Two Level Security Approaches System Architecture for Secure XML Database Centric Web Services against XpathInjections

[1]ziahAsmawi, [2]Lilly SurianiAffendey, [3]NurIzuraUdzir, [4]RamlanMahmod

[1]Computer Science, UniversitiPutra,Malaysia, Serdang, Malaysia
[2]Computer Science, UniversitiPutra, Malaysia, Serdang, Malaysia
[3]Computer Science, UniversitiPutra, Malaysia, Serdang, Malaysia
[4]Computer Science, UniversitiPutra, Malaysia, Serdang, Malaysia

[1]a_aziah@upm.edu.my, [2] lilly@upm.edu.my, [3]izura@upm.edu.my
[4]ramlan@upm.edu.my

https://doi.org/10.26782/jmcms.2019.03.00075

Abstract

Web services are deployed using eXtensible Markup Language (XML), which is an independent language for easy transportation and storage. As an important transportation for data, Web services has become increasingly vulnerable to malicious attacks that could affect essential properties of information systems such as confidentiality, integrity, or availability. Like any other application that allows outside user submission data, Web services can be susceptible to code injection attacks, specifically XPath (XML Path Language) injection attacks. This kind of attack can cause serious damage to the database at the backend of Web services as well as the data within it. To cope with this attack, it is necessary to develop effective and efficient secure mechanism from various angles, outsider and insider. This paper addresses both outsider and insider threats with respect to XPath injections in providing secure mechanism for XML database-centric Web services. We propose the two level security approaches for the ultimate solution within XML database-centric Web services. The first approach focuses on preventing malicious XPath input within Web services application. In order to address issues of XPath injections, we propose a model-based validation (XIPS) for XPath injection attack prevention in Web service applications. The second approach focuses on preventing insider threat within XML database. In order to deal with insider threat, we propose a severity-aware trust-based access control model (XTrust) for malicious XPath code in XML database.

Keywords : Web Services, Database Security, Blind Xpath Injection, Model-Based, Hotspot

I. Introduction

The growing acceptance of XML technologies for documents and protocols, it is logical that security should be integrated with XML solutions. In a Web services

application, an improper user input is root 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 used to exploit applications that construct XPath (XML Path Language) queries from user-supplied input to query or navigate XML document.

In this section, we provide the two level security approaches as the solution for XML database centric Web services. In this solution, we have XIPS and XTrust as the complete solution to protect against XPath injection attacks. The first approach is XPath Injection Prevention System (XIPS), a model-based validation against malicious XPath input within Web service applications while the second approach is XTrust, a severity-aware trust-based access control against malicious XPath code within XML databases stored procedure.

II. Two Level Security Approaches

This section describes details of proposed solution for XML database centric Web service against XPath injections. In this propose solution, we provide two level security approaches which are XIPS and XTrust. XIPS provides protection from external threat against malicious XPath input in Web services while XTrust provides protection from internal threats against malicious XPath code in XML database. This section provides system architecture for overall security protection for XML database centric Web services environment. In this research, we have two level security approaches which consist of XIPS and XTrust for securing the databases of Web services applications as shown in Figure 1.

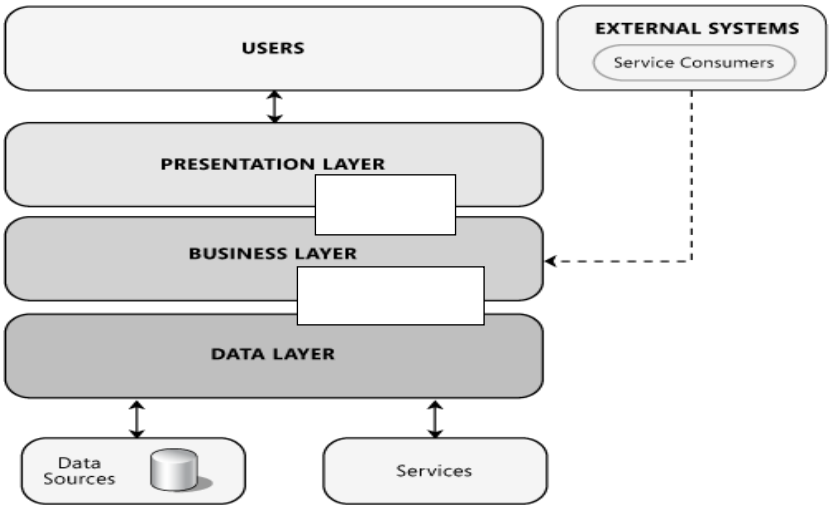

Fig. 1. System Architecture

The system flow includes all steps below as depicted in Figure 2:

i) User submits input through Web services application.
ii) XIPS will check the user input using static and runtime validation.
iii) If the input is an XPath injection attack, the query will be denied.

iv) If the input is valid, the query is passed to the XTrust.

v) Then, XTrust will evaluate the code by checking user queries; capturing bad transactions and errors.

vi) XTrust record the bad transaction ID, error ID and severity ID in XLog.

vii) XTrust calculates the TV.

viii) Making the access decision for XML databases by comparing users' TV and data's TV. The users' TV is updated automatically in the Users' Access Permission Policy file. The XML Database's Access Permission Policy file describes the data's TV required to access data which have its own trust value. The Access Manager matches two files to manage a user's right to access requested data. The Access Decision Maker allows or denies users access to XML databases according to the results.

ix) If the user's TV is equal or larger than the data's TV, the access decision allows the user to access the data. Otherwise access is denied.

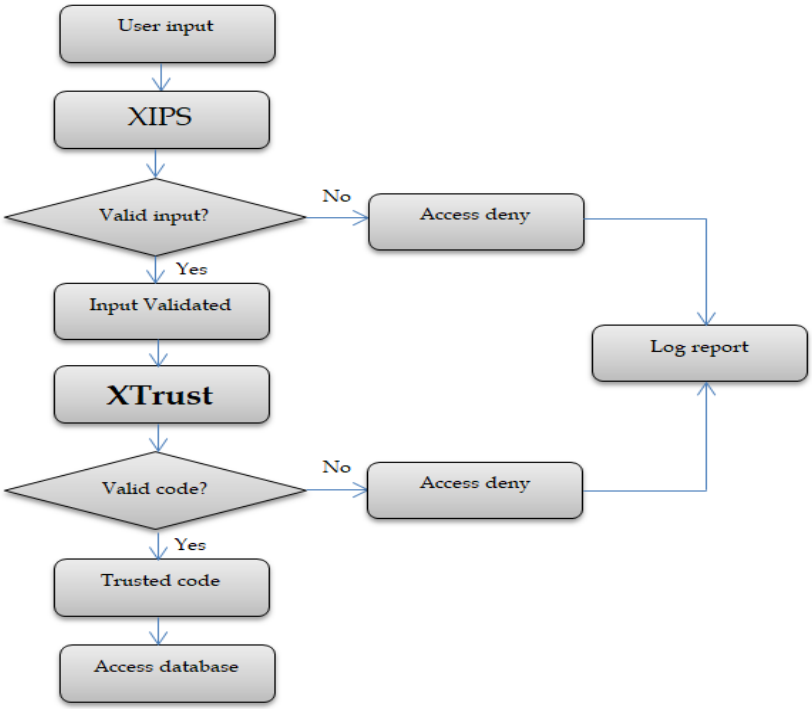

Fig. 2. System flow

III. Related Works

Researchers have started to contribute in the area of XPath injection and its possible liabilities. A detailed study conducted by (Klein, 2004) illustrates the nature of XPath injection attacks and their consequences. The author presented the possible mechanisms of attacking an XPath query with different samples for each type. The Blind injection technique retrieves the structure of the XML database, which allows

different types of attacks. The author did not provide any practical solution to prevent the XPath injection.

A solution for XPath injection was proposed by Blasco (2007). The author provided a brief introduction to XPath injection techniques and also compared it with another similar attack, namely, SQL injection. The author also portrays various scenarios, where possible attacks can completely retrieve the XML document for a given sample attack query. It also highlights the unavailability of access rights for these XML databases, which can be the major reason for the attack, unlike the traditional relational databases.

Jinghua and Sven (2008) described the satisfiability test for XPath query. This defines the structure of the query and the possible optimization of the query for obtaining the desired result set. In their approach, the authors analysed the XQuery for the possibility of the XPath injection. Only in 2009, a method to detect an XPath injection is proposed. Mitropoulos (2009) uses the location specific identifiers to validate the executable XPath query by reflecting the call sites within the application. The major drawback of this proposed technique is any change in the source code will require a new training to reassign the identifiers.

Another work in detecting the XPath injection is by using Aspect Oriented Programming (AOP) whereby the Web services are instrumented to intercept all XPath commands executed (Antunes et al., 2009). This technique will generate a white list consisting of legitimate workload based on the Web services operation through learning of the XPath queries. After the attack workload has been generated, the technique will be able to detect the XPath injection by comparing both lists. Solutions proposed for preventing XPath falls in false positives, and are more time consuming in detecting the XPath injection.

Moreover, Shanmughaneethi et al. (2011) had proposed a schema-based validation for the detection of XPath injection vulnerabilities in XML database. In this method, an XPath query is converted into XML document. This XML document will be checked with already defined XML schema for validation. If the XML document passes, XPath query has no injection otherwise will be considered as XPath injection and the corresponding user not be allowed to continue the process. However, maintaining schemas is difficult and need most efforts at the beginning.

More recently, Karumanchi and Aquicciarini (2015) introduce a client-based classification of Web service vulnerabilities, as well as a proxy-based solution for efficient vulnerability detection. The implementation of proxy-based solution requires that all protected websites revise their authentication software which is costly and time consuming. Furthermore, Thome et al. (2015) propose an approach to complement existing vulnerability detection by forming sound and precise slices thus identifying false and true positives.

IV. Conclusions and Future Work

This paper discusses the security issues of Web services and XML databases in general, as well as the attacks that occur within them, and the proposed solutions for solving those problems. The goal of this research was to provide better protection for XML database-centric Web services against both outsider and insider threats with respect to XPath injections. In the process of realizing the idea, several achievements have been made which contribute towards the body of knowledge in database security generally, and XML database-centric Web services specifically.

We have contributed in providing a thorough investigation on existing works proposed by other researchers to provide protection to secure database-centric Web service. The investigation was carried out to give an idea on what has been done and what still needs to be done to secure database-centric Web services. As the result, we have produced solutions for malicious XPath input and malicious XPath code through the implementation of two level security approaches solution which consists XIPS and XTrust. The first approach, XIPS is a model-based prevention model for Web services application. It employs static analysis and runtime monitoring to provide better protection for Web services application against malicious XPath input. The second approach is XTrust, a severity-aware trust-based access control, which prevent malicious XPath code within XML database-stored procedures. It employs severity features to enhance the existing trust-based access control for XML database.

References

I. A. Klein, Blind XPath Injection. (2005). Whitepaper from Watchfire, Director of Security and Research, Sanctum, (2004) 1–10.

II. J. Blasco, Introduction to XPath Injection techniques, (2007), 24–31.

III. Jinghua Groppe,Sven Groppe, Filtering unsatisfiable XPath queries", Journal Data & Knowledge Engineering , Vol.64 No. 1,Amsterdam, (2008) 134-169.

IV. D. Mitropoulos, Fortifying Applications Against Xpath Injection Attacks, (2009), 1169–1179.

V. N. Antunes, N. Laranjeiro, M. Vieira, & H. Madeira, Effective detection of SQL/XPath Injection vulnerabilities in web services. SCC 2009 - 2009 IEEE International Conference on Services Computing, (2009), 260–267. http://doi.org/10.1109/SCC.2009.23

VI. Shanmughaneethi, Ravichandran, & Swamynathan, PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications. International Journal on Web Service Computing, 2(3), (2011), 57–64. http://doi.org/10.5121/ijwsc.2011.2305

VII. S. Karumanchi, & A. Squicciarini, A Large Scale Study of Web Service Vulnerabilities. Journal of Internet Services and Information Security (JISIS), 5(1), (2015), 53-69.