Safeguarding a Native XML Database System

Yuman Huang
Computer Science Department, the University of Auckland

Abstract: A system with Native XML Database (NXD) can not only store some important data assets such as business to business transaction records and personal medical histories, but also provide e-services to its legitimate users. Since the information stored is so important, those NXD systems are designed with some security goals which may be hard to maintain without intrusion detection systems (IDS). It is essential to look into these security goals of NXD systems and think about solutions for preserving them. This paper will not only discuss an IDS solution to NXD systems, detecting intrusions in databases through fingerprinting transactions (DIDAFIT), but also propose a hypothetical security control design method to preserve the security goals of NXD systems.

Keywords: native XML database (NXD) system, intrusion detection system (IDS), detecting intrusions in databases through fingerprinting transactions (DIDAFIT), XQuery injection and fingerprints.

1. Introduction

In this paper, a native XML database (NXD) system refers to a system with native XML database parts. A NXD system can not only store some important data assets such as business to business transaction records and personal medical histories, but also provide e-services to its legitimate users. Because leaking such important data assets may cause some serious problems, like unfair business prizing competitions or human privacy ruin, it is necessary to make a NXD part reach a certain level of security goals [Pfle97][Thom03], before actually plugging it into a system.

Since native XML databases are not intended to replace existing databases [Kimb01], some people may argue that these data assets are not as important as those stored in the relational database (RDB), such as credit card numbers or account balances.

However, in my opinion, the most important data assets like voice, facial recognition image, fingerprint, and iris scan sources for authentication are likely to be stored in a NXD system. Once one of the four biometric authentication technologies, voice biometrics, face recognition, finger scan, and iris scan mentioned in [AHKM02] is put into practice, these data assets stored will represent an individual of a social community within database domain, which will be checked against anything related to the individual and stored within database, including its credit card numbers, account balances, medical histories, etc. In one word, a person will be represented uniquely by these data assets inside the digital world whereas in real world this is done by his/her identification card, like passport, driver license, or birth certificate. The consequences of leaking such information are unpredictable and serious even worse than the lost of identifications.

Furthermore, the NXD part of a system may lead to some security problems to the whole system because it plays a crucial role in the system as a part of system and indirectly responses to submitted queries of users. If it's vulnerable to intruders, the system would have the risks of being attacked. For this reason, I consider that it is important to look for some ways of safeguarding a NXD system by detecting intrusion on native XML database.

A new technique detecting intrusion on relational database was presented by Low et al [LLT02] in 2002, called detecting intrusions in databases through fingerprinting transaction (DIDAFIT). Since DIDAFIT identifies anomalous access on database level, it is different from existing intrusion detection system approaches: host-based misuse, host-based anomaly-based, network-based misuse, and network-based anomaly-based mentioned in [McHu01].

Because of the characteristics of DIDAFIT and the increasing importance of safeguarding a NXD system, this paper will not only discuss the DIDAFIT to NXD system solution, but also presume a hypothetical security control design method to NXD.

2. Background

2.1 Native XML Database System related Definitions

In this paper, a Native XML Database (NXD) System shown in *Figure 1* is defined as a system provides online services with native XML database parts. There are three layers in the system: application layer, database layer, and middleware. The middleware layer is between application layer and database layer, and the database layer often has a relational database (RDB) as its backend. The system can serve different classes of users through Internet.

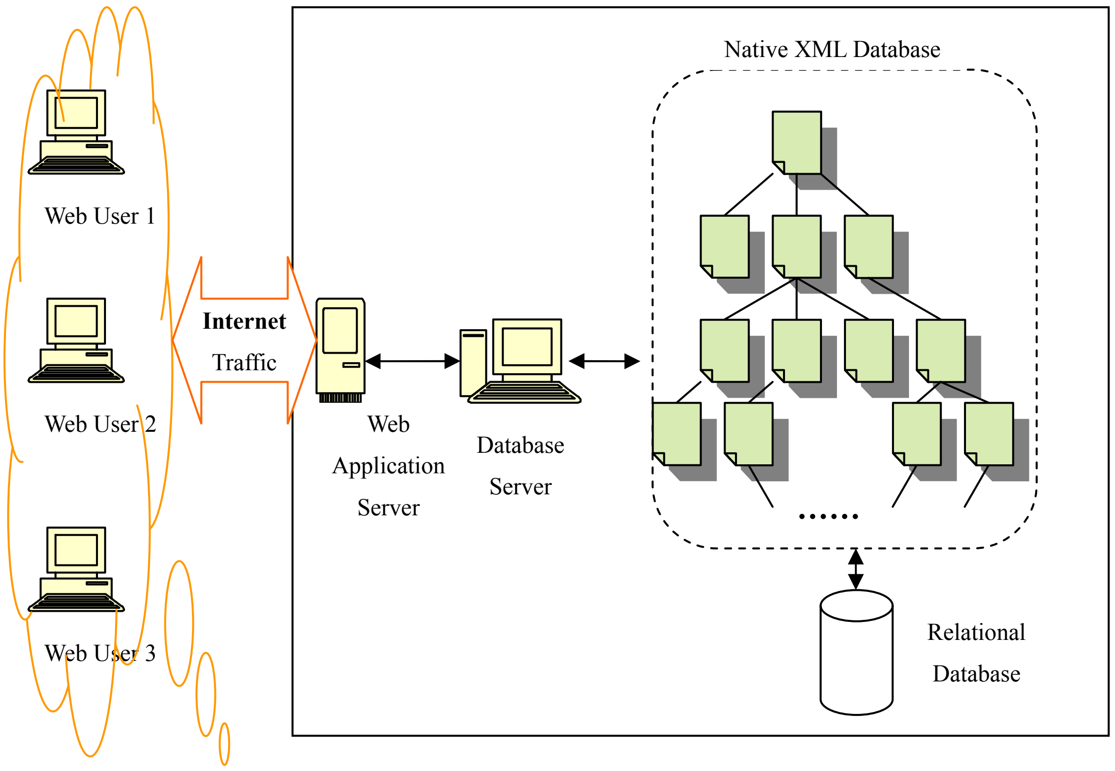

Figure 1: Native XML Database (NXD) System Architecture

Defined by XML:DB Initiative [XMLDB03], a native XML database (NXD):
"a) Defines a (logical) model for an XML document -- as opposed to the data in that document -- and stores and retrieves documents according to that model. At a minimum, the model must include elements, attributes, PCDATA, and document order. Examples of such models are the XPath data model, the XML Infoset, and the models implied by the DOM and the events in SAX 1.0.
b) Has an XML document as its fundamental unit of (logical) storage, just as a relational database has a row in a table as its fundamental unit of (logical) storage.
c) Is not required to have any particular underlying physical storage model. For example, it can be built on a relational, hierarchical, or object-oriented database, or use a proprietary storage format such as indexed, compressed files."
[Quoted from XML:DB Initiative, http://www.xmldb.org/]

While RDB handles structured data with schema, NXD is designed for dealing with semi-structured data [Dobb03]. Semi-structured data is often nested hierarchically, and cannot be modeled naturally or usefully using a standard data model like structured data. Also because of its self-describing, defining a schema for the data is not required. As its nature described above, NXD could not only be used for providing online services over the web, but also be plugged into business network to support high-speed and reliable business to business (B2B) transactions.

RDB has standard mature Structured Query Language (SQL) for data retrieval and update in major database management systems (DBMS) such as Oracle and Microsoft SQL Server. In contrast, the standard XML query languages for NXD, such as XPath, XQuery, XSLT and XUpdate, are innovative in their working drafts. But since XQuery is the most recent recommended XML query language of the World Wide Web Consortium (W3C), I will discuss the possible DIDAFIT approach to NXD based on its working draft [W3C03] in Section 3.

2.2 Intrusion, Database Intrusion & Intrusion Detection System

In computer security, intrusion refers to the act of illegitimate access or use of computer assets (hardware, software and data). Following the database intrusion definition given by Low et al [LLT02], in this paper, native XML database intrusion is defined as "the act of individuals or groups of individuals who use the native XML database without authorization, and those who are authorized, but abuse their privileges". A NXD intrusion detection system (IDS) aims to detect NXD database intrusions.

2.3 DIDAFIT & its learning fingerprints process

DIDAFIT is a new IDS technique presented by Low et al [LLT02] in 2002, which detects relational database intrusions by matching incoming transactions with fingerprints of the signature database at the application level. Its architecture is shown in *Figure 2*. It can be seen from the figure that only one class of application user is considered in the system and building up the signature database is an essential part of the architecture.

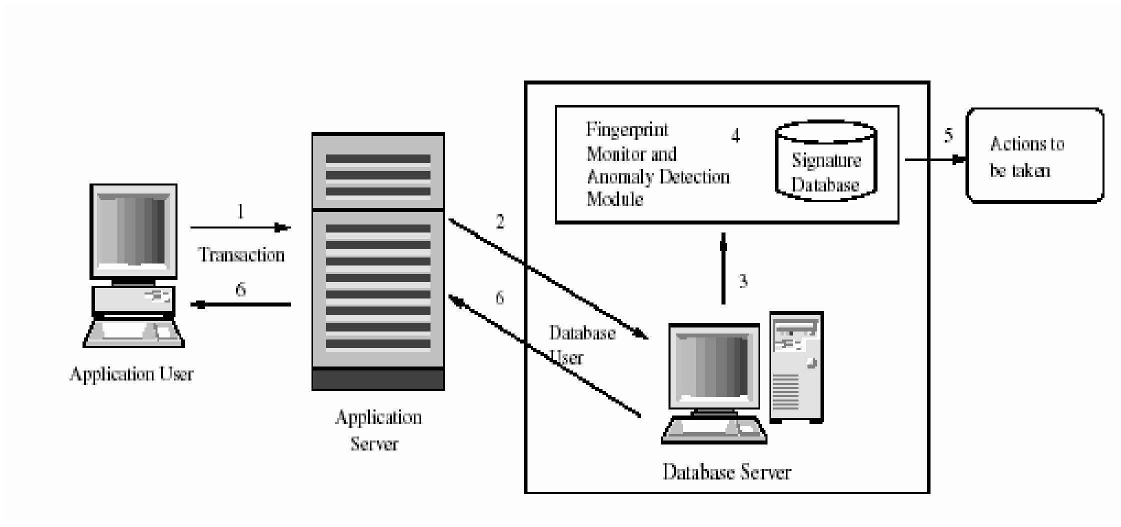

Figure 2: Architecture for DIDAFIT [Quoted from Lee et al, 2002]

DIDAFIT works on a relational database based on a number of conditions. First, because it's designed as a misuse detection system, the consistency of legitimate database transactions is assumed. In other words, illegitimate transactions should be abnormal from legitimate ones where fingerprints would be summarized. Second, it says that SQL injection techniques are used in database intrusions. I consider that DIDAFIT can identify SQL injection intrusions. Third, a database administrator (DBA) takes part in the learning fingerprints process of DIDAFIT. Last, a log function for recording all SQL statements submitted to database server is supplied in a database management system (DBMS), though its initial purpose is not only for this.

3. Discussions with a working example

NXD systems may have several security goals when they were developed, and need to face some security problems caused by its NXD part because of XQuery injection technique. Whether a DIDAFIT approach can safeguard these NXD systems will be discussed in this section. In addition, because of the characteristics of NXD, a hypothetical security control design to the NXD logical data model is proposed.

For better discussing the possibility of applying the principles of DIDAFIT on a NXD, also the hypothetical security control design method, I presume a NXD data model example in *Figure 3* to illustrate the working process.

3.1 Security Goals of a NXD system

Native XML Database is designed to handle extensible markup language (XML) documents, and XML was initially defined as a replacement of HTML for web information sharing purpose. A NXD system can provide different online services to satisfy different users. In the working example presumed, a NXD system provides passport management service and banking services to different users including residents of a country, immigration department, police department, and a bank, so its NXD database stores several XML documents named residents, passports, face images, accounts, and customers. The data model defined is shown in *Figure 3*, its data redundancy is not considered.

The NXD system may have a RDB as its backend to support the storage of structured data, such as banking transaction details, but as only considering the security problems of the NXD part, the internal communication strategies between NXD and RDB will not be questioned.

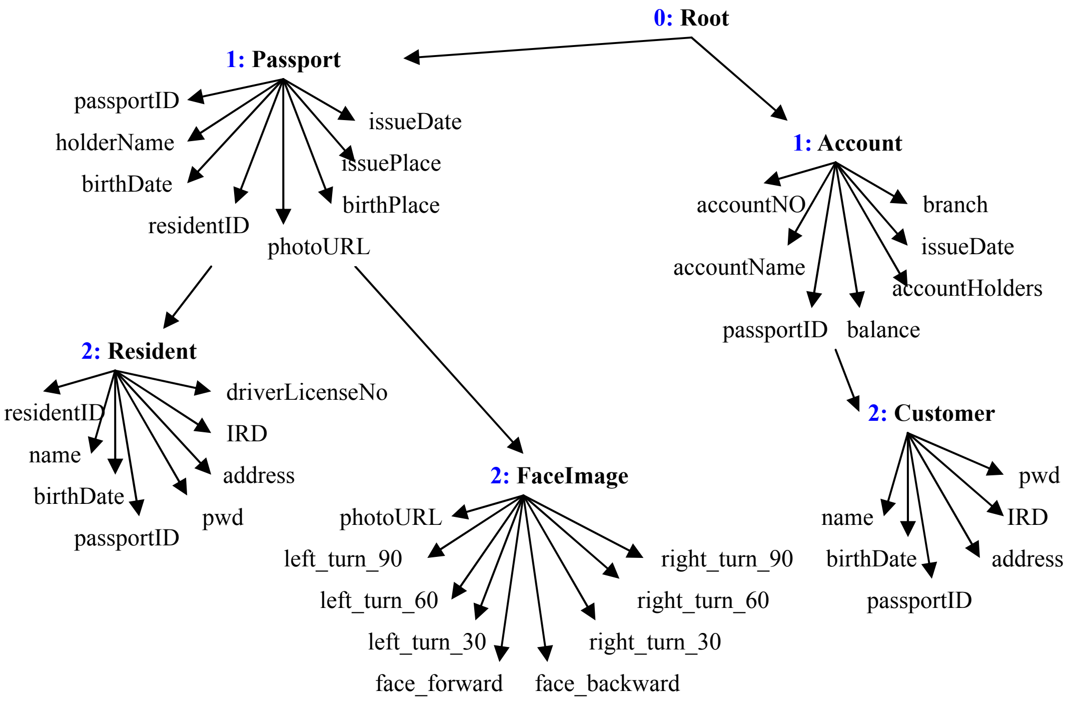

Figure 3: The data model fragment of the working example

In the NXD system, Pfleeger's security goals (Confidentiality, Integrity, and Availability) [Pfle97][Thom03] are presented in some ways which are described following, and be summarized in *Figure 4*.

After logging in, a resident can view and modify his/her own residential information, can view his/her own passport and account information in the NXD, cannot view or modify any other residents' information.

After logging into a machine located in the airport and also the passport checking system, an immigration officer can view all residents' passport records, insert new departure or arrival records corresponding to the bar code read from the passport, but not modify any existing records and view/update any account data.

Passport photo enrolment only can be done in authorized place and by authorized persons. After the enrolment, passport photo only can be reenrolled with a new passport request. It cannot be modified by any parties, and only viewed by working immigration and police officers while need.

After logging into a machine located in the bank and also the banking teller system, a teller can create a new account according to customer request plus his/her passport copy, view or update account details including address and account balance according to customer request and system balance checks, but he/she cannot access customer's passport record.

After logging into a machine in a police station and also the residential management system, a police officer can view people's passport information with face images and residential records, but not modify any. Even for police, account details are confidential, only can request to view in the bank by showing some investigation permission papers.

Users / XML documents	Resident		Immigation		Authorised		Police		Bank	
	read	write	read	write	read	write	read	write	read	write
resident	yes	yes	yes	no	no	no	yes	no	no	no
passport	yes	no	yes	yes	no	no	yes	no	no	no
faceImageDB	no	no	yes	no	no	yes	yes	no	no	no
account	yes	no	no	no	no	no	no	no	yes	yes
customer	no	no	no	no	no	no	no	no	yes	yes

Figure 4: A working example with Pfleeger's security goals

Note: Confidentiality (unauthorized people can't read); Integrity (unauthorized people can't write); Availability (authorized people can read and write). **[Quoted Pfleeger's security goals in [Pfle97][Thom03]];** "yes" with black 'y' means yes with conditions, details explained in words above the figure.

In conclusion, for different users of the system, their corresponding views to the system are different so should be classified differently in order to achieve certain level of the security goals. If DIDAFIT is practicable to NXD, its signature database should be designed specifically for safeguarding NXD from different users. Further discussion is in section 3.2.

3.2 Basics of DIDAFIT approach to NXD

DIDAFIT works to detect database intrusions based on SQL injection technique, and needs to have a signature database, which stores legitimate database transaction fingerprints, to identify illegitimate access [LLW02][LLT02]. If XML query languages injection technique does not exist, or the transaction statements of XML query languages do not have any regular patterns to be summarized to fingerprints, it is obvious that the discussion of applying DIDAFIT approach to NXD would not make sense. This section will look into these prerequisites before continuing the discussion.

As said in section 2.1, XQuery is the most recent recommended XML query language of the World Wide Web Consortium (W3C), so XML query languages injection and fingerprints will be specified as XQuery injection and fingerprints in the following sections.

3.2.1 XQuery Injection

Like in a RDB, you can send an XQuery for-let-where-return (FLWR) expression statement to retrieve a data model instance from a NXD. For example, assumes that “residents”, “passports”, “accounts” and “customers” in *Figure 3* are corresponding to tables with the same names in a RDB, also the text data stored are exactly the same as those in NXD.

purpose	A resident Jennifer can only see her own residential record by providing her resident id and correct password. If either id or password is not matching, no data is returned.
SQL	select * from residents where residentID=’SC00000001’ and pwd =’12345678’;

XQuery	for $r in /residents/resident *`/*FROM in SQL*/`*
	where $x/residentID=”SC00000001” and $y/pwd=”12345678” *`/*WHERE in SQL*/`*
	return $r *`/*SELECT in SQL*/`*

Figure 5: Comparison between SQL and XQuery statements

Although two statements offer the same functionality, the input and output for them are different. In RDB, queuing on a database has a table “residents”, a row of the table where resident id is “SC00000001” will be returned, whereas in NXD, queuing on the XML document “residents”, a new XML document with a element whose resident id attribute has text “SC00000001” is returned. Similar XQuery statements could be used to maintain the read integrity and confidentiality of the working example shown in *Figure 4*.

On “XQuery 1.0: An XML Query Language, W3C Working Draft 02 May 2003” [W3C03], no update and delete statements are defined, though its editors mention that update statements may be supported by future versions. There is XUpdate update language defined by XML:DB Initiative [XMLDB00] to update XML documents and it’s also recommended by W3C. Since the learning fingerprints process presented in [LLW02] for SQL select, delete, update, insert statements on RDB are similar, I will not specifically discuss the issues of XUpdate on NXD.

Is it necessary to worry about intrusions based on XQuery injection? The answer is positive, not only because of the similiarity of XQuery and SQL and the increasing popularity and importance of NXDs, but also the similar architectures of NXD and RDB systems.

As SQL injection being defined in [LLW02], XQuery injection refers to the technique cracks XQuery statements using “string building” methods to trick the application server into executing the malicious code submitted by intruders. Because modification functions are not yet supported by XQuery, the possible results caused by injection type intrusions are not as serious as written by Lee et al in [LLW02].

However, the confidentiality of a NXD system can be broken by using XQuery injection technique, information may available to unauthorised parties, the privacy rights of individuals are ruined. In the working example, Jennifer can view not only her own residential information if she knows about how to use XQuery injection, but also those of others. On the other side, she will no longer trust the system because it is insecure and her personal information may view by the others.

Following the original SQL injection ideas, a XQuery injection code should looks like the left of in *Figure 6*, under assumptions that the web application uses Perl, Perl supports the application as it does in the application of old DIDAFIT, Perl supports XQuery the same way as it does to SQL, and in Perl, the double quotation marks inside XQuery statements are replaced by single quotation mark, because I think that the syntax *““$name””* (a varible name inside two pairs of double quotation marks) may lead to confusions to audios, and also errors in compiling time.

Assume Adam enters “x’ OR ‘x’=’x” into the text field for password in submitted form, because of the keyword “OR”, the system returns a XML document with a element whose resident id is text “SC00000001” even the inputed password is wrong.

```
/*Perl script*/
my $passwd = $cgi->param('passwd');
my $name = $cgi->param('name');
$sql = "for $r in /residents/resident".
        " where $x/residentID='$name'".
        " and $y/pwd='$passwd'".
        " return $r";
$sth = $dbh->prepare($sql);
$sth = $dbh->execute;
if (!($sth->fetch)) { report_illegal_user();
} ...
```

```
/*possible XQuery injection code*/
for $r in /residents/resident
where $x/residentID=”SC00000001”
        and $y/pwd=”x”
        or ‘x’=’x’
return $r
```

Figure 6: Possible XQuery injection code [modified from Section 2.1 of [LLW02]]

There are some more SQL injection techniques introducted on the papers refered by Lee et al. It seems that some ideas of those techniques (excluding other ideas which closely relate to update, insert, delete statements and specific DBMSs such as Microsoft SQL server) can also be transformed on XQuery, because XQuery serves the same purpose to NXD as that of SQL to RDB. Hence the analysis to XQuery injection stands at this point whereas that to XQuery fingerprints starts.

3.2.2 XQuery Fingerprints

Most existing web applications for database services can fit into a NXD system in the same matter as that for RDB system, since the three-layer architecture of a NXD system is the same as that of a RDB system. In these applications of a NXD system, the transaction statements written in SQL would be replaced by those of standard XML query languages such as XQuery. Because XQuery statements are generated by application server programs, they are likely to form some sort of regular patterns. From these regular patterns, XQuery fingerprints of legitimate transactions can be summarized and used for identifying illegitimate statements.

In theory, the DIDAFIT fingerprint learning algorithm presented by Lee et al in [LLW02] can work in a NXD system to summarize XQuery fingerprints into regular expressions in consequence of the existence of XQuery WHERE-clause. However, I consider that some situations may affect the performance of the algorithm while it serves a NXD system.

First, the trace log is made up of XQuery expressions, so they may need to be pre-processed before being submitted to the fingerprint learning process. Second, the actual data model defined in NXD influences the way of storing and retrieving data. Third, attributes and columns in RDB are represented as nodes within an XML document or a collection of XML documents in NXD, so the actual operations for comparing their value may be implemented differently.

In this case, the fingerprint learning algorithm needs to be customized in order to go with the specific NXD environment, and also for ensuring the accuracy of the signature database, the database administrator (DBA) involved in the process should have additional knowledge and experiences of NXD.

As the discussion goes, I believe that the basic principles of DIDAFIT and its fingerprint learning algorithm do work for a native XML database (NXD) system if it uses a database server which is able to log all submitted XQuery statements, because the rest of the prerequisites for DIDAFIT mentioned in section 2.3 are all satisfied.

As mentioned previously, because the views of different users to the system are different, so the fingerprints of a specific class of users will have their own patterns. In DIDAFIT, signature database should be designed specifically for safeguarding NXD from different users. For a RDB, more than one signature databases are required in order to provide different security views for different users. This is also practicable for a NXD. The overall architecture for DIDAFIT to a NXD system is shown in *Figure 7*.

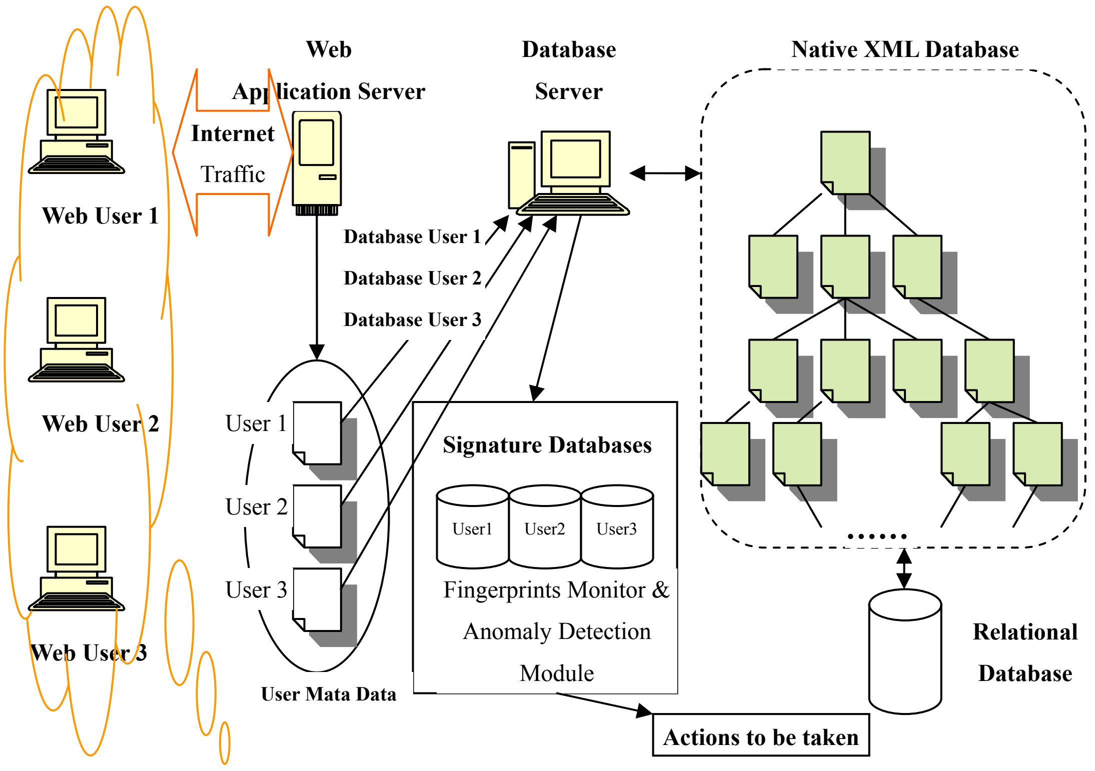

Figure 7: Architecture for DIDAFIT to a NXD system

Assume that a set of documents has been generated for describing the properties of every category of database users, and also the signature (legitimate fingerprints) databases for different database users has been built up.

Web application users must log into application server to use the online services provided by the system. According to user login information, the application server knows which category the user belongs to and issues him/her to the database server through a corresponding database user. The database user logs into the database and submits some XQuery statements to the database through the application. The XQuery statements are directed to monitor and anomaly detection module to match with the set of legitimate fingerprints in the signature database of the database user. If any statements are appeared to be abnormal, intrusions are directed to reaction module, actions like alarming on console will be taken, otherwise, the user gets back the output for the query. [modified from the paper [LLW02]].

3.3 Hypothetical Security Control Design to NXD

Security is so important for NXDs for the reason that they are always used in a system providing online services, so we must have some control over the database in order to keep our crucial data secure. Since XML documents are tree-structured and NXD is designed to handle XML documents, the data model of NXD can be a directed graph as being defined in XQuery and XPath Data Model [W3C03]. In a directed graph, a path always can be found when the start and the end are constant. My hypothetical security control design is based on the path of the logical data model defined by XQuery and XPath in [W3C03].

There are some **blue** numbers added in front of each document node the data model shown in *Figure 3*, because I think that they can indicate some sort of security levels. The root, which is not referred by any of the others, is attached the least number inside the whole graph. This means that it is in the highest security level inside the database and should be most restricted for accessing. Along with the direction goes, the number is increased, and the security restriction is loosed.

One category of users can be assigned to a path of the graph together with a user security number. The users can only access the nodes, which have the equal or larger number than their user security number, and on the path assigned. For example, a path root to passport to resident and security number one are assigned to residents who live in the country, so they can only access the passport and the resident XML documents, but not the passport face image; path root to account to customer and security number two is also assigned to residents, so they can only access the customer XML document but not the account; path root to passport to face image and number two are assigned to authorized face image enrolment people, so they can only access the face image but not the passport or resident.

There are different types of access mention in Pfleeger's security goals, like read and write. To restrict the type of the access, an extra user access security number can be added to indicate this, number one for only read, two for only write, and three for both. For example, because residents can read and modify their residential information, so path root to passport to resident, security number two, and user access security number three are assigned to them. You might find that the security number is changed from one to two, and this is because residents must not be able to modify their passport records. So for giving read passport record permission, path root to passport, security number one, and user access security number one are also assigned to residents. It is clear that when access type is specified, more operations need to be done because more security controls are given to each operation.

By adding this special element to all XML documents of NXD, this hypothetical method relates a category of users to a path of the data model in the NXD so that controls the corresponding views of these users to the system. However, it cannot restrict to the lower level of a document node. For example, a specific resident named "Jennifer" should only view about her own information, but not others. Since the view becomes specific, the restriction cannot cover at element node level.

There are two ways to solve the problem. First, to make the path specific, split a document node into several document nodes. For example, originally a "residents" XML document has more than one element nodes, and one of them is "Jennifer" element node. For specifying the path, turn each of the "resident" element nodes into a document node, so all residential information for "Jennifer" is in a document, and only information of a resident called "Jennifer" can be found in this document. Because now a resident record is an XML document, it can be assigned its own security number, and a path to retrieve a specific resident named "Jennifer" can be formed. Second, XQuery limits accesses on element node level by using WHERE-clause.

This method is presumed to control the security of a NXD by adjusting its data model. The most important XML documents should be place at the top level of the model, and also users should only access other documents by following the uniform resource locators on his/her authorized document. Since this hypothetical security control design method to NXD is based on the logical data model defined by XQuery and XPath in [W3C03], it might not work with other types of data models, also it may be difficult to assign security number to the deeply nested data model with crossing references.

4. Conclusion

DIDAFIT and its learning fingerprints algorithm presented by Low et al and Lee et al in [LLW02][LLT02] can not only protect a relational database system, but also safeguard a native XML database system with some transformations. The implementation of actual solution depends on the actual characteristics of the database, the query language used, and the security goals of the system. According to the nature of native XML database, I proposed a security control design method to manage its user accesses. Because of the importance of security and the innovation of native XML database, I believe that further solutions for safeguarding a NXD system will be proposed in future. Once developers have security in mind when they are designing a NXD system, the system they build will be secure in certain levels, then safeguarding a system will be not as difficult as nowadays.

Acknowledgement

I would like to thank Professor Clark Thomborson and Professor James Goodman for their useful comments and suggestions.

References

[LLW02] Lee, S. Y., Low, W. L., and Wong, P. Y., "Learning Fingerprints for a Database Intrusion Detection System", in D. Gollmann, G.Karjoth, M. Waidner (Eds.): Computer Security - ESORICS 2002, Proc. 7th European Symposium on Research in Computer Security Zurich, Switzerland, LNCS 2502, p. 264ff.

[LLT02] Low, W. L., Lee, S. Y., Teoh, P., "DIDAFIT: Detecting Intrusions in Databases Through Fingerprinting Transactions", in Proc. of the 4th International Conference on Enterprise Information Systems (ICEIS), pp. 264-269, 2002.

[McHu01] McHugh J, "Intrusion and Intrusion Detection", International Journal of Information Security 1, 2001, pp. 14-35.

[XMLDB00] XML:DB Initiative organization, “XUpdate Working Draft” technical report 2000.

[XMLDB03] XML:DB Initiative organization, “Introduction to XML: DB Initiative for XML Databases” and “FAQ: What is an XML Database?”, http://www.xmldb.org/, April 20 2003.

[Kimb01] Kimbro, S., “Introduction to Native XML Databases”, October 31 2001.

[AHKM02] Armington, J., Ho, P., Koznek, P. and Martinez, R., “Biometric Authentication in Infrastructure Security”, LNCS 2437, pp.1-18, 2002.

[Pfle97] Pfleeger, C. P., “Is there a security problem in computing?”, Security in Computing, Prentice Hall PTR, 1997.

[Thom03] Clark Thomborson, Handout 4 for Software Security, CompSci 725 S1 C 2003, Computer Science department, the University of Auckland.

[Dobb03] Dobbie, G., Lecture notes for Software Tools and Techniques, CompSci 732 S1 C 2003, Computer Science department, the University of Auckland.

[W3C03] W3C Consortium
“XQuery 1.0: An XML Query Language - W3C Working Draft”, 02 May 2003, version http://www.w3.org/TR/2003/WD-xquery-20030502/.
“XQuery 1.0 and XPath 2.0 Data Model - W3C Working Draft” 02 May 2003, version http://www.w3.org/TR/2003/WD-xpath-datamodel-20030502/.