Testing for Server-side Template Injection (Sunucu Tarafı Şablon Enjeksiyonu Testi)

Summary (Özet)

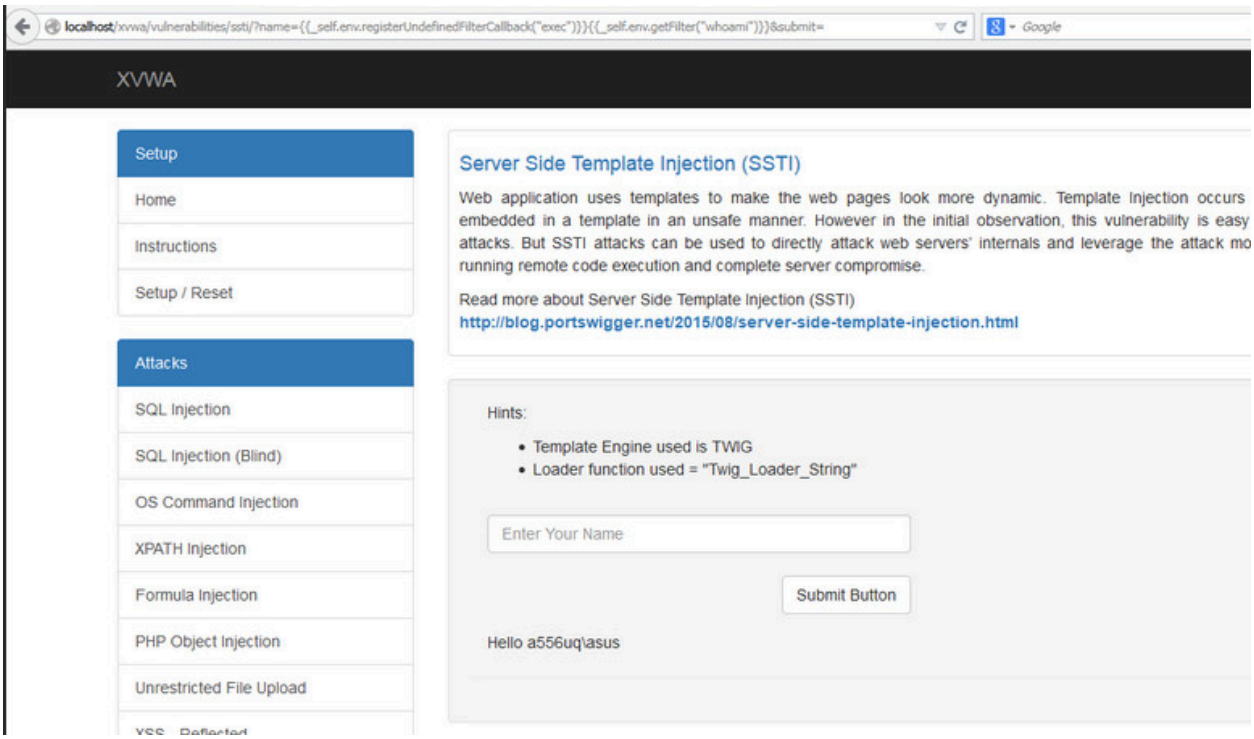
Web uygulamaları, dinamik HTML yanıtları oluşturmak için sunucu tarafı şablon teknolojilerini (Jinja2, Twig, FreeMaker vb.) yaygın olarak kullanır. Sunucu tarafı Şablon Enjeksiyonu güvenlik açıkları (SSTI), kullanıcı girişi bir şablona güvenli olmayan bir şekilde gömülü olduğunda ve sunucuda uzaktan kod yürütülmesine neden olduğunda ortaya çıkar. Gelişmiş kullanıcı tarafından sağlanan işaretlemeyi destekleyen herhangi bir özellik, wiki sayfaları, incelemeler, pazarlama uygulamaları, CMS sistemleri vb. dahil olmak üzere SSTI'ye karşı savunmasız olabilir. Bazı şablon motorları SSTI'ye karşı korunmak için çeşitli mekanizmalar (örneğin kum kutusu, listeye izin verin vb.) kullanır.

Example - Twig (Örnek - Twig)

Aşağıdaki örnek, Extreme Savunmasız Web Uygulaması projesinden bir alıntıdır.

```
public function getFilter($name)
{
        [snip]
        foreach ($this->filterCallbacks as $callback) {
        if (false !== $filter = call_user_func($callback, $name)) {
            return $filter;
        }
    }
    return false;
}
```

Get-Filter fonksiyonunda `call_user_func($callback, $name)` SSTI'ye karşı savunmasızdır: `name` parametre HTTP GET isteğinden alınır ve sunucu tarafından gerçekleştirilir:



Şekil 4.7.18-1: SSTI XVWA Örnek

Example - Flask/Jinja2 (Örnek - Flask/Jinja2)

Aşağıdaki örnek Flask ve Jinja2 ayarlayıcı motor kullanır. The (İngilizce) `page` İşlev, bir HTTP GET talebinden bir 'isim' parametresini kabul eder ve HTML yanıtı ile verir `name` Değişken içerik :

```
@app.route("/page")
def page():
    name = request.values.get('name')
    output = Jinja2.from_string('Hello ' + name + '!').render()
    return output
```

Bu kod snippet, XSS'ye karşı savunmasızdır, ancak SSTI'ye karşı da savunmasızdır. Aşağıdakileri bir yük olarak kullanmak `name` Parametre:

```
$ curl -g 'http://www.target.com/page?name={{7*7}}'
Hello 49!
```

Test Objectives (Test Hedefleri)

- Şablon enjeksiyonu güvenlik açığı puanlarını tespit edin.
- Cazibe motoru tanımlayın.
- Sömürüyü inşa et.

How To test (Nasıl Test Edilir)

SSTI güvenlik açıkları metin veya kod bağlamında mevcuttur. Düz metin bağlamında, kullanıcıların doğrudan HTML kodu ile freeform 'text' kullanmasına izin verilir. Kod bağlamında kullanıcı girişi bir şablon beyanında da yerleştirilebilir (örneğin değişken bir ad halinde)

(Şablon Enjeksiyonu Kırılganlığını Tanımlayın)

SSTI'yi düz metin bağlamında test etmenin ilk adımı, çeşitli şablon motorları tarafından yük olarak kullanılan ortak şablon ifadelerini yüklemek ve hangi şablon ifadesinin sunucu tarafından yürütüldüğünü belirlemek için sunucu yanıtlarını izlemektir.

Yaygın şablon ifade örnekleri:

```
a{{bar}}b
a{{7*7}}
{var} ${var} {{var}} <%var%> [% var %]
```

Bu adımda kapsamlı bir şablon ifadesi test dizeleri / maaş yüklemeleri listesi önerilir.

SDTI için kod bağlamında test biraz farklıdır. İlk olarak, test cihazı bu isteksizliği boş veya hata sunucusu yanıtları oluşturur. Aşağıdaki örnekte HTTP GET parametresi eklenir. Değişken `personal_greeting` Şablon beyanında:

`personal_greeting=username`
`Hello user01`

Aşağıdaki yükü kullanarak - sunucu yanıtı boş "Merhaba":

```
personal_greeting=username<tag>
Hello
```

Bir sonraki adımda, şablon beyanından çıkmak ve aşağıdaki yükü kullandıktan sonra HTML etiketini enjekte etmektir.

```
personal_greeting=username}}<tag>
Hello user01 <tag>
```

(Şablonu ayarlayan motoru tanımlayın)

Testçinin önceki adımdaki bilgilere dayanarak, test cihazının çeşitli şablon ifadeleri sağlayarak hangi şablon motorunun kullanıldığını belirlemelidir. Sunucu yanıtlarına dayanarak tester, kullanılan şablon
motorundan çıkar. Bu manuel yaklaşım bu thisPortSwigger makalesinde daha ayrıntılı olarak tartışılmaktadır. SSTI güvenlik açığının ve ayarlı motorun tanımlanmasını otomatikleştirmek için Tplmap veya Backslash Powered Scanner Burp Suite uzantısı da dahil olmak üzere çeşitli araçlar mevcuttur.

(RCE Sömürüsünü İnşa Edin)

Bu adımdaki ana amaç, şablon dokümantasyonu ve araştırmayı inceleyerek bir RCE istismarı ile sunucuda daha fazla kontrol elde etmektir. İlgi duyulan önemli alanlar şunlardır:

- Temel sözdizimi kapsayan **şablon yazarları** bölümleri için.
- **Güvenlik hususları** bölümleri.
- Yerleşik yöntemlerin, fonksiyonların, filtrelerin ve değişkenlerin listeleri.
- uzantıların/artılar listeleri.

Test cihazı ayrıca diğer nesnelerin, yöntemlerin ve özelliklerin neye odaklanarak açığaçıkabileceğini de belirleyebilir. `self` Nesne. Eğer eğer varsa `self` Nesne mevcut değildir ve dokümantasyon teknik ayrıntıları ortaya çıkarmaz, değişken adın kaba bir kuvveti önerilmektedir. Nesne tanımlandıktan sonra, bir sonraki adım, şablon motoru aracılığıyla erişilebilen tüm yöntemleri, özellikleri ve nitelikleri tanımlamak için nesneden geçmektir. Bu, ayrıcalıklı artışlar, uygulama şifreleri, API tuşları, yapılandırmalar ve ortam değişkenleri vb. Hakkında bilgi açıklamaları da dahil olmak üzere diğer güvenlik bulgularına yol açabilir.

Tools (Araçlar)

- Tplmap
- Backslash Powered Scanner Burp Suite extension
- Template expression test strings/payloads list

References (Referanslar)

- James Kettle: Server-Side Template Injection:RCE for the modern webapp (whitepaper)
- Server-Side Template Injection
- Exploring SSTI in Flask/Jinja2
- Server Side Template Injection: from detection to Remote shell
- Extreme Vulnerable Web Application
- Divine Selorm Tsa: Exploiting server side template injection with tplmap
- Exploiting SSTI in Thymeleaf