Chetan Soni – Sr. Security Specialist
Email ID - chetansoni@live.com

XPATH Injection

In a typical Web Application architecture, the data is stored on a Database server. This Database server store data in various formats like an **LDAP**, **XML** or **RDBMS** database.

The application queries the server and accesses the information based on the user input. Normally attackers try to extract more information than allowed by manipulating or using the query with specially crafted inputs.

First we'll cover some basic concepts,

XML - XML stands for Extensible Markup Language and was designed or used to describe data.

It provide platform for programmers to create their own customized tags to store data on database server. An XML document is mostly similar to an RDBMS Database except for the way data is stored in them. In case of a normal database, data is stored in a table rows and columns and in XML the data is stored in nodes in a tree form.

XPATH - XPath injection, much like SQL injection, exists when a malicious user can insert arbitrary XPath code into form fields and URL query parameters in order to inject this code directly into the XPath query evaluation engine.

Doing so would allow a malicious user to bypass authentication (if an XML-based authentication system is used) or to access restricted data from the XML data source.

XPath Injection is an attack technique used to exploit applications that construct XPath (XML Path Language) queries from user-supplied input to query or navigate XML documents.

It can be used directly by an application to query an XML document, as part of a larger operation such as applying an XSLT transformation to an XML document, or applying an XQuery to an XML document.

Chetan Soni – Sr. Security Specialist
Email ID - chetansoni@live.com

In this, we try to inject data into an application so that it executes user-controlled XPath queries. When successfully injected, this **vulnerability** may allow the hackers to bypass complete authentication systems or access information without proper authorization.

Here's the simple xml database file,

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<chetan_database>

<chetan_user>
<username>chetansoni</username>
<password>password</password>
<account>Administrator</account>
</chetan_user>

<chetan_user>
<username>chetan</username>
<password>pass</password>
<account>Subscriber</account>
</chetan_user>

<chetan_user>
<username>chetanprojects</username>
<password>pass123</password>
<account>Subscriber</account>
</chetan_user>

</chetan_database>
```

Here's the basic format how XML file that is used to store the sensitive information.

Now if we want to retrieve the information about **Administrator** from the above XML file, we have to write XPath query as,

```
string(//chetan_user[username/text()='chetansoni' and password/text()='password']/account/text())
```

Now if the web designer has not property filtered the user input, then the hacker will be able to inject XPath code into the website and hence interfere with the query result.

Here is the example of XPath query that hacker will use to hack the XML file database:

```
string(//chetan_user[username/text()='' or '1' = '1' and password/text()='' or '1' = '1']/account/text())
```

Actual Code:-

```
<?php
$login = simplexml_load_file("chetan_database.xml");
$result=$login->xpath("//chetan_user[username/test()='".$_POST['chetansoni']." AND
password/text()='".$_POST['password']."'");
?>
```

Types of XPATH Injection –

1. Blind XPATH Injection
2. Advanced XPATH Injection

Prevention –

XPATH Injection can be prevented in the same way as SQL injection since XPath injection attacks are much like SQL injection attacks. Most of these preventative methods are the same as well to prevent other typical code injection attacks.

Input Validation: It is one of the best measures to defend applications from XPATH injection attacks. The developer has to ensure that the application does accept only legitimate input.

Parameterization: In Parameterized queries, the queries are precompiled and instead of passing user input as expressions, parameters are passed.

Most sites have a way to store data, the most common of which is a database. However some sites use XML to store data, and use a method of looking at the data known as XPath.

- When using SQL Databases to store site data, a site owner has to beware of SQL Injection attacks which can steal or alter data.
- Similarly, sites which store information using XML and XPath have to beware XPath injection to prevent data theft or modification.

Imagine that you have user data stored in an XML file, and that XML file looks something like this:

```
<users>
        <user ID =1>
                <username>Admin</username>
                <password>Password</password>
                <role>admin</role>
        </userid>
</users>
```

Looking at the code the site uses to log in, it dynamically builds an XPath query in PHP, after a user has given username and password in a form (with user and pass variables):

Chetan Soni – Sr. Security Specialist
Email ID - chetansoni@live.com

```
<?php
$login = simplexml_load_file("users.xml");
$result=$login->xpath("//User[username/test()='".$_POST['user']." AND
password/text()='".$_POST['pass']."'");
?>
```

The main problem above is that data from the user is not sanitized; the POST variable is being used to pull data directly from the login form, and placing it into the XPATH query.

At this point, an attacker could try putting in a special string for username:

```
' or 1=1 or '
```

Chetan Soni – Sr. Security Specialist
Email ID - chetansoni@live.com

Black box Testing Example –

The XPath attack pattern was first published by Amit Klein and is very similar to the usual SQL Injection.

In order to get a first grasp of the problem, let's imagine a login page that manages the authentication to an application in which the user must enter his/her username and password.

Let's assume that our database is represented by the following XML file:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<users>
<user>
<username>admin</username>
<password>server123</password>
<account>admin</account>
</user>
<user>
<username>mohit</username>
<password>Mohit123</password>
<account>guest</account>
</user>
<user>
<username>chetan</username>
<password>Chetan123</password>
<account>guest</account>
</user>
</users>
```

An XPath query that returns the account whose username is "**admin**" and the password is "**server123**" would be the following:

```
string(//user[username/text()='admin' and password/text()='server123']/account/text())
```

If the application does not properly filter user input, the tester will be able to inject XPath code and interfere with the query result. For instance, the tester could input the following values:

```
Username:     ' or '1' = '1
Password:     ' or '1' = '1
```

Using these parameters, the query becomes:

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

As in a common SQL Injection attack, we have created a query that always evaluates to true, which means that the application will authenticate the user even if a username or a password have not been provided.

And as in a common SQL Injection attack, with XPath injection, the first step is to insert a single quote (') in the field to be tested, introducing a syntax error in the query, and to check whether the application returns an error message.