Exploiting XXE via File Uploads

By Neha Gupta

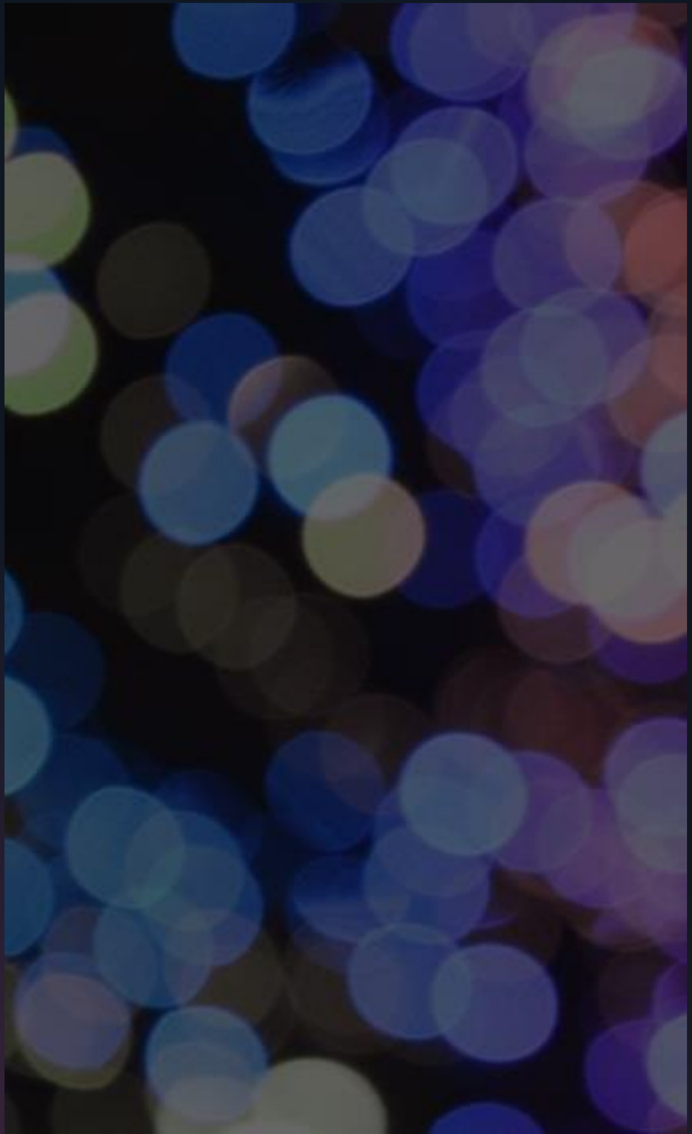
Instructions

This document talks about XXE and how you can exploit it with file upload

We will also take a look of the exploitation of the vulnerability.

Description

XXE or XML external Entity injection is a security vulnerability in an application which parses the XML inputs. The vulnerability occurs because the XML parser parsing the user inputs doesn’t perform the input validation and parses each and every instructions sent to it. It allows attacker to view files on the application server file system and interact with any backend system or application. XXE can also be used to perform SSRF or Server Side Request Forgery against the backend systems.

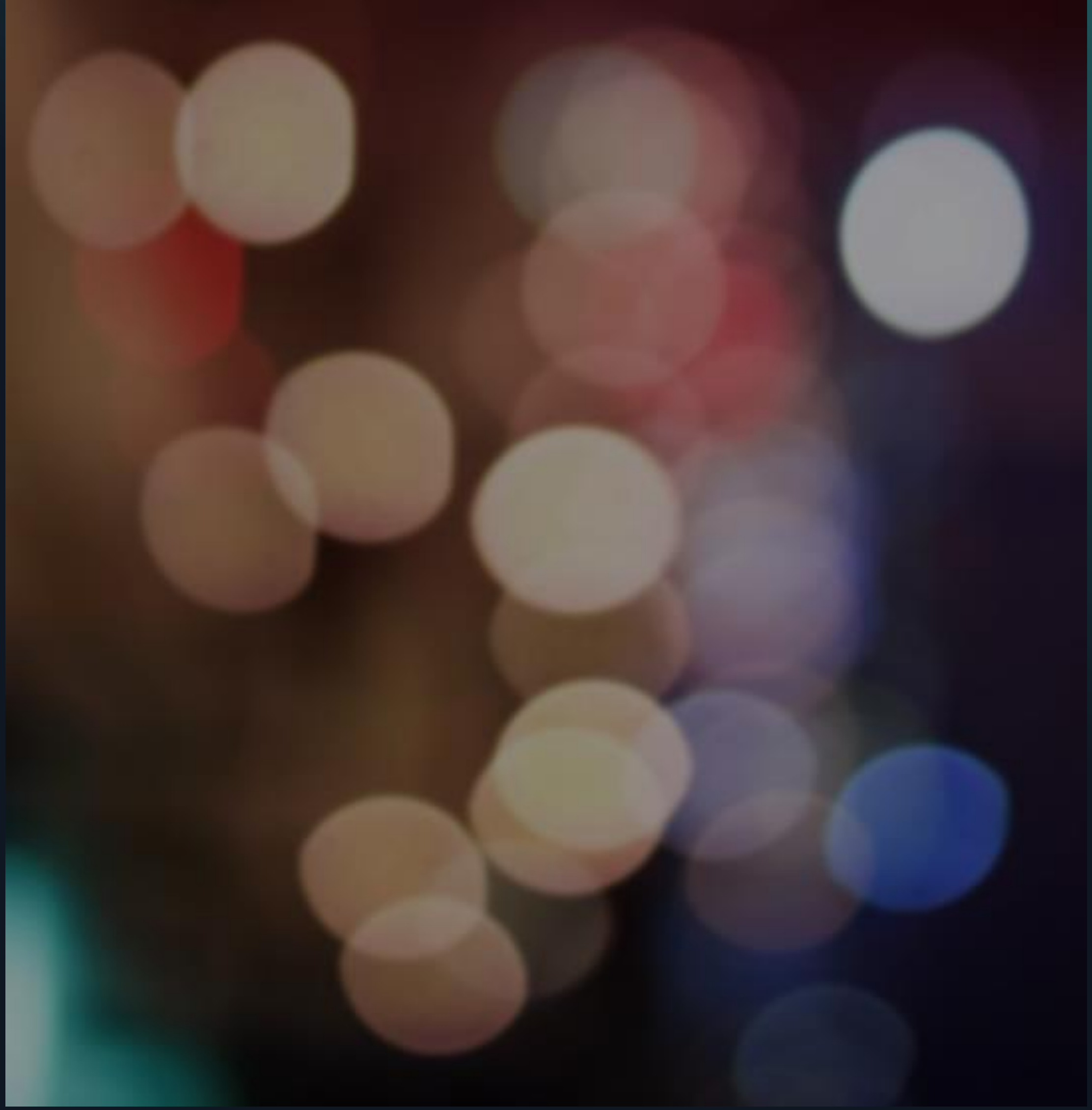


Types of XXE Attacks

- There are various types of the XXE attacks. Some common are mentioned below.

- Exploiting XXE to Retrieve files from the server
- Exploiting XXE to perform SSRF on the backend Systems.
- Blind XXE to exfiltrate the data out of band

We are only going to discuss XXE to retrieve file as an example



Exploiting XXE to Retrive files from the server

- We can also exploit the XXE to retrieve files from the system and this is the most common attack scenario of XXE. Lets take an example of this.

There is bitcoin website “example.com”, whenever a user tries buy the bitcoin the website actually makes a XML request to check the current price of bitcoin. The request which is used to check the price looks like this

```
<?xml version="1.0" encoding="UTF-8"?>
<price><current_price>btc</current_price></price>
```

You can see that this is an XML request which is being sent to check price. The parameters of this request is being parsed by an XML parser and then the response is returned to the user. In order to test the vulnerability we can craft a simple payload to check whether the application is vulnerable to XXE or not.

Exploitation

So we are crafting a request which tries to fetch the /etc/passwd file from the server. The crafted request looks like this.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [ <!ENTITY fetch SYSTEM "file:///etc/hostname">]>
<price><current_price>&fetch;</current_price></price>
```

This payload actually contains an external entity &fetch; whose value is the content of the of /etc/hostname file.

As soon as this crafted request is sent to the server, since the XML parser and the application is not validating the user inputs, the request gets processed and the attacker gets the content of the /etc/hostname file.

This also means that the application is vulnerable to XXE.

XXE Via File Upload

If the application allows user to upload svg files on the system, then the XXE can be exploited using them.

First lets discuss what are SVG files.

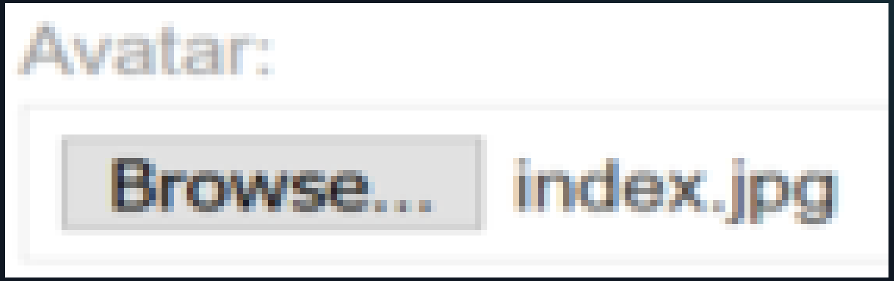
SVG Files: SVG file actually defines graphics in XML format.

Since these files defines graphics in XML format then these files create a lot of attack scenarios like we can actually execute the XSS using the SVG file and can do a lot more. We can also execute XXE using these files which

When we upload SVG image from client side, and there is no verification of content/ commands on server side. Therefore, a situation may arise where attacker can execute malicious commands to fetch the internal details. Such as fetching /etc/passwd file and if the server handling the request is using AWS then we can fetch the credentials as well

You can see that we have an option to upload the profile picture on the website. Luckily the application accepts SVG files as profile pictures as well.

So lets now create an malicious SVG file to exploit XXE



Creating of Malicious SVG file

The SVG files format starts defining the XML version first and then we can include our custom payload with some attributes such as height width and font size in the image.

Here is the malicious payload

```
<?xml version="1.0" standalone="yes"?>

<!DOCTYPE foo [ <!ENTITY fetch SYSTEM "file:///etc/passwd">]>

<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=“23" x=“8" y=“28">&fetch;</text>

</svg>
```

Lets discuss the attribute of the payload now.

Attributes in the file

I hope you remember he example of XXE to retrieve files we will use the first line as it is to retrieve the file.

<!DOCTYPE foo [<!ENTITY fetch SYSTEM "file:///etc/passwd">]>

In the next lines I have defined the width and height of the SVG files in pixels.

The another sensitive attribute in the file is

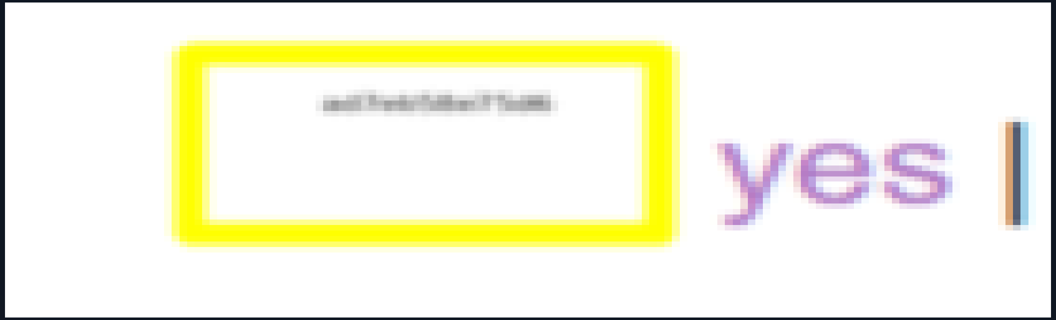
<text font-size=“10" x=“0" y=“28">&fetch;</text>

This actually defines the font-size of the characters which will be fetched from the /etc/passwd file. It is very important because if the font size is larger then you will not be able to read anything fetched from the /etc/passwd file and it is the same case if they are smaller. You can actaully modify the font size on case to case basis. The other attribute of x and y defines the axis on which the text is going to render.

Our file is now ready save this file as image.svg and upload it.

Exploitation

Upload the file and take a look at your avatar image. You will find that it have the contents of /etc/passwd file. I have blurred the image to hide the sensitive information



Remediation

- 1. If you are allowing user to upload image file, whitelist the extension which are needed
- 2. If accepting svg files are business requirements then put a restrictions in place so that the instructions in those files doesn’t get processed without any validation.

References

- https://portswigger.net/web-security/xxe
- https://gupta-bless.medium.com/exploiting-xxe-for-ssrf-c23892374c0c
- https://gupta-bless.medium.com/exploitation-xml-external-entity-xxe-1f5f3e7bc5c4