# ntroduction to XPath Injection

[XML Path Language (XPath)](https://www.w3.org/TR/xpath-3/) is a query language for [Extensible Markup Language (XML)](https://datatracker.ietf.org/doc/html/rfc5364) data. Specifically, we can use XPath to construct XPath queries for data stored in the XML format. If user input is inserted into XPath queries without proper sanitization, [XPath Injection](https://owasp.org/www-community/attacks/XPATH_Injection) vulnerabilities arise similar to SQL Injection vulnerabilities.

## XPath Foundations

In order to dive into XPath, we first need to establish a baseline in XPath terminology. To do so, let us consider the following XML document:

Code: xml

<?xml version="1.0" encoding="UTF-8"?>  
   
<academy\_modules>   
 <module>  
 <title>Web Attacks</title>  
 <author>21y4d</author>  
 <tier difficulty="medium">2</tier>  
 <category>offensive</category>  
 </module>  
  
 <!-- this is a comment -->  
 <module>  
 <title>Attacking Enterprise Networks</title>  
 <author co-author="LTNB0B">mrb3n</author>  
 <tier difficulty="medium">2</tier>  
 <category>offensive</category>  
 </module>  
</academy\_modules>

An XML document usually begins with the XML declaration, which specifies the XML version and encoding. In the above XML document, the XML declaration is <?xml version="1.0" encoding="UTF-8"?>. If the declaration is omitted, the XML parser assumes the version 1.0 and the encoding UTF-8.

The data in an XML document is formatted in a tree structure consisting of nodes with the top element called the root element node. In our case, the root node is the academy\_modules node. Furthermore, there are element nodes such as module and title, and attribute nodes such as co-author="LTNB0B" or difficulty="medium". Additionally, there are comment nodes which contain comments such as this is a comment and text nodes which contain character data from element or attribute nodes such as Web Attacks and LTNB0B in our example. There are also namespace nodes and processing instruction nodes, which we will not consider here, adding up to a total of 7 different node types.

Since XML documents form a tree structure, each element and attribute node has exactly one parent node, while each element node may have an arbitrary number of child nodes. Nodes with the same parent are called sibling nodes. We can traverse the tree upwards or downwards from a given node to determine all ancestor nodes or descendant nodes.

## Nodes

Now that we have discussed the basic terminology of XPath, we can dive into the query syntax. In this module, we will only discuss the abbreviated syntax. For more details on the XPath syntax, look at the [W3C specification](https://www.w3.org/TR/xpath-3/).

Each XPath query selects a set of nodes from the XML document. A query is evaluated from a context node, which marks the starting point. Therefore, depending on the context node, the same query may have different results. Here is an overview of the base cases of XPath queries for selecting nodes:

|  |  |
| --- | --- |
| **Query** | **Explanation** |
| module | Select all module child nodes of the context node |
| / | Select the document root node |
| // | Select descendant nodes of the context node |
| . | Select the context node |
| .. | Select the parent node of the context node |
| @difficulty | Select the difficulty attribute node of the context node |
| text() | Select all text node child nodes of the context node |

We can use these base cases to construct more complicated queries. To avoid ambiguity of the query result depending on the context node, we can start our query at the document root:

|  |  |
| --- | --- |
| **Query** | **Explanation** |
| /academy\_modules/module | Select all module child nodes of academy\_modules node |
| //module | Select all module nodes |
| /academy\_modules//title | Select all title nodes that are descendants of the academy\_modules node |
| /academy\_modules/module/tier/@difficulty | Select the difficulty attribute node of all tier element nodes under the specified path |
| //@difficulty | Select all difficulty attribute nodes |

**Note:** If a query starts with //, the query is evaluated from the document root and not at the context node.

## Predicates

Predicates filter the result from an XPath query similar to the WHERE clause in a SQL query. Predicates are part of the XPath query and are contained within brackets []. Here are some example predicates:

|  |  |
| --- | --- |
| **Query** | **Explanation** |
| /academy\_modules/module[1] | Select the first module child node of the academy\_modules node |
| /academy\_modules/module[position()=1] | Equivalent to the above query |
| /academy\_modules/module[last()] | Select the last module child node of the academy\_modules node |
| /academy\_modules/module[position()<3] | Select the first two module child nodes of the academy\_modules node |
| //module[tier=2]/title | Select the title of all modules where the tier element node equals 2 |
| //module/author[@co-author]/../title | Select the title of all modules where the author element node has a co-author attribute node |
| //module/tier[@difficulty="medium"]/.. | Select all modules where the tier element node has a difficulty attribute node set to medium |

Predicates support the following operands:

|  |  |
| --- | --- |
| **Operand** | **Explanation** |
| + | Addition |
| - | Subtraction |
| \* | Multiplication |
| div | Division |
| = | Equal |
| != | Not Equal |
| < | Less than |
| <= | Less than or Equal |
| > | Greater than |
| >= | Greater than or Equal |
| or | Logical Or |
| and | Logical And |
| mod | Modulus |

## Wildcards & Union

Sometimes, we do not care about the type of node in a path. In that case, we can use one of the following wildcards:

|  |  |
| --- | --- |
| **Query** | **Explanation** |
| node() | Matches any node |
| \* | Matches any element node |
| @\* | Matches any attribute node |

We can use these wildcards to construct queries like so:

|  |  |
| --- | --- |
| **Query** | **Explanation** |
| //\* | Select all element nodes in the document |
| //module/author[@\*]/.. | Select all modules where the author element node has at least one attribute node of any kind |
| /\*/\*/title | Select all title nodes that are exactly two levels below the document root |

**Note:** The wildcard \* matches any node but not any descendants like // does. Therefore, we need to specify the correct amount of wildcards in our query. In our example XML document, the query /\*/\*/title returns all module titles, but the query /\*/title returns nothing.

Lastly, we can combine multiple XPath queries with the union operator | like so:

|  |  |
| --- | --- |
| **Query** | **Explanation** |
| //module[tier=2]/title/text() | //module[tier=3]/title/text() | Select the title of all modules in tiers 2 and 3 |

##### Table of Contents

###### Introduction to Injection Attacks

###### XPath Injection

###### LDAP Injection

###### HTML Injection in PDF Generators

###### Skills Assessment

##### My Workstation

OFFLINE

/ 1 spawns left