PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications

V.Shanmughaneethi

Computer Centre, National Institute of Technical Teachers Training and Research (NITTTR), Govt. of India Taramani, Chennai, India

shanneethi@gmail.com

R.Ravichandran

Resource Centre, National Institute of Technical Teachers Training and Research (NITTTR), Govt. of India Taramani, Chennai, India

keelairavi@hotmail.com

S.Swamynathan

Dept. IST, CEG, Anna University Chennai, India

swamyns@annauniv.edu

Abstract

Generally, most Web applications use relational databases to store and retrieve information. But, the growing acceptance of XML technologies for documents it is logical that security should be integrated with XML solutions. In a web application, an improper user inputs is a main cause for a wide variety of attacks. XML Path or XPath language is used for querying information from the nodes of an XML document. XPath Injection is an attack technique, 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. Through the crafted input a malicious user would bypass authentication or to access restricted data from the XML data source.Hence, we proposed an approach to detect XPath injection attack in XML databases at runtime. Our approach intercept XPath expression and parse the XQuery expression to find the inputs to be placed in the expression. The identified inputs are used to design an XML file and it would be validated through a schema.

Keywords:

XPath Injection, SQL Injection, XQuery, Web Application Security, XSLT, XML Schema, XML Security, Command Injection

1. Introduction

Web applications have become one of the most important communication channels between various kinds of service providers and clients. This applications are dynamic extension of a Web server as presentation-oriented and service-oriented. In a presentation oriented web application generates dynamic Web pages containing various types of markup language (HTML, XML, and

DOI : 10.5121/ijwsc.2011.2305

so on) in response to requests. A service-oriented Web application implements the endpoint of a fine-grained Web service. Service-oriented Web applications are often invoked by presentation-oriented applications. This service-oriented applications offering wide range of services, such as on-line stores, e-commerce, and social network services, etc. To meet seamless communication environment, security has always been vitally important in the information world to ensure the integrity of content and transactions, to maintain privacy and confidentiality, and to make sure information is used appropriately. So the the increased importance of Web applications, the negative impact of security flaws in such applications has grown as well. The applications may be designed with the hypothesis that the users in the web applications are legitimate and the inputs provided by them are valid for the application. Moreover, the backend database in the database server is designed to handle the query as a trusted query and act on it. But, the malicious users take the advantage on the assumptions and hypothesis and try to steal information or make vulnerable attacks on the data. So, increased deployment of such web applications there has been an in equal number of attacks targeting those applications. Although application-level firewalls offer immediate assurance of Web application security, but they have drawbacks like requires careful configuration, and they only for a Web application security assessment framework that offers black-boxed testing for identifying Web application vulnerabilities. Among many types of vulnerabilities, command injection vulnerability is quite common and it has become one of the most serious security threats in web applications. A command injection is an exploit of a system weakness to gain access to the system for the purpose of executing malicious code, harvesting user data, and engaging in other activities [1]. In the category of command injection, an XPath Injection attacks may occur when a web site requires user-supplied information to construct an XQuery for XML data. XQuery is a powerful language designed for processing XML data.

2.0 Related Work:

Researchers have started to contribute in the area of XPath injection and its possible liabilities. Amit Kelvin [2] illustrates the nature of XPath injection attacks and its consequences. The paper presents the possible mechanisms of attacking an XPath query with different samples for each type. Jaime Blasco [3] provides a brief introduction to Xpath Injection Techniques and also compares it with another similar attack namely, SQL injection. The paper also portrays various scenarios where possible attacks can completely retrieve the XML document for a given attack query. It also highlights the un-availability of access rights for these XML databases which can be major reason for attack unlike normal RDBMS. Jinghua and Sven [7] describe the satisfiablility test for a XPath query. This defines the structure of the query and the possible optimization of the query for obtaining a desired result set. Dimitris et al, [4] described a novel way for detecting XPath injections. In this paper the location specific identifiers where used to validate the executable XPath code. These identifiers reflect the call sites within the application. The major disadvantage was any source code change required a training mode for re-assigning the identifiers. Nuno Antunes et al, [5] describe the detection of XPath injection in web services using AOP. They initially, instrumented the web service to intercept all XPath commands executed, and then they generate legitimate workload based on the web service operations through that learn XPath queries and then generated an attack load and finally detect vulnerability by comparison of both set. Gabriel et al [6] presents a framework named AProSec, which is a security aspect for detecting SQL injections and Cross Scripting Site (XSS).The authors define clearly the need for AOP for providing security to web applications.

3.0 XPath Injection

XPath is a standard language used to refer to parts of an XML document. It can be used directly by an application to query an XML document. Today, many organizations have adopted XML as a data format for everything from configuration files to remote procedure calls. So, like any other application or technology that allows outside user submission data, XML applications can be

susceptible to code injection attacks, specifically XPath injection attacks. 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. The syntax of XPath bears some resemblance to an SQL query, and indeed, it is possible to form SQL-like queries on an XML document using XPath.

3.1 XPath Injection Consequences

In XPath injection, when a malicious user can insert arbitrary XPath code into the form fields and URL query parameters in order to inject this code directly into the XPath query parser 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. Consider an application that uses XML database to authenticate its users. The application retrieves the user id and password from a request and forms an XPath expression to query the database. An attacker can successfully bypass authentication and login without valid credentials through XPath injection. Improper validation of user-controlled input and use of a non-parameterized XPath expression enable the attacker to injection an XPath expression that causes authentication bypass. For example, consider the xml document

```
<?xml version="1.0"?>
<Login xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
 xsi: noNamespaceSchemaLocation = "authenticate.xsd">
<user>
        <uname>computer</username>
        <passwd>centre</passwd>
</user>
<user>
        <uname>asia</username>
        <passwd>india</passwd>
</user>
<user>
        <uname>tamil</username>
        <passwd>chenai</passwd>
</user>
```

The above XML document stores information about the registered user for the particular web application. To perform authentication for the user, a web application receives username and password from the user.

```
XPathExpression ex = xpath.compile("//Login/user[username/ text() = '"+loginID+"'and
passwd/text()='"+password+"' ]");
```

The user supplied username and password placed into the appropriate place of the XQuery to perform user validation. The following XQuery is generated in server and it would be sent to xml document for user validation.

```
("//Login/user[username/text()='tamil' and passwd/text()='chennai' ]"
```

If the user name and password presented in the XML data then this will return true to the web page otherwise it will return false. This is a simple authentication procedure in web application which uses XML data as back-end service. If the attacker can craft the input, such way that,

always the user becomes an authenticated user for a web site by XPath injection. For example the above XQuery can be crafted as the following Xquery

```
let $str := doc("login.xml")/Login/user
        return if ($str/username='asia' and $str/passwd="or'1'='1' ) then
<b>true</b> else     <b>false</b>
```

Here, the password data is always true in this XQuery. So that, whenever such password is given to the password filed, the user authentication would be treated as a purely authenticated user for the web application. Although this attack grants the attacker access to the application, it does not necessarily grant them access as the most privileged account. In some cases, an attacker can further manipulate the XPath query to force the server to return various parts of the document. Hence, this XPath injection also leads to extracting document structure and modify the document information in addition to escalate privileges.

3.1 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. The common ways to prevent XPath Injections are *Strong input validation, Use of parameterized XPath queries and Use of custom error.* So, the developer has to ensure that the application does accept only legitimate input and another way is use parameterized queries to prevent XPATH injection. Even then, these methods are not consistent to prevent XPath injection in web applications[11]. Hence, we proposed a new approach for effective detection of XPath injection vulnerabilities by schema based validation of the user input that are provided to the web applications.

4.0 Proposed system approach

The proposed system involves a new approach for detecting XPath injection vulnerabilities in web applications shown in figure.1. This approach integrates with Xpath expression scanner, that plays a major role in the detection of XPath injection vulnerabilities by intercepting XQuery which framed by the input parameters.

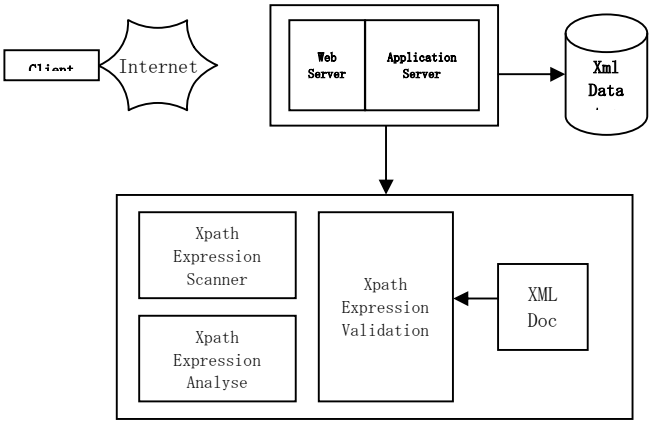

Figure 1: Proposed System Architecture for XPath Detection

The above architecture describes the XPath injection detection technique implemented in a tool named XPath Injection Vulnerability Detector (PXpathV). When the user provides the required inputs into the web forms, they are then placed into the XQuery in an appropriate place of the application. Finally, the complete XQuery string is generated for processing data transaction 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. Sometime, illegitimate inputs may leads to bypass authentication or retrieve privileged XML data. The inputs that lead to XPath injection have to be prevented to run on the XML data. Moreover, the XPath injection is not restricted only by the web form input. It is also possible by HTTP header or by Cookie. But at end, all the inputs would be placed into the XQuery for run on XML data. Hence, our approach analyzes the XQuery for identifying the vulnerabilities and preventing XPath injection.

4.1 Xpath Expression scanner

Query interception involves intercepting the Xpath expression that is generated at run time. This expression that is generated is executed on the XML data store and results are obtained. In order to detect injection vulnerabilities, this dynamically generated expression has to be intercepted. Also intercepting the run time query can be used to detect any type of injection, since the sink points for causing the injections are these queries that are generated at run time. Expression scanner is the best technique in the case of intercepting functions without affecting the business logic. Using expression scanner, these run time generated queries can be intercepted before they are executed in the database server.

4.2 Xpath Expression Analyze module

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 query and retrieves the input parameter. Different types of queries can also be tokenized in order user inputs. After obtaining the input parameters the detection of possible vulnerability should be done. 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 a XML (eXtensible Mark-up Language) file which would be validated by our proposed schema. This file is a well-formed document, platform indepented and provides lesser over head for validation. This module standardizes the detection process by using a simpler and effective way of generating a XML file for user provided inputs. This approach would help in decreasing the false positive rate because the identifying the vulnerabilities becomes more effective. This module is also a part of the AOP layer since this XML file is to be generated for whatever user input that is provided to the web service that connects to a XML database. For example consider the following query

("//Login/user[username/text()='asia' and passwd/text()='any' or '1'='1']")

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

```
<? xml version="1.0" encoding= "utf-a"?>
<xper>
<in>asia</in>
<in> any' or '1' ='1</in>
```

Figure 2: Sample XML file for the above expression

Figure 2 illustrates a sample XML file that would be generated after the Xquery expression 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.

4.3 XQuery Validation

The validation process is to identify the injected parameters with the help of our schema[13] and the generated XML file. Though several validation methods are possible, a 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 schema is a generalized meta data which define structure and type of user input . Hence in our approach, a well defined XML schema is defined for detecting possible injection characters in the input values provided by the user.

The validation process identifies any possible injections in the input values. In case if 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. The schema is vital in detecting injections in the inputs. Inputs can be of any type and hence the schema restricts values for each data type thereby providing an effective validation process.

The XML file generated from the previous module that consists of the user inputs is now validated with a well defined XML schema. If the validation passes then injection is not present in the input parameters, in case of failure the injection is logged in a log file. The log file clearly indicates the attack input mismatch with the schema thereby avoiding the injection to take place.

5.0 Results & Discussions

To evaluate our proposed approach, we have analyzed the performance of the developed tool based on the response time with our PXpathV based approach as well as without our approach. The response time in real web environment is collected and tabulated is shown in table1.

TABLE -1 RESPONSE TIME ASSESSMENT

No. of Test	Response Time Without PXpathV module (ms)	Response Time With PXpathV module (ms)	Response Time Difference (ms)
1	96.4	127.65	31.25
2	104.3	149.85	45.55
3	152.45	193.8	41.35
4	97.5	115.8	18.3
5	128.7	179.9	51.2
6	158.3	204.05	45.75
7	87.4	138.95	51.55
8	97.2	161.7	64.5
9	113.7	160.1	46.4
10	102.8	139.9	37.1

The response time difference with our proposed approach is very minimal compared to the consequences of Xpath injection. The time delay could be compromised when to compare the threat. The following graph represents pictorial representation of response time assessment.
The following graph in figure 3 shows the comparison of the response time, between the proposed PXpathV tool and without it. As shown in the graph, the PXpathV does not bring a huge difference in the response time. In the following graph *X – Axis represents Number of Test and Y – Axis represents Response Time*

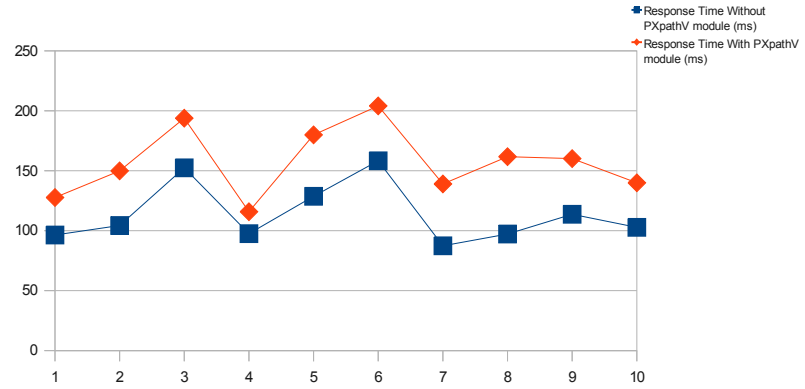

Figure 3: Sample log file generated for different attack inputs

Since the PXpathV is a modular approach, it can be very widely used in the case of security, logging, etc. When compared to other approach the over head was found to better.

6.0 Conclusion and Future Work

In this paper we have proposed a new approach for detecting XPath injection vulnerabilities. This based approach is a very effective method for detecting vulnerabilities. Comparing with previous approaches, there can be a significant reduction in overhead and also less number of false positives can be achieved. The main advantage of this approach is that our scanning module provides modularity as well as avoids the need for changes in the source code of the application and it wont affect the business login of the application. We have analyzed the approach using our tool PXpathV, with different web applications and found that the response time is limited. In future, we intend to analyze other forms of web attacks like SQL injection, Cross-site scripting and LDAP injection.

References

[1]. Zhendong Su, Gary Wassermann, “The Essence of Command Injection Attacks in Web Applications”, Proceedings of the thirty third ACM symposium on Principles of Programming Languages,South Carolina,2006,pp. 372-382.
[2].Amit Kelvin, “Blind XPath Injection”, a whitepaper from Watchfire, Director of Security and Research, Sanctum, 2005.
[3]. Jaime Blasco, “Introduction to XPath Injection Techniques”, Hakin9, Conference on IT Underground, Czech Republic, 2007, pp.no 23-31.

[4]. Dimitris Mitropoulos, Vassilios Karakoidas, and Diomidis Spinellis, “Fortifying Applications against XPath Injection Attacks”, MCIS 2009: 4th Mediterranean Conference on Information Systems, 2009, Athens, pp.no 1169–1179.

[5]. Nuno Antunes, Nuno Laranjeiro, Marco Vieira, Henrique Madeira, “ Effective Detection

of SQL/XPath Injection Vulnerabilities in Web Services", IEEE International Conference on Services Computing,Portugal, 2009, pp. 260-267.

[6].Gabriel Hermosillo, Roberto Gomez, Lionel Seinturier, Laurence Duchien , "Using Aspect Programming to Secure Web Applications", Journal of Software,Vol. 6, No. 2,Vienna,2008,pp. 53-63.

[7].Jinghua Groppe,Sven Groppe, " Filtering unsatisfiable XPath queries", Journal Data & Knowledge Engineering , Vol.64 No. 1,Amsterdam, 2008,pp.no 134-169.

[8] XML Path Language (XPath) version 2.0, http://www.w3.org/TR/XPath

[9].OWASP Guide, http://www.owasp.org/index.php/Blind_XPath_Injection

[10] Vieira, M., Antunes, N., Madeira, H., "Using Web Security Scanners to Detect Vulnerabilities in Web Services", Intl.Conf. on Dependable Systems and Networks, Lisbon, 2009.

[11] Laranjeiro, N., Vieira, M., Madeira, H., "Protecting Database Centric Web Services against SQL/XPath Injection Attacks", DEXA 2009, Linz, Austria, September 2009.

[12] Wu, R., Hisada, H. and Ranaweera, R., ''Static analysis of web security in generic syntax format'', The 2009 International Conference on Internet Computing (ICOMP 2009), Las Vegas, NV, pp. 58-63.

[13] Velu Shanmughaneethi, Ra. Yagna Pravin and S. Swamynathan "XIVD: Runtime Detection of XPath Injection Vulnerabilities in XML Databases through Aspect Oriented Programming" Communications in Computer and Information Science, 1, Volume 198, Advances in Computing and Information Technology (ACITY 2011), Part 1, Pages 192-201