5.1 INTRODUCTION

A code injection is an exploitation of a system weakness, to gain access to the system for the purpose of executing a malicious code, harvesting user data, and engaging in other activities. One of the commonest points of vulnerability for a code injection is a form, either on a web page or in a computer system. The code injection attack allows an attacker to access the underlying data stored in the back-end service, execute arbitrary commands at intent, and receive a dynamically generated output, such as HTML web pages. In the category of code injections, XPath Injection attacks may occur when a web site requires user-supplied information to construct an XQuery for XML data. In a typical web application architecture, all data is stored on a database server in various formats like a table in database, XML or LDAP. With the growing acceptance of XML technologies for documents and protocols, it is logical that security should be integrated with XML solutions. In a web application, an improper user input is the underlying root cause for a wide variety of attacks.

The XML Path or XPath language is used for querying information from the nodes of an XML document. XPath injection is an attack technique used to exploit applications that construct XPath (XML Path Language) queries from a user-supplied input to a query, or navigate XML documents, such as SQL in databases. The XQuery / XPath can be used directly by an
application to query an XML document (Chamberlin 2002), as part of a larger operation such as applying an XSLT transformation on XQuery to an XML document.

5.2 XPATH INJECTION MOTIVATION AND CONSEQUENCES

Basically there are two main motivations for XPath Injection. The first is hacking for profit, which is primarily targeted to obtain customer data or accessing personal information and confidential data (Vieira et al 2009). Another motivation is ideological hacking which focuses on reputation loss for the victims, including downtime and web defacement. In XPath injection, a malicious user can insert an arbitrary XPath code into the form fields and URL query parameters, in order to inject this code directly into the XPath query parser engine. Through this malicious activity, confidential information is disclosed to the threat agent (Laranjeiro et al 2009).

5.3 COMMON PREVENTIVE MEASURES FOR XPATH INJECTION

XPath injection can be prevented in the same way as SQL injection, since XPath injection attacks are much like SQL injection attacks (Mitropoulos et al 2009). Common ways to prevent XPath Injections are:

**Strong input validation** - All user-controllable input must be validated and filtered for illegal characters as well as content that can be interpreted in the context of an XPath expression. Characters such as a single-quote(‘) or operators such as or (|), and (&) and such should be filtered, if the application does not expect them in the context in which they appear. If such content cannot be filtered, it must at least be properly escaped to avoid its being interpreted as part of XPath expressions.
Use of parameterized XPath queries - Parameterization causes the input to be restricted to certain domains, such as strings or integers, and any input outside such domains is considered invalid and the query fails.

Use of custom error pages - Attackers can gather information about the nature of queries from descriptive error messages. Input validation must be coupled with customized error pages that inform about an error without disclosing information about the database or application.

So, the developer has to ensure that the application accepts only a legitimate input; another way is to use parameterized queries to prevent XPath injection. In parameterization, the queries are precompiled and instead of passing the user input as expressions, parameters are passed. For example

```
"//users/user[LoginID/text()='" + loginID+ "' and password/text()="" + password +"]"
```

The above query keeps important explicit variables, $loginID$ and $password$ from being processed as executable expressions at runtime. In this way, the execution logic and data are separated. Even then, these methods are not consistent to prevent XPath injection in web applications. Hence, a system has been proposed called XPath injection preventer, for effective detection of XPath injection vulnerabilities, by validating the inputs that will be converted in the form of an XML file with a designed schema, which is against the XPath Injection in web applications.

5.4 XPATH INJECTION PREVENTER

The proposed and designed system involves a new approach for detecting XPath injection vulnerabilities in web applications, as shown in Figure 5.1, which is part of the WAPS-CIVS. Since the validation of the input
is very essential for the XPath injection preventer system, the inputs are intercepted from the XQuery, by integrating Aspect Oriented Programming. Aspect-Oriented Programming is a good candidate for solving security issues. The AOP has been proposed as a technique for improving the separation of concerns in software systems, and for adding crosscutting functionalities without changing the business part of the application. The AOP provides specific language mechanisms that make it possible to address concerns, such as security, in a modular way. This module is used for detecting malicious input values given by the user for unearthing vulnerabilities present in the web application.

The XPath injection preventer architecture describes the XPath injection detection technique, implemented in the WAPS-CIVS. In a web application, whenever the user provides the required inputs through the web forms, all the inputs are placed into the XQuery in an appropriate place and the XQuery will be used to perform its operation at the data logic. The formation of the XQuery with the user inputs is at the application logic or the business logic.

Finally, the complete XQuery string is generated for processing the data transactions on XML databases. The generated (or framed) XQuery string may cause XPath injection in a web application. This attack is possible, only when the user inputs are passed directly to the web application. Sometimes, illegitimate inputs may lead to bypass authentication or retrieve privileged XML data. The inputs that lead to XPath injection have to be prevented to run on XML data.
Figure 5.1  XPath injection preventer system architecture
The XPath injection preventer system consists of the XQuery interceptor, XQuery analyzer, XQuery validator and the XML file generation module, all of which are implemented by means of web services to overcome limitations, like platform dependence and ease of maintenance of the web application.

Moreover, the XPath injection is not restricted only by the web form input. It is also possible by the HTTP header or by the cookie. But at the end, all the inputs would be placed into the XQuery to be run on XML data. Hence, our approach analyzes the XQuery for identifying the vulnerabilities and preventing XPath injection.

5.4.1 XQuery Interception

This module involves intercepting the user input that would be associated with the XQuery in web application. These input parameters are the source of the injection vulnerabilities and can be named as sink points, since they provide the attacker with an opportunity to make use of a vulnerable code. The input interception module is placed in the web service section, which intercepts the input through the Aspect Oriented Programming technique. Once the user provides the input in a client application, these values are given to the web application server, which in turn, sends the input parameters to the web application. These input parameters may consist of some of the malicious values. If these malicious values are directly passed to the application then, it may possibly unearth the vulnerability. Hence, it is necessary to intercept the user input parameters before the actual execution of the web application. The code defined in this module is used for intercepting the query. These methods are used for executing the query in order to obtain the results from the XML databases.
5.4.2 XQuery Analyzer

In this module the intercepted XQuery is analyzed and the input parameters are obtained in order to detect possible injections. The analyzer basically tokenizes the XQuery, and retrieves the input parameter in two categories, as XQuery keywords and non-XQuery keywords. After obtaining the different kinds of input parameters, the detection of possible vulnerabilities are analysed. This process needs to be generic and effective in order to detect any type of possible injection. Though several methods are available, a powerful technique is to use an XML file which would be validated by our designed schema. To create the XML file, an analysis of the user input is necessary. Hence, all the user inputs are intercepted and classified as XQuery keywords and Non-XQuery keywords. With the help of the classified keyword, an XML file is generated which is equivalent to the XQuery. This file is a well-formed document, platform independent and provides lesser overhead for validation. This module standardizes the detection process by using a simpler and effective way of generating an XML file for user provided inputs. This approach would help in decreasing the false positive rate, because identifying the vulnerabilities becomes more effective. This module is also a part of the AOP, since the XML file is to be generated for whatever user input that is provided to the web application that connects to an XML database.

For example consider the following query

```java
String xpathExpression = "/Login/username[text()='shan' and passwd/text()='tautology'or'1'='1']";
InputSource inputSource = new InputSource("login.xml");
NodeList nodes = (NodeList) xpath.evaluate
(xpathExpression, inputSource, XPathConstants.NODESET);
```

After intercepting the query, the analyzer obtains the inputs from the query and stores them in an XML document. This document is then further used for validation in order to detect the vulnerabilities.
Figure 5.2 XML file for the tautology query

Figure 5.2 illustrates a sample XML file that would be generated after the XQuery is intercepted. This XML file consists of only the input parameters that were given as user inputs from the client application. Further, this can be used for validation, in order to find if any injection is present.

5.4.3 XQuery Validation

The validation process is to identify the injected parameters with the help of our designed schema and the generated XML file. Though several validation methods are possible, an XML schema is the most powerful and effective technique. The XML schema can be used to define the structure of the XML document and even provide various constraints for it. The proposed schema is a generalized meta data which defines the structure and type of user input, as shown in Figure 5.3. Hence, in this XPath injection preventer
approach, a well defined XML schema is defined for detecting possible injection characters in the input values provided by the user.

![XML schema definitions](image)

**Figure 5.3 XML schema definitions**

The validation process identifies any possible injections in the input values. In case the validation fails, the execution of the intended operation is stopped, and a log file is generated indicating that an injection has occurred. If the validation process is passed, then the operation is allowed to execute and the desired results are obtained. Figure 5.3 depicts the partial schema for the XPath injection preventer approach. The schema is vital in detecting injections in the inputs. Inputs can be of any type, and hence, the schema restricts the values for each data type, thereby providing an effective validation process.

The XML file generated from the XQuery analyzer module is shown in Figure 5.2. It consists of the user inputs now validated with an XPath injection preventer well defined XML schema. The tree structure of the XML file for the XML query is shown in Figure 5.4.
The common method for parsing the XML files is the DOM. Unfortunately this method involves reading the entire file and storing it in a tree structure, which will be inefficient, slow, and a strain on the resources. Hence, an alternative parsing method, called Simple API for XML (SAX) parsing method is used to validate the XML file.
The core part of the validation is shown in Figure 5.5. If the validation passes, then no injection is present in the input parameters; in case of failure, the injection details are entered in a log file for further vulnerability analysis.

Figure 5.5 XML file validation through SAX

Figure 5.6 Log file

Figure 5.6 illustrates a log file generated for a set of attack inputs that were tested. The log file clearly indicates the attack input mismatch with the schema, thereby avoiding injection from taking place.
5.5 RESULTS AND DISCUSSION

To evaluate the XPath Injection preventer system, the designed system has been compared with the existing methods.

The proposed and designed XPath injection preventer system of the WAPS-CIVS,

- prevents tautology based XPath injection in web applications.
- does not demand any change in the business logic.
- designed by means of web services to meet the loosely coupling with any web application.
- has an error log system which will be very helpful for further analysis.
- is different from the existing approaches based on the
  - Schema for validation
  - Interception of inputs
  - Generation input xml file
  - Maintaining log file system
  - Response time

A set of XQuery inputs were submitted to the XPath injection preventer for the period of five weeks. The results are shown in Table 5.1. The average XPath prevention rate is calculated for the XQuery inputs. WAPS-CIVS prevents 93.4 % XPath injections, compared with other approaches and it is lacking in prevention of 6.6% of the XPath injection due to non-inclusion of error customization, other than tautology XQuery.
Table 5.1 Comparison of XPath injection preventer of WAPS-CIVS with the other methods

<table>
<thead>
<tr>
<th></th>
<th>No. of queries</th>
<th>Tautology query</th>
<th>WAPS-CIVS</th>
<th>Static method</th>
<th>Static method with AOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week 1</td>
<td>80</td>
<td>68</td>
<td>63</td>
<td>58</td>
<td>54</td>
</tr>
<tr>
<td>Week 2</td>
<td>58</td>
<td>46</td>
<td>44</td>
<td>37</td>
<td>38</td>
</tr>
<tr>
<td>Week 3</td>
<td>75</td>
<td>62</td>
<td>59</td>
<td>44</td>
<td>49</td>
</tr>
<tr>
<td>Week 4</td>
<td>60</td>
<td>56</td>
<td>55</td>
<td>51</td>
<td>53</td>
</tr>
<tr>
<td>Week 5</td>
<td>65</td>
<td>61</td>
<td>58</td>
<td>52</td>
<td>54</td>
</tr>
</tbody>
</table>

From the Table 5.1, the XPath injection preventer of WAPS-CIVS will prevent all tautology based queries, and it also proves that it is a better method compared to the other two methods as shown in Figure 5.7.

Figure 5.7 Comparison of XPath injection preventer with other the methods
Further, the proposed system is analyzed, and its performance is verified based on the response time, with the XPath Injection Preventer and without it. The response time in a real web environment is collected and tabulated as shown in Table 5.2.

### Table 5.2 Response time assessment of the WAPS –CIVS system

<table>
<thead>
<tr>
<th>Test ID</th>
<th>Response Time Without WAPS-CIVS module (ms)</th>
<th>Response Time With WAPS-CIVS module (ms)</th>
<th>Response Time Difference (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>94.25</td>
<td>125.35</td>
<td>30.85</td>
</tr>
<tr>
<td>2</td>
<td>109.25</td>
<td>155.45</td>
<td>46.25</td>
</tr>
<tr>
<td>3</td>
<td>156.75</td>
<td>195.35</td>
<td>39.40</td>
</tr>
<tr>
<td>4</td>
<td>98.50</td>
<td>115.45</td>
<td>16.95</td>
</tr>
<tr>
<td>5</td>
<td>125.45</td>
<td>175.45</td>
<td>50.0</td>
</tr>
<tr>
<td>6</td>
<td>156.75</td>
<td>200.50</td>
<td>43.75</td>
</tr>
<tr>
<td>7</td>
<td>88.50</td>
<td>140.75</td>
<td>52.25</td>
</tr>
<tr>
<td>8</td>
<td>95.45</td>
<td>157.75</td>
<td>62.3</td>
</tr>
<tr>
<td>9</td>
<td>112.25</td>
<td>156.50</td>
<td>44.25</td>
</tr>
<tr>
<td>10</td>
<td>105.50</td>
<td>135.50</td>
<td>35.0</td>
</tr>
</tbody>
</table>

Different types of XPath queries attempted to hack the XML data with the XPath injection preventer module. Here, the XPath injection preventer module prevents all the attempts and an error log is created in the file system. For all the attempts, the response time difference with and without the XPath injection preventer approach is very minimal in a real time environment as shown in Figure 5.8. The response time overhead with the XPath injection preventer could be compromised, when compared with the consequences of XPath injection. The following graph is a pictorial representation of the response time assessment.
Figure 5.6 shows a comparison of the response time, between the proposed XPath injection preventer tool and without it. As shown in the graph, the XPath injection preventer does not bring about a huge difference in the response time. Since the XPath injection preventer is a modular approach, it can be very widely used in the case of security, logging, etc. When compared to the other approaches the overhead was found to be better.

5.6 CHAPTER SUMMARY

An emerging code injection attack is the XPath injection attack, which takes advantage of the loose typing and forgiving nature of XPath parsers to allow malcontents to piggyback malicious XPath queries on URLs, forms, or other methods to gain access to privileged information and change it. To prevent the XPath Injection, a prevention system called XPath injection preventer has been designed and implemented through web services. In the prevention mechanism, all the user inputs are intercepted from the XQuery, using aspect oriented programming. When the user inputs are placed in the XQuery, the XQuery will be fetched by the XPath injection preventer and will
be converted in the form of an XML file. In the XPath injection preventer, a new XPath injection preventer Schema is defined, to prevent an invalid input which leads to XPath Injection. Since, the XQuery will be converted in the form of an XML file, it will be validated with the defined XPath injection preventer schema for any invalid input. In the validation process, SAX parsing is technique is used to detect the XPath Injection. The performance of the XPath injection preventer is analysed with different XQueries which leads to XPath Injection. The designed XPath injection preventer prevents all the XPath injections in a web application environment, and the response time overhead is very minimal. Since, the XPath injection preventer system has been implemented through a web service, the system can be used independently without disturbing the existing web application.