi sonuca sebep olabilir.

CSTI Örnek Uygulama

HTML ile CSTI

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

Kullanıcı, URL'ye name adında bir parametre vererek uygulamaya veri gönderir.

Bu veri doğrudan HTML içerisine (innerHTML) yerleştirilir.

Güvenlik önlemi alınmadığı için, kullanıcı tarafından sağlanan veri doğrudan tarayıcıda işlenir.

ekran görüntüsü, metin, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

metin, ekran görüntüsü, yazılım, multimedya yazılımı içeren bir resim

Yapay zeka tarafından oluşturulan içerik yanlış olabilir.

**CSTI Zafiyetinin Sömürülmesi**

HTML dosyasını tarayıcıda açtıktan sonra aşağıdaki gibi bir URL kullanarak normal bir kullanım sağlanabilir:

file:///C:/.../csti\_example.html?name=Emir

Bu ekranda Merhaba, Emir! Gibi bir çıktı versede saldırgan zararlı bir payload gönderebilir. Örneğin:

Dosyaların bulunduğu dizinde şu komutu çalıştırmanız ve sırasıyla verilen url’leri kullanmanız adım adım bu zafiyeti gösterecektir

python -m http.server 8000 (port isteğe bağlı olarak değiştirlebilir) -> Küçük bir web sunucu açar

http://localhost:8000/csti\_ex.html?name=<img src=x onerror=alert('CSTI')> -> Zafiyetli URL

Bu durumda tarayıcıda bir JavaScript alert penceresi açılır. Bu, kullanıcıdan alınan verinin doğrudan şablon içerisine yerleştirildiği ve kötü niyetli kodların çalıştırılabildiği anlamına gelir.

Bu örnek, kullanıcı girdilerinin doğrudan şablon motorlarına veya HTML yapısına aktarılmasının nasıl güvenlik riskleri oluşturabileceğini göstermektedir.

İstemci tarafı şablon motorlarının güvenli kullanımı, kullanıcı verileri üzerinde doğru filtreleme ve doğrulama yapılması ile sağlanmalıdır.

**Sonuç:**

Bu yazıda SSTI ve CSTI zafiyetlerinin ne olduğunu, nasıl ortaya çıktıklarını ve basit örneklerle nasıl sömürülebileceklerini inceledik.  
Bu tür güvenlik açıkları, hem sunucu tarafında hem de istemci tarafında ciddi tehditler oluşturabilir.  
Geliştiricilerin şablon motorlarını kullanırken her zaman kullanıcı girdilerini güvenli bir şekilde işlediğinden emin olmaları gerekmektedir.

Özellikle veri doğrulama, çıktı kodlaması ve güvenli şablon motoru kullanımı gibi temel güvenlik pratikleri, bu zafiyetlerin önlenmesinde kritik rol oynamaktadır.

Günümüz dünyasında, küçük bir şablon hatası bile büyük veri ihlallerine ve kurumsal itibar kaybına yol açabileceği için, SSTI ve CSTI gibi zafiyetlere karşı farkındalık ve önlem almak son derece önemlidir.