Safely handling untrusted XML server-side

Wednesday, March 13, 2013

Windows Security (/nazim/Tags/Windows%20Security) XML (/nazim/Tags/XML)

If you didn't think that processing XML on the server side can lead to a Denial of Service, Information Disclosure or even Remote Code Execution, read on. The issues discussed here are include a class of issues that is commonly referred to as XML External Entity vulnerabilities (XXE), but are not limited to this. If you are NOT processing untrusted XML and the data comes from a trusted source this article doesn't really apply for you but is still good to enforce safe usage for hygiene.

Most would consider XML as simple markup data, but you should actually consider it a 'language' that actually has powerful runtime features. These runtime features could be enabled on the server-side even if their use is not intended, and this is what unwittingly leads to XML based vulnerabilities on the server side. Following are a few of the classes of issues you could run into with XML processing.

Denial of Service

XML allows you to define entities that are essentially tokens that get replaced at runtime. Some of these are well-known, like *>* which would get replaced by the greater than sign '**>**'. But you can actually custom define entities as in the example below.

```
<!DOCTYPE doc [<!ENTITY name "Nazim">]>
<doc>
   &name;
</doc>
```

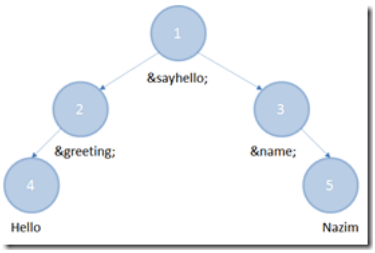
During XML pre-processing, this XML would essentially become

```
<doc>
   Nazim
</doc>
```

But we can go further and define entities that reference other entities, like

```
<!DOCTYPE doc [
  <!ENTITY greeting “Hello">
  <!ENTITY name “Nazim">
  <!ENTITY sayhello “&greeting; &name;">
]>
<doc>
   &sayhello;
</doc>
```

So now the XML pre-processor has to go through multiple stages of entity expansion.



(https://iisnetblogs.blob.core.windows.net/media/nazim/Media/image_7D522F15.png)

You have probably put 2 and 2 together at this point and see the issue already. Since this is essentially a tree structure, I can grow the number of nodes to pre-process exponentially. So something like the doc below even though small in size (<1KB) will expand out to a billion lols and can take up almost ***3GB in memory***. This is commonly referred to as the billion laughs attack and you can read more about it in this MSDN article (http://msdn.microsoft.com/en-us/magazine/ee335713.aspx).

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!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>
```

Information Disclosure

Entity expansion is not limited to string literals though. It could very well refer to external data like the example below.

```
<!DOCTYPE doc [
   <!ENTITY win SYSTEM “c:\windows\win.ini">
]>
<doc>
   &win;
</doc>
```

If this document is somehow reflected back to the client it would result in disclosing information on the server that a client wouldn’t have access to.

Remote Code Execution

I think this is best shown by directly diving into an example.

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:msxsl="urn:schemas-microsoft-com:xslt"
   xmlns:user="http://example.com/ns">
<msxsl:script language="C#" implements-prefix="user">
   <![CDATA[
   public string Code()
   {
      return Environment.MachineName;
   }
   ]]>
</msxsl:script>
<xsl:template match="/">
   <xsl:value-of select="user:Code()"/>
</xsl:template>
</xsl:stylesheet>
```

Here we have an XML style sheet that essentially has some code that is run as part of applying the style sheet. Imaging replacing the code highlighted with a payload of your choice, including things like a network scan :)

Preventing XML Entity Attacks

You essentially want to disable <!DOCTYPE> definitions (DTDs) when parsing untrusted XML. There are many APIs, that are not safe by default in this respect and you need to explicitly disable DTD resolution on an XML document. See below for API specifics.

Preventing XSL Attacks

This is pretty much the same as above and you want to disable DTD and XSL script support. See below for unsafe API specifics.

Safe XML API usage on Windows/.NET

The list below was accurate at the time this article was compiled. It is possible that updates to components may have changed the defaults, so it is **always better to test your usage**.

- System.Xml.XmlDocument
  - Load and LoadXml UNSAFE unless you pass a safe XmlReader (DTD disabled) into it during initialization.
  - InnerXml is NEVER SAFE.
- System.Xml.XmlTextReader
  - UNSAFE by default in .NET 3.5 and below.
    - You need to set ProhibitDtd=true to make this safe.
  - .NET 4.0 and above are safe be default.
- System.Xml.Xsl.XslTransform
  - UNSAFE as it supports both entities and XSL script.
- System.Xml.Xsl.XslCompiledTransform
  - Safe for XSL script since this is blocked by default.
  - UNSAFE for entity expansion unless a secure resolver is specified.
    - Pass an instance of XmlSecureResolver or null
- System.Web.UI.WebControls.XmlDataSource
  - NEVER SAFE – supports both entities and XSL script.
- MSXML 5 and below
  - UNSAFE by default
  - Need to set ProhibitDtd=true, AllowXsltScript=false and AllowDocumentFunction=false
- MSXML 6
  - Safe by default

Resources

OWASP XXE Page (https://www.owasp.org/index.php/XML_External_Entity_(XXE)_Processing)

Adobe Reader XXE Attack (http://shh.thathost.com/secadv/adobexxe/)

Attacking server side XML parsers (PDF) (http://www.exploit-db.com/download_pdf/16093/)

No Comments