Defending Web Application Against Xpath Injection Attack

Prof. Krupa Trambadiya
Assistant Professor, Dept of IT
Vadodara Institute of Engineering, Kotambi

Abstract*- Nowadays XML is used as communication protocol in web applications. The expressiveness and flexibility of XML gives opportunities for attackers to perform injection attacks. By using XML databases instead of relational databases the web applications are more vulnerable to injection attacks. In this scheme, using dynamic analysis, XPath Injection attack detection system is developed intended to scale up the performance with reasonable response time and high detection coverage with low false positives. Followed by this, Attack Dataset Generation Phase generates number of attacks. Attack set is expected to cover number of possible attack strings in order to be useful in XML and XPath injection Attack Detection Schemes, Researchers and Vulnerability Scanners for testing purpose which is again used as the training set for decision tree learning. Here decision tree learning is used to detect new unknown attacks using generated attack set of known attacks.*

Keywords- XPath, Injection Attack, Attack Detection, Pattern Matching, Regular Expression, Decision Tree Learning.

I. INTRODUCTION

There is no doubt that web application security is current and very news - worthy subject [1]. People throughout the globe prefer to do their business, banking, shopping etc. online. And to perform some analysis on the data, they access a variety of information and transactions through Web sites and Web applications. Web application is developed in a programming language. It is a software application that runs in a web browser. The web applications use a backend database which stores data. In a service-based environment, providers offer a set of services that relies on frequent access to backend database for service consumer to discover and exploit [2]. Most of the Web applications use relational databases to store and retrieve information. Instead of relational database web applications are increasingly depending on XML database. It understands different programming languages and is supported by standard protocols such as Simple Object Access Protocol (SOAP) for message exchange, Web Services Description Language (WSDL) for interface description, and Universal Description, Discovery, and Integration (UDDI) [2]. With the growing acceptance of XML technologies for documents, it is reasonable that security should be integrated with XML solutions [3]. The flexibility and expressiveness of XML makes it prone to injection attacks. Unlike SQL database, no access control mechanism is integrated there with XML database. This is also one of the reasons for the XML to be vulnerable against attacks and security threats. The main focus of the work is on XML based vulnerability detection. There are mainly two approaches of vulnerability detection, static and dynamic. Static analysis is done by developers by analyzing the source code which is not realistic but the systematic way to test the system like white box testing. While dynamic analysis does not need to have source code handed. It gives more realistic view like black box testing. In this scheme, the dynamic approach is used for injection attack detection. Here pattern matching approach is used using regular expressions of attack string. The performance is expected to have high detection rate with low false positives and reasonable response time.

If malicious value are inserted via application form (for example, registration form), it can harm database in many ways like, executing malicious scripts in the application, making database invalid by exploiting XML meta-characters, escalation of privileges by overwriting the XML tags, unauthorized access of data and many more. The major focus is on developing attack database for detecting number of XML based Injection attacks. Using the attack database it can be determined whether the XML query entered by the user of the web application is valid query or an injection attack.

According to OWASP - Top Ten list, Injection attack is most dangerous one [4]. The main focus of web application developers is on fulfilling client requirements and to meet time constraints. And thus they develop application in short time and enough attention is not given to security constraints resulting in the possibilities for existence of vulnerabilities in the developed application [5] [6]. According to survey reports many organizations suffer from financial losses due to the exploitation of vulnerabilities in web applications. Most of the vulnerabilities are due to improper input validations or insecure programming practices [7].

Analysis of web application is necessary in order to mitigate security vulnerabilities. There are mainly two types of analysis. The first type is Static Analysis which analyses the source code of the web application for checking presence of vulnerabilities. The second type is Dynamic Analysis which analyses the web application during execution time. The source code of the web application is not available for dynamic analysis technique. The high rate of false positives is a problem with the static analysis methods [4]. Static analysis tools detect certain patterns that usually indicate vulnerabilities, but many times they detect vulnerabilities that do not exist, due to basic limitations of the static profile of the code. In this scheme dynamic analysis of web application is used due to its advantages it offers and to overcome the shortcoming like performance of previously proposed schemes. The major advantage is the low rate of false positives when compared to static analysis. The second advantage is that it checks a web application in the same way an attacker would attack a web application. Thus by using dynamic analysis, security vulnerabilities can be recognized and fixed before it is exploited by malicious users.

Designing of dynamic XPath injection attack detection scheme using regular expressions to detect the queries passes from web application to database server for XPath injection vulnerabilities in web application. Generating attack set which is in order to be useful to researchers or penetration testing tool developers to test their system. Using the same attackset as the training set, the decision tree learner is developed to detect new or unknown attacks like zero day attacks. Improvement in performance maintaining high detection coverage and low false positives with reasonable response time is the main objective.

II. LITERATURE SURVEY

Many times developers pay their attention on implementing functionalities and some constraints like time to market and ignore security aspects. And thus, the security of web applications is very poor. And web applications are so widely exposed that any security vulnerability will most probably be uncovered and exploited by hackers [5] [8].

Vulnerability detection is identifying loop holes or semantic gaps in the application in various phases like development or debugging where vulnerability is nothing but a loop hole in the application or semantic gap between web application front end and back end (database). There are two types of vulnerability detection - Static analysis and Dynamic analysis. In static code analysis, source code is analyzed in order to find vulnerabilities which can be exploited. Penetration testing is dynamic analysis which analyses the response of web service at run time which is more realistic or real time [HYPERLINK \l "Nun092" 5]6]**}**. Signature based and knowledge based approach are widely used for vulnerability detection. Signature is a payload that identifies an attack through malicious context. It gives less false positives but does not detect new unknown attacks having small variations from a known payload. Another way is knowledge based detection, which detects attacks based on previously known attack behavior. They learn from known attacks and can detect new unknown attacks. Strategy based detection which combines both signature based and knowledge based gives less false positives and more coverage for unknown attacks [10].

A. RELATED WORK

Nuno Antunes and his colleagues [5] proposed a scheme for dynamic analysis. Hash value is calculated for each valid command and compared with hash values of incoming commands. If match does not found, vulnerability is recognized and logged for future reference. There are two performance metrics considered - Detection coverage and False positive rate. Results show high detection coverage and low false positives. Results obtained are only based on java programming language. Not others like c++, c#.

T.M.Rosa et al. [10] developed a strategy based scheme – a hybrid approach in which knowledge based is derived from a signature based approach. Known attacks were detected by ontology developed using known attack classes and their instances and new or previously unknown attacks were detected by reasoner which inferred based on the behavior of known attacks.

Nuno Laranjeiro et al. [6] developed an approach for anomaly detection to find deviation from historical learned valid commands. Original workload is re-executed to verify the working of the scheme and to make sure there is no vulnerability remains. Performance of workload generation is good in all cases. Learning process is fast and effective consuming few seconds. The final stage which is the security mechanism consumes hundredth of millisecond.

D. Mitropoulos et al. [12] have developed a proxy library to implement security features of XPath. In this approach a valid unique location identifier is generated from each valid XPath query and is compared against the incoming query's identifiers. The prototype counters many forms of XPath injection attacks which exploit the vulnerabilities like incorrectly passed parameters, incorrect type handling and incorrectly filtered quotation marks.

V. Shanmughaneethi et al.[3] have developed XPath injection prevention technique for user inputs. To analyze the performance of the scheme, Response time of the query execution in the presence of the system was compared against response time of the query execution without this scheme. The difference was very minimal i.e. in few milliseconds.

III. PROPOSED WORK

The proposed work is designed to defend against XML injection attack techniques. Web applications which use XML documents in backend are vulnerable to such injection attacks through user supplied data from backend. In simple terms, this approach is used between backend and web application which uses XML documents as database instead of / along with relational database system to fill semantic gap between them. The attack detection is done through the Error Pattern Matching approach. Regular Expressions will be used for pattern matching. Regular Expressions for attacks of each type are constructed and user supplied input values are checked against the pattern of regular expression if match found, the input string is attack and in this case the execution of query will be stopped otherwise it is legitimate input and generated query will be executed on the database.

Construction of Regular Expressions for each XML based injection attack type like Tag Injection, Tautology Injection, Meta-character Injection, Comment Injection, CDATA Injection attack, Alternate Encoding Injection, External Entity Injection, Piggybacked Injection and Blind XPath injection attack. Construct the code for Error Pattern Matching using these Regular Expressions for all above attacks. Construct Attack Dataset from Regular Expressions. Taking generated attack set as training set generate Decision tree learner to learn new unknown attacks like zero day attack.

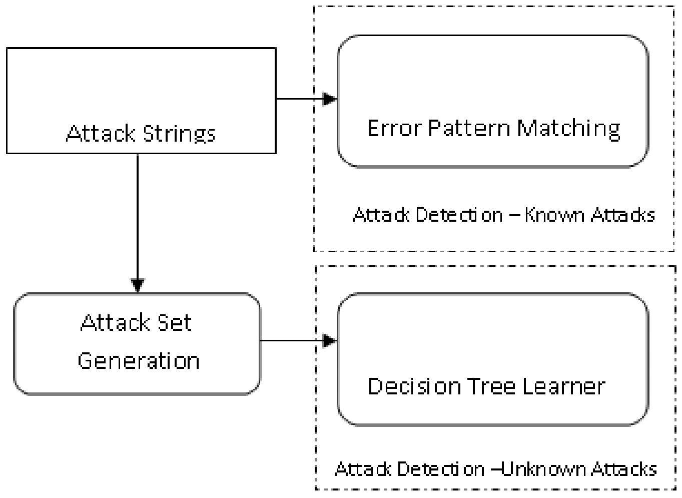


Figure 1.1: Flow of Proposed Scheme

A. WORK FLOW

Construction of Regular Expressions for each type of attack and Implement Error Pattern Matching approach using Java is done here. Matcher class and Pattern of REGEX Packages are used for the same. DFA is constructed from Regular Expressions and Traverse it to get possible strings from Current Regular Expression. Output them to a file of Attack set. A decision tree learner has been developed using generated attack set as training set in Weka Explorer.Decision tree algorithm C4.5 (improved ID3) is implemented by J48.

B. REGULAR EXPRESSIONS

A Regular Expression defines a search pattern for strings. The abbreviation for regular expression is regex. The search pattern can be anything from a simple character, a fixed string or a complex expression containing special characters which describes the pattern. The pattern defined by the regex may match one or several times or not at all for a given string. Regular expressions can be used to search, edit and manipulate text. At first, regular expressions were written for each type of attack. The regular expression based solution was written in such a way that it covers all possible cases of attack that can happen in web application.

a) Regular Expression for Tautology Injection Attack

In tautology injection attack the main attack string used for attack contains the logical OR and AND symbol. The equal operator (=) is also used mostly in this type of attack which is used to construct tautology expressions like 1=1, ‘a’=’a’ etc. So the following regular expression can be used for detecting Tautology Injection Attacks.
(\s*\d+\s*\=\s*\d+\s*)|(\s*\w*('|")\s*\=\s*('|")\w*('|")\s*)|(\s*\w*('|")\s*)|(\s*('|")\w*('|")\s*\=\s*('|")\w*\s*)(\s*(OR|or|AND|and)\s*)*

b) Regular Expression for Comment Injection Attack

In comment injection attack the comment symbol <!-- > is used to comment out the remaining part of an xml document. So the regular expression for detecting comment injection attack should be like below.
(\w*\s*)*('*\s*\w*"*)*(\s*\w*)*\<\!\-\-\w*(\-\-)*(\>)*\w*

c) Regular Expression for Tag Injection Attack

In tag injection attack tag names and values are inserted in user input fields to manipulate existing values in xml document and overwriting existing tags and their values.

So the following regular expression can be used to detect tag injection attack.

d) Regular Expression for Meta-Character Injection Attack

Meta-characters are used by attackers in order to perform meta-character injection. Meta-characters like <, &, ‘, “ etc.
are mostly used for performing meta-character injection. The regular expression for meta-character injection detection is
\s*[a-zA-Z0-9]*\s*('|"|&|<|>|.)|(('|"|&|<|>|.)\s*[a-zA-Z0-9]*\s*)

e) Regular Expression for Meta-Character Injection Attack

Meta-characters are used by attackers in order to perform meta-character injection. Meta-characters like <, &, ‘, “ etc. are mostly used for performing meta-character injection. The regular expression for meta-character injection detection is
\s*[a-zA-Z0-9]*\s*('|"|&|<|>|.)|(('|"|&|<|>|.)\s*[a-zA-Z0-9]*\s*)

f) Regular Expression for CDATA Injection Attack

CDATA Injection can be detected with the help of following regular expression
(\s*\w*\s*\<!\[CDATA\[\s*('|"|<|>)*\w*('|"|<|>)*\s*\]\]\>)+

g) Regular Expression of Alternate encoding Injection Attack

(\w*\s*\'*\"*)*\;*exec(\w*)\-*(\w*\s*)*

h) XML External Entity Attack

\w*\s*(!ENTITY|&)\W*\S*

i) Piggybacked Injection Attack

(\w*\s*\'*\"*)*\;(\w*\s*\'*\"*)*

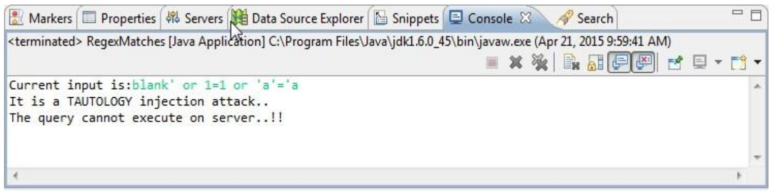
j) Blind XPath Injection Attack

ischildnode(\w*)|count(\w*)|strlen(\w*)|substring(\w*)

IV. ATTACK DETECTION

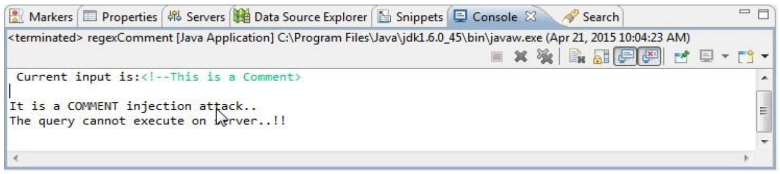
Regular Expressions were written for detecting XPath Injection Attacks like Tautology Injection, Meta-character Injection, Tag Injection, Comment Injection and CDATA Injection, Alternate Encoding, External Entity Injection, Piggybacked Injection and Blind XPath Injection attacks. The Java code is written for Error Pattern Matching Approach using Regular expressions for web applications and detects attacks that can attack Web Application.

A. DETECTION OF INJECTION ATTACKS



```
<terminated> RegexMatches [Java Application] C:\Program Files\Java\jdk1.6.0_45\bin\javaw.exe (Apr 21, 2015 9:59:41 AM)
Current input is:blank' or 1=1 or 'a'='a
It is a TAUTOLOGY injection attack..
The query cannot execute on server..!!
```

Fig 4.1: Detection of Tautology Injection for attack string



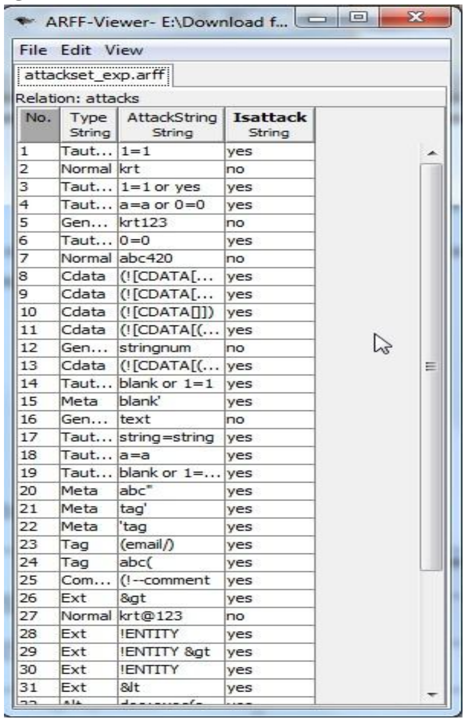
```
<terminated> regexComment [Java Application] C:\Program Files\Java\jdk1.6.0_45\bin\javaw.exe (Apr 21, 2015 10:04:23 AM)
 Current input is:<!--This is a Comment>

It is a COMMENT injection attack..
The query cannot execute on server..!!
```

Fig 4.2: Detection of Comment Injection for malicious Input

B. ATTACK SET GENERATION AND DECISION TREE LEARNING FOR NEW UNKNOWN OR ZERO - DAY ATTACKS

As any injection attack is detected, it is automatically added to attack dataset called attack Set here. Attack dataset is the set of attack strings which represents all the XPath Injection attacks implemented in the scheme. The generated attack Set is in the form of Notepad file. Attack Set is shown in below figure.

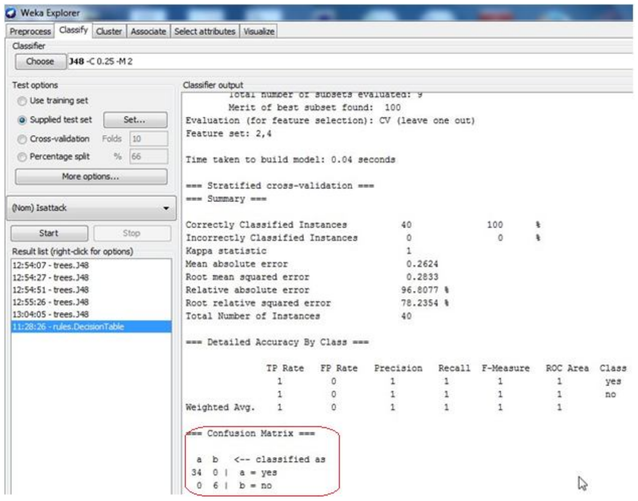

ARFF-Viewer- E:\Download f...
File Edit View
attackset_exp.arff

Relation: attacks

No.	Type String	AttackString String	Isattack String
1	Taut...	1=1	yes
2	Normal	krt	no
3	Taut...	1=1 or yes	yes
4	Taut...	a=a or 0=0	yes
5	Gen...	krt123	no
6	Taut...	0=0	yes
7	Normal	abc420	no
8	Cdata	(![CDATA[...	yes
9	Cdata	(![CDATA[...	yes
10	Cdata	(![CDATA[]])	yes
11	Cdata	(![CDATA[(...	yes
12	Gen...	stringnum	no
13	Cdata	(![CDATA[(...	yes
14	Taut...	blank or 1=1	yes
15	Meta	blank'	yes
16	Gen...	text	no
17	Taut...	string=string	yes
18	Taut...	a=a	yes
19	Taut...	blank or 1=...	yes
20	Meta	abc"	yes
21	Meta	tag'	yes
22	Meta	'tag	yes
23	Tag	(email/)	yes
24	Tag	abc(	yes
25	Com...	(!--comment	yes
26	Ext	&gt	yes
27	Normal	krt@123	no
28	Ext	!ENTITY	yes
29	Ext	!ENTITY &gt	yes
30	Ext	!ENTITY	yes
31	Ext	&lt	yes

Figure 4.3: Generated .ARFF file of Attack Set

The generated ARFF file acts as a training set for learning process. Decision tree analysis is applied on this file in order to learn unknown attacks from set of known attacks.



J48 -C 0.25 -M 2

Supplied test set

(Nom) Isattack

Result list (right-click for options)
12:54:07 - trees.J48
12:54:27 - trees.J48
12:54:51 - trees.J48
12:55:26 - trees.J48
13:04:05 - trees.J48
11:28:26 - rules.DecisionTable

```
        Total number of subsets evaluated: 9
        Merit of best subset found:  100
Evaluation (for feature selection): CV (leave one out)
Feature set: 2,4

Time taken to build model: 0.04 seconds

=== Stratified cross-validation ===
=== Summary ===

Correctly Classified Instances          40              100      %
Incorrectly Classified Instances         0                0      %
Kappa statistic                          1
Mean absolute error                      0.2624
Root mean squared error                  0.2833
Relative absolute error                 96.8077 %
Root relative squared error             78.2354 %
Total Number of Instances               40

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 1         0          1          1         1          1        yes
                 1         0          1          1         1          1        no
Weighted Avg.    1         0          1          1         1          1

=== Confusion Matrix ===

  a  b   <-- classified as
 34  0 |  a = yes
  0  6 |  b = no
```

Figure 4.4: Generated Confusion Matrix

a) Analyzing Confusion Matrix

The row indicates the true class and the column indicates the classifier output. So, each entry indicates the number of instances of row classified as column. In this example, 0 "Nos" were incorrectly classified as "Yes", 6 "Nos" were correctly classified as "No", etc. So there is zero false positive rate yielded. As a result, all correct classifications are on the top-left to bottom-right diagonal. Other values than that diagonal is an incorrect classification which may yield false positives.

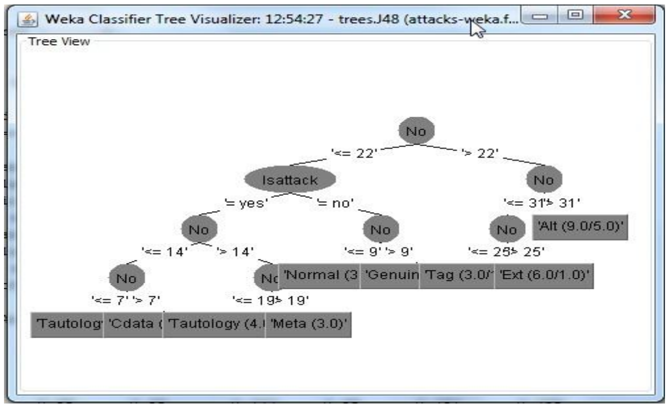


Figure 4.5: Visualizing Decision Tree

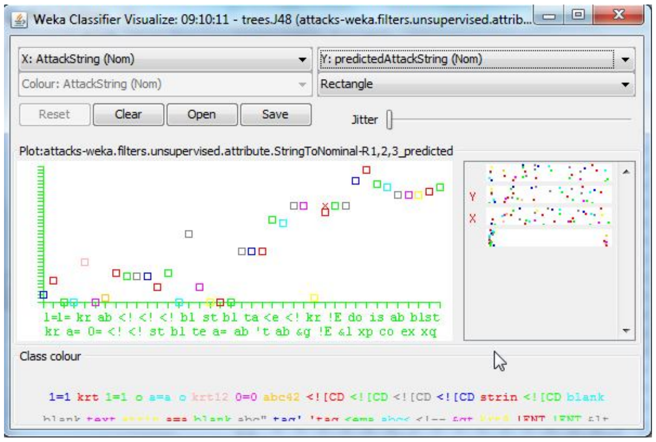


Figure 4.6: Visualizing Classifier errors

C. EXPERIMENTAL RESULTS OF ERROR PATTERN MATCHING APPROACH

Table 4.1 Experimental Results classified by attack type

Type of Attack	Detection Coverage (%)	False Positive Rate (%)
Tautology Injection	97	3
Comment Injection	100	0
Tag Injection	100	2
Meta-character Injection	100	0
CDATA Injection	100	0

V. CONCLUSION

The study of the related work done for the XPath injection detection reveals that there is a need to fulfill all the desired performance metrics like high detection rate, lower false positives and reasonable response time. The proposed scheme for XPath Injection detection is intended to meet all these important criteria by building a system which detects number of XPath injection attacks using Error Pattern Matching approach. Followed by this, XPath attack set is generated which can be useful as the attack set to researchers, Detection scheme developers or vulnerability scanners in order to test their system to check functionality. Using this attack set as training set the decision tree learner is developed to learn from known attack set and detect new unknown attacks like zero day attacks and enhance detection coverage and false positive rates.

REFERENCES

[1] Marcus Pinto Dafydd Stuttard, The Web Application Hacker's Handbook: Discovering and Exploiting Security Flaws.: Wiley Publishing, Inc., 2007.

[2] Lilly Suriani Affendey, Nur Izura Udzir, Ramlan Mahmod Aziah Asmawi, "XIPS: A Model-based Prevention Mechanism for Preventing Blind XPath Injection in Database-Centric Web Services Environment," International Journal of Advancements in Computing Technology, vol. 5, no. 10, pp. 69-77, June 2013.

[3] R.Ravichandran, S.Swamynathan V.Shanmughaneethi, "PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications," International Journal on Web Service Computing, vol. 2, no. 3, pp. 57-64, September 2011.

[4] (2014, August) Open Web Application Security Project. [Online]. https://www.owasp.org/index.php/Top_10_2017-Top_10

[5] Nuno Laranjeiro, Marco Vieira, Henrique Madeira Nuno Antunes, "Effective Detection of SQL/XPath Injection Vulnerabilities in Web Services," in International Conference on Services Computing, University of Coimbra – Portugal, 2009.

[6] Marco Vieira and Henrique Madeira Nuno Laranjeiro, "Protecting Database Centric Web Services against SQL/XPath Injection Attacks," in Database and Expert Systems Applications. Coimbra, Portugal: Springer, 2009.

[7] M. Zulkernine H. Shahriar, "Mitigating and Monitoring Program Security Vulnerabilities," School of Computing, Queen‟s University, Kingston, Canada, Technical Report 2010-572, 2010.

[8] Marco Vieira, Henrique Madeira Nuno Laranjeiro, "A Learning-Based Approach to Secure Web Services from SQL/XPath Injection Attacks," in Dependable Computing (PRDC), Tokyo, December 2010.

[9] Amit Klein, "Blind XPath Injection," White Paper, 2005.

[10] Altair Olivo Santin and Andreia Malucelli Thiago Mattos Rosa, "Mitigating XML Injection 0-Day Attacks through Strategy-Based Detection Systems," Security & Privacy, vol. 11, no. 4, pp. 46-53, August 2013.

[11] Nils Gruschka, Ralph Herkenhoner Meiko Jensen, "A Survey of Attacks on Web Services," Computer Science-Research and Development, vol. 24, no. 4, pp. 185-197, May 2009.

[12] Vassilios Karakoidas and Diomidis Spinellis Dimitris Mitropoulos, "Fortifying Applications Against Xpath Injection Attacks," in MCIS 2009 Proceedings, Athens, 2009.

[13] OWASP Testing Guide V 4.

[14] http://docs.oracle.com/javase/7/docs/api/java/util/regex/package-summary.html

[15] http://stackoverflow.com