Intro to XXE

`XML External Entity (XXE) Injection` vulnerabilities occur when XML data is taken from a user-controlled input without properly sanitizing or safely parsing it, which may allow us to use XML features to perform malicious actions. XXE vulnerabilities can cause considerable damage to a web application and its back-end server, from disclosing sensitive files to shutting the back-end server down, which is why it is considered one of the Top 10 Web Security Risks by OWASP.

XML

`Extensible Markup Language (XML)` is a common markup language (similar to HTML and SGML) designed for flexible transfer and storage of data and documents in various types of applications. XML is not focused on displaying data but mostly on storing documents' data and representing data structures. XML documents are formed of element trees, where each element is essentially denoted by a `tag`, and the first element is called the `root element`, while other elements are `child elements`.

Here we see a basic example of an XML document representing an e-mail document structure:

Code: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<email>
  <date>01-01-2022</date>
  <time>10:00 am UTC</time>
  <sender>john@inlanefreight.com</sender>
  <recipients>
    <to>HR@inlanefreight.com</to>
    <cc>
        <to>billing@inlanefreight.com</to>
        <to>payslips@inlanefreight.com</to>
    </cc>
  </recipients>
  <body>
  Hello,
      Kindly share with me the invoice for the payment made on January 1, 2022.
  Regards,
  John
  </body>
</email>
```

The above example shows some of the key elements of an XML document, like:

Key	Definition	Example
`Tag`	The keys of an XML document, usually wrapped with (`<`/`>`) characters.	`<date>`
`Entity`	XML variables, usually wrapped with (`&`/`;`) characters.	`&lt;`
`Element`	The root element or any of its child elements, and its value is stored in between a start-tag and an end-tag.	`<date>01-01-2022</date>`
`Attribute`	Optional specifications for any element that are stored in the tags, which may be used by the XML parser.	`version="1.0"/encoding="UTF-8"`
`Declaration`	Usually the first line of an XML document, and defines the XML version and encoding to use when parsing it.	`<?xml version="1.0" encoding="UTF-8"?>`

Furthermore, some characters are used as part of an XML document structure, like `<`, `>`, `&`, or `"`. So, if we need to use them in an XML document, we should replace them with their corresponding entity references (e.g. `&lt;`, `&gt;`, `&amp;`, `&quot;`). Finally, we can write comments in XML documents between `<!--` and `-->`, similar to HTML documents.

XML DTD

`XML Document Type Definition (DTD)` allows the validation of an XML document against a pre-defined document structure. The pre-defined document structure can be defined in the document itself or in an external file. The following is an example DTD for the XML document we saw earlier:

Code: xml

```
<!DOCTYPE email [
  <!ELEMENT email (date, time, sender, recipients, body)>
  <!ELEMENT recipients (to, cc?)>
  <!ELEMENT cc (to+)>
  <!ELEMENT date (#PCDATA)>
  <!ELEMENT time (#PCDATA)>
  <!ELEMENT sender (#PCDATA)>
  <!ELEMENT to  (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
```

As we can see, the DTD is declaring the root `email` element with the `ELEMENT` type declaration and then denoting its child elements. After that, each of the child elements is also declared, where some of them also have child elements, while others may only contain raw data (as denoted by `PCDATA`).

The above DTD can be placed within the XML document itself, right after the `XML Declaration` in the first line. Otherwise, it can be stored in an external file (e.g. `email.dtd`), and then referenced within the XML document with the `SYSTEM` keyword, as follows:

Code: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email SYSTEM "email.dtd">
```

It is also possible to reference a DTD through a URL, as follows:

Code: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email SYSTEM "http://inlanefreight.com/email.dtd">
```

This is relatively similar to how HTML documents define and reference JavaScript and CSS scripts.

XML Entities

We may also define custom entities (i.e. XML variables) in XML DTDs, to allow refactoring of variables and reduce repetitive data. This can be done with the use of the `ENTITY` keyword, which is followed by the entity name and its value, as follows:

Code: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email [
  <!ENTITY company "Inlane Freight">
]>
```

Once we define an entity, it can be referenced in an XML document between an ampersand `&` and a semi-colon `;` (e.g. `&company;`). Whenever an entity is referenced, it will be replaced with its value by the XML parser. Most interestingly, however, we can `reference External XML Entities` with the `SYSTEM` keyword, which is followed by the external entity's path, as follows:

Code: xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE email [
  <!ENTITY company SYSTEM "http://localhost/company.txt">
  <!ENTITY signature SYSTEM "file:///var/www/html/signature.txt">
]>
```

Note: We may also use the `PUBLIC` keyword instead of `SYSTEM` for loading external resources, which is used with publicly declared entities and standards, such as a language code (`lang="en"`). In this module, we'll be using `SYSTEM`, but we should be able to use either in most cases.

This works similarly to internal XML entities defined within documents. When we reference an external entity (e.g. `&signature;`), the parser will replace the entity with its value stored in the external file (e.g. `signature.txt`). `When the XML file is parsed on the server-side, in cases like SOAP (XML) APIs or web forms, then an entity can reference a file stored on the back-end server, which may eventually be disclosed to us when we reference the entity.`

In the next section, we will see how we can use External XML Entities to read local files or even perform more malicious actions.