IDS53-J. Prevent XPath Injection

Extensible Markup Language (XML) can be used for data storage in a manner similar to a relational database. Data is frequently retrieved from such an XML document using XPaths. *XPath injection* can occur when data supplied to an XPath retrieval routine to retrieve data from an XML document is used without proper sanitization. This attack is similar to SQL injection or XML injection (see IDS00-J. Sanitize untrusted data passed across a trust boundary). An attacker can enter valid SQL or XML constructs in the data fields of the query in use. In typical attacks, the conditional field of the query resolves to a tautology or otherwise gives the attacker access to privileged information.

This guideline is a specific example of the broadly scoped IDS52-J. Prevent code injection.

XML Path Injection Example

Consider the following XML schema:

```
<users>
  <user>
    <username>Utah</username>
    <password>e90205372a3b89e2</password>
  </user>
  <user>
    <username>Bohdi</username>
    <password>6c16b22029df4ec6</password>
  </user>
  <user>
    <username>Busey</username>
    <password>ad39b3c2a4dabc98</password>
  </user>
</users>
```

The passwords are hashed in compliance with MSC62-J. Store passwords using a hash function. MD5 hashes are shown here for illustrative purposes; in practice, you should use a safer algorithm such as SHA-256.

Untrusted code may attempt to retrieve user details from this file with an XPath statement constructed dynamically from user input.

```
//users/user[username/text()='&LOGIN&' and password/text()='&PASSWORD&' ]
```

If an attacker knows that `Utah` is a valid user name, he or she can specify an input such as

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

This yields the following query string.

```
//users/user[username/text()='Utah' or '1'='1' and password/text()='xxxx']
```

Because the `'1'='1'` is automatically true, the password is never validated. Consequently, the attacker is inappropriately authenticated as user `Utah` without knowing `Utah`'s password.

Noncompliant Code Example

This noncompliant code example reads a user name and password from the user and uses them to construct the query string. The password is passed as a `char` array and then hashed. This example is vulnerable to the attack described earlier. If the attack string described earlier is passed to `evaluate()`, the method call returns the corresponding node in the XML file, causing the `doLogin()` method to return `true` and bypass any authorization.

```
private boolean doLogin(String userName, char[] password)
  throws ParserConfigurationException, SAXException, IOException, XPathExpressionException {

  DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
  domFactory.setNamespaceAware(true);
  DocumentBuilder builder = domFactory.newDocumentBuilder();
  Document doc = builder.parse("users.xml");
  String pwd = hashPassword( password);

  XPathFactory factory = XPathFactory.newInstance();
  XPath xpath = factory.newXPath();
  XPathExpression expr = xpath.compile("//users/user[username/text()='" +
       userName + "' and password/text()='" + pwd + "' ]");
  Object result = expr.evaluate(doc, XPathConstants.NODESET);
  NodeList nodes = (NodeList) result;

  // Print first names to the console
  for (int i = 0; i < nodes.getLength(); i++) {
    Node node = nodes.item(i).getChildNodes().item(1).getChildNodes().item(0);
    System.out.println( "Authenticated: " + node.getNodeValue());
  }

  return (nodes.getLength() >= 1);
}
```

Compliant Solution (XQuery)

XPath injection can be prevented by adopting defenses similar to those used to prevent SQL injection:

- Treat all user input as untrusted, and perform appropriate sanitization.
- When sanitizing user input, verify the correctness of the data type, length, format, and content. For example, use a regular expression that checks for XML tags and special characters in user input. This practice corresponds to input sanitization. See IDS52-J. Prevent code injection for additional details.
- In a client-server application, perform validation at both the client and the server sides.
- Extensively test applications that supply, propagate, or accept user input.

An effective technique for preventing the related issue of SQL injection is parameterization. Parameterization ensures that user-specified data is passed to an API as a parameter such that the data is never interpreted as executable content. Unfortunately, Java SE currently lacks an analogous interface for XPath queries. However, an XPath analog to SQL parameterization can be emulated by using an interface such as XQuery that supports specifying a query statement in a separate file supplied at runtime.

`Input File: login.qry`

```
declare variable $userName as xs:string external;
declare variable $password as xs:string external;
//users/user[@userName=$userName and @password=$password]
```

This compliant solution uses a query specified in a text file by reading the file in the required format and then inserting values for the user name and password in a `Map`. The `XQuery` library constructs the XML query from these inputs.

```
private boolean doLogin(String userName, String pwd)
  throws ParserConfigurationException, SAXException, IOException, XPathExpressionException {

  DocumentBuilderFactory domFactory = DocumentBuilderFactory.newInstance();
  domFactory.setNamespaceAware(true);
  DocumentBuilder builder = domFactory.newDocumentBuilder();
  Document doc = builder.parse("users.xml");

  XQuery xquery = new XQueryFactory().createXQuery(new File("login.xq"));
  Map queryVars = new HashMap();
  queryVars.put("userName", userName);
  queryVars.put("password", pwd);
  NodeList nodes = xquery.execute(doc, null, queryVars).toNodes();

  // Print first names to the console
  for (int i = 0; i < nodes.getLength(); i++) {
    Node node = nodes.item(i).getChildNodes().item(1).getChildNodes().item(0);
    System.out.println(node.getNodeValue());
  }

  return (nodes.getLength() >= 1);
}
```

Using this method, the data specified in the `userName` and `password` fields cannot be interpreted as executable content at runtime.

Applicability

Failure to validate user input may result in information disclosure and execution of unprivileged code.

According to OWASP [OWASP 2014],

> *[Prevention of XPath injection] requires the following characters to be removed (that is, prohibited) or properly escaped:*
>
> - `< > / ' = "` *to prevent straight parameter injection.*
> - *XPath queries should not contain any meta characters (such as* `' = * ? //` *or similar).*
> - *XSLT expansions should not contain any user input, or if they do, [you should] comprehensively test the existence of the file, and ensure that the files are within the bounds set by the Java 2 Security Policy.*

Automated Detection

Tool	Version	Checker	Description
The Checker Framework	2.1.3	**Tainting Checker**	Trust and security errors (see Chapter 8)

Bibliography

[Fortify 2008]	"Input Validation and Representation: XML Injection"
[Oracle 2011b]	Ensure Data Security
[OWASP 2014]	Testing for XPath Injection
[Sen 2007]	Avoid the Dangers of XPath Injection
[Sun 2006]	Ensure Data Security