Volume 3, Issue 5, May 2013 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Anomaly detection in RDBMS

P. Senthil
Dept of CSE Bharath University
India.

S. Pothumani
Dept of CSE Bharath University
India.

Abstract Intrusion and malicious modification are more prevalent in Database so Organizations show considerable effort in maintaining the secrecy and security of the Database Management System. It shows interest in Database activity monitoring and anomaly detection .Once an anomalous request is found suitable response scheme is needed to respond. Then once a request is found there are three response actions involved .Those are conservative, fine grained and aggressive response actions. The main problem is issuing suitable response to the detected anomalous request. In this paper two issues Policy matching and Policy administration are addressed and to overcome the problems caused due to anomalous request. With the help of Event Condition Action (ECA) triggering database rule the necessary things are verified. Especially it involve in verifying the two policies to issue suitable response for the detected anomalous request. A novel approach to dynamically change the state of access of the system is also proposed with multi hand administration of relational databases

Index Terms—Databases, intrusion detection, response, prevention, policies, query limitation.

I. Introduction

RECENTLY, we have seen an interest in products that continuously monitor a database system and report any relevant suspicious activity [1]. Database activity monitor- ing has been identified by Gartner research as one of the top five strategies that are crucial for reducing data leaks in organizations [2], [3]. Such step-up in data vigilance by organizations is partly driven by various US government regulations concerning data management such as SOX, PCI, GLBA, HIPAA, and so forth [4]. Organizations have also come to realize that current attack techniques are more sophisticated, organized, and targeted than the broad-based hacking days of past. Often, it is the sensitive and proprietary data that is the real target of attackers. Also, with greater data integration, aggregation and disclosure, preventing data theft, from both inside and outside organizations, has become a major challenge. Standard database security mechanisms, such as access control, authentication, and encryption, are not of much help when it comes to preventing data theft from insiders [5]. Such threats have thus forced organizations to reevaluate security strategies for their internal databases [4]. Monitoring a database to detect potential intrusions, intrusion detection (ID), is a crucial technique that has to be part of any comprehensive security solution for high-assurance data- base security. Note that the ID systems that are developed must be tailored for a Database Management System (DBMS) since database-related attacks such as SQL injection and data exhilaration are not malicious for the underlying operating system or the network. Our approach to an ID mechanism consists of two main elements, specifically tailored to a DBMS: an anomaly- detection (AD) system and an anomaly response system. The first element is based on the construction of database- access profiles of roles and users, and on the use of such profiles for the AD task. A user-request that does not conform to the normal access profiles is characterized as anomalous. Profiles can record information of different levels of details; we refer the reader to [6] for additional information and experimental results. The second element of our ap- proach—the focus of this paper—is in charge of taking some actions once an anomaly is detected. There are three main types of response actions, that we refer to, respectively, as conservative actions, fine-grained actions, and aggressive actions. The conservative actions, such as sending an alert, allow the anomalous request to go through, whereas the aggressive actions can effectively block the anomalous request. Fine-grained response actions, on the other hand, are neither conservative nor aggressive. Such actions may suspend or taint an anomalous request [5], [6]. A suspended request is simply put on hold, until some specific actions are executed by the user, such as the execution of further authentication steps. A tainted request is marked as a potential suspicious request resulting in further monitoring of the user and possibly in the suspension or dropping of subsequent requests by the same use With such different response options, the key issue to address is which response measure to take under a given situation. Note that it is not trivial to develop a response mechanism capable of automatically taking actions when abnormal database behavior is detected. Let us illustrate this with the following example. Consider a database monitoring system in place that builds database user profiles based on SQL queries submitted by the users. Suppose that a user U , who has rarely accessed table T , issues a query that accesses all columns in T . The detection mechanism flags such request as anomalous for U . The major question is what should the system do next once a

© 2013, IJARCSSE All Rights Reserved

request is marked as anomalous by the AD mechanism. Since the anomaly is detected based on the learned profiles, it may well be a false alarm. It is easy to see then there are no simple intuitive response measures that can be defined for such security-related events. If T contains sensitive data, a strong response action is to revoke the privileges corresponding to actions that are flagged as anomalous. In our example, such a response would translate into revoking the select privilege on table T from U . However, if the user action is a one-time action part of a bulk-load operation, when all objects are expected to be accessed by the request, no response action may be necessary. The key idea is to take different response actions depending on the details of the anomalous request, and the context surrounding the request (such as time of the day, origin of the request, and so forth). Therefore, a response policy is required by the database security administrator to specify appropriate response actions for different circumstances. In this paper, we propose a high-level language for the specification of such policies which makes it very easy to specify and modify them The two main issues that we address in the context of such response policies are that of policy matching and policy administration. Policy matching is the problem of searching for policies applicable to an anomalous request. When an anomaly is detected, the response system must search through the policy database and find policies that match the anomaly. Our ID mechanism is a real-time intrusion detection and response system; thus efficiency of the policy search procedure is crucial. In Section 4, we present two efficient algorithms that take as input the anomalous request details, and search through the policy database to find the matching policies. We implement our policy matching scheme in the PostgreSQL DBMS [5], and discuss relevant implementation issues.

The second issue that we address is that of administration of response policies. Intuitively, a response policy can be considered as a regular database object such as a table or a view. Privileges, such as create policy and drop policy, that are specific to a policy object type can be defined to administer policies. However, a response policy object presents a different set of challenges than other database object types. Recall that a response policy is created to select a response action to be executed in the event of an anomalous request. Consider the case of an anomalous request from a user assigned to the DBA role. Since a DBA role is assigned all possible database privileges, it will also possess the privileges to modify a response policy object. Now consider a scenario, where organizational policies require auditing and detection of malicious activities from all database users including those holding the DBA role. Thus, response policies must be created to respond to anomalous requests from all users. But since a DBA role holds privileges to alter any response policy, it is easy to see that the protection offered by the response system against a malicious DBA can trivially be bypassed. The fundamental problem in such administration model is that of conflict-of-interest. The main issue is essentially that of insider threats, that is, how to protect a response policy object from malicious modifications made by a database user that has legitimate access rights to the policy object. The rest of the paper is organized as follows: Section 2 presents the details of the response policy language. Section 3 extended ECA policies. We discuss the policy matching algorithms in Section 4. Section 5 discusses related work. We conclude in Section 6 with directions for future work.

II. Policy Language

The detection of an anomaly by the detection engine can be considered as a system event. The attributes of the anomaly, such as user, role, SQL command, then correspond to the environment surrounding such an event. Intuitively, a policy can be specified taking into account the anomaly attributes to guide the response engine in taking a suitable action. Keeping this in mind, we propose an Event- Condition-Action (ECA) language for specifying response policies. Later in this section, we extend the ECA language to support novel response semantics. ECA rules have been widely investigated in the field of active databases[6]. An ECA rule is typically organized as follows: ON {Event} IF {Condition} THEN {Action} As it is well known, its semantics is as follows: if the event arises and the condition evaluates to true, the specified action is executed. In our context, an event is the detection of an anomaly by the detection engine. A condition is specified on the attributes of the detected anomaly. An action is the response action executed by the engine. In what follows, we use the term ECA policy instead of the common terms ECA rules and triggers to emphasize the fact that our ECA rules specify policies driving response actions. We next discuss in detail the various components of our language for ECA policies.

Anomaly Attributes

The anomaly detection mechanism provides its assessment of the anomaly using the anomaly attributes. We have identified two main categories for such attributes. The first category, referred to as contextual category, includes all attributes describing the context of the anomalous request such as user, role, source, and time. The second category, referred to as structural category, includes all attributes conveying information about the structure of the anomalous request such as SQL command, and accessed database objects. Details concerning these attributes are reported in Table 1. The detection engine submits its characterization of the anomaly using the anomaly attributes. Therefore, the anomaly attributes also act as an interface for the response engine, thereby hiding the internals of the detection mechanism. Note that the list of anomaly attributes provided here is not exhaustive. Our implementation of the response system can be configured to include/exclude other user-defined anomaly attributes.

Policy Conditions

A response policy condition is a conjunction of predicates where each predicate is specified against a single anomaly attribute. Note that to minimize the overhead of the policy matching procedure (cf. Section 4),

© 2013, IJARCSSE All Rights Reserved

we do not support disjunctions between predicates of different attributes such as SQLCmd = “Select” OR “IPAddress” = “10.10.21.200.” However, disjunctions between predicates of the same attribute are still supported.

Table 1
Anomaly attributes

Attribute	Description
CONTEXTUAL	
User	The user associated with the request.
Role	The role associated with the request.
Client App	The client application associated with the request.
Source IP	The IP address associated with the request.
Date Time	Date/Time of the anomalous request.
STRUCTURAL	
Database	The database referred to in the request.
Schema	The schema referred to in the request.
Obj Type	The object types referred to in the request such as table, view, stored procedure
Obj(s)	The object name(s) referred in the request
SQLCmd	The SQL Command associated with the request
Obj Attr(s)	The attributes of the object(s) referred in the request.

For example, if an administrator wants to create a policy with the condition SQLCmd = “Select” OR SQLCmd = “Insert”; such condition can be supported by our framework by specifying a single predicate as SQLCmd IN {“Select”, “Insert”}. More exam- ples of such predicates are given below:
Role != DBA
Source IP IN 192.168.0.0/16
Objs IN {dbo.*}
We formally define a response policy condition as follows:
Definition (Policy Condition). Let PA ={ A1 , A2 ... An} be the set of anomaly attributes where each attribute Ai has domain Ti of values. Let a predicate Pr be defined as P r: $A_k\Theta c$, where $A_K \in$ P A, is a comparison operator in {>, <, >=, <=, =,! =; like ,IN, BETWEEN }, and c is a constant value in T_k. The condition of a response policy Pol is defined as Pol(C): Prk and Prl and ... and Prm where Prk , Prl ... Prm are predicates of type Pr.

Response Actions

Once a database request has been flagged off as anomalous, an action is executed by the response system to address the anomaly. The response action to be executed is specified as part of a response policy. Table 2 presents a taxonomy of response actions supported by our system. The conservative actions are low severity actions. Such actions may log the anomaly details or send an alert, but they do not proactively prevent an intrusion. Aggressive actions, on the other hand, are high severity responses. Such actions are capable of preventing an intrusion proactively by either dropping the request, disconnecting the user or revoking/denying the necessary privileges. Fine-grained response actions are neither too conservative nor too aggressive. Such actions may suspend or taint an anomalous request. A suspended request is simply put on hold, until some specific actions are executed by the user, such as the execution of further authentication steps. A tainted request is simply marked as a potential suspicious request resulting in further monitoring of the user and possibly in the suspension or dropping of subsequent requests by the same user. We refer the reader to [6] for further details on request suspension and tainting. Note that a sequence of response actions can also be specified as a valid response. For example, LOG can be executed before ALERT in order to log the anomaly details, as well as send a notification to the security administrator.

Table 3 describes two response policy examples. The threat scenario addressed by Policy 1 is as follows: In many cases, the database users and applications have read access to the system catalogs tables by default. Such access is sometimes misused during a SQL Injection attack to gather sensitive information about the DBMS structure. An anomaly detection engine will be able to catch such requests, since they will not match the normal profile of the user. Suppose that we want to protect the DBMS from anomalous reads to the system catalogs (“dbo” schema) from unprivileged database users. Policy one aggressively prevents against such attacks by disconnecting the user. Policy two prevents the false alarms originating from the privileged users during usual business hours. The policy is formulated to take no action on any anomaly that originates from the internal network of an organization from the privileged users during normal business hours.

© 2013, IJARCSSE All Rights Reserved

Table 2
Taxonomy of response actions

Action	Description
CONSERVATIVE: low severity	
NOP	No OPeration. This option can be used to filter unwanted alarms.
LOG	The anomaly details are logged.
ALERT	A notification is sent.
FINE-GRAINED: medium severity	
TAINT	The request is audited.
SUSPEND	The request is put on hold till execution of a confirmation action.
AGGRESSIVE: high severity	
ABORT	The anomalous request is aborted.
DISCONNECT	The user session is disconnected.
REVOKE	A subset of user-privileges are revoked.
DENY	A subset of user-privileges are denied.

Interactive ECA Response Policies

An ECA policy is sufficient to trigger simple response measures such as disconnecting users, dropping an anomalous request, sending an alert, and so forth. In some cases, however, we need to engage in interactions with users.

Table 3
Response policy

Policy 1
ON *ANOMALY DETECTION* IF *Role != DBA* and *Obj Type = table* and *Objs IN dbo.** and *SQLCmd IN {Select}* THEN *DISCONNECT*
Policy 2
ON *ANOMALY DETECTION* IF *Role = DBA* and *Source IP IN 192.168.0.0/16* and *Date Time BETWEEN* 0800 - 1700 THEN *NOP*

For example, as described in Section 2.2, suppose that upon detection of an anomaly, we want to execute a fine- grained response action by suspending the anomalous request. Then we ask the user to authenticate with a second authentication factor as the next action. In case the authentication fails, the user is disconnected. Otherwise, the request proceeds. As ECA policies are unable to support such sequence of actions, we extend them with a confirmation action construct. A confirmation action is the second course of action after the initial response action. Its purpose is to interact with the user to resolve the effects of the initial action. If the confirmation action is successful, the resolution action is executed, otherwise the failure action is executed.1 Thus, a response policy in our framework can be symbolically represented as follows:2

ON {Event}
IF {Condition}
THEN {Initial Action} CONFIRM {Confirmation Action} ON SUCCESS {Resolution Action}
ON FAILURE {Failure Action}

An example of an interactive ECA response policy is presented in Table 4. The initial action is to suspend the anomalous user request. As a confirmation action, the user is prompted for reauthentication. If the confirmation action fails, the failure action is to abort the request and disconnect the user. Otherwise, no action is taken and the request is processed by the DBMS.

RSA Public-Private Keys.

The DBMS chooses p, q as two large prime numbers such that

2p'=2p'+1 and q=2q'+1,

Where p' and q' are themselves large primes. Let n=p*q be the RSA modulus. Let m = =p'*q' The DBMS also chooses e as the RSA public exponent such that e > l. Thus, the RSA public key is PK = (n,e).The server also computes the private key d 2 Z such that

de≡1 mod m.

Secret Key Shares.

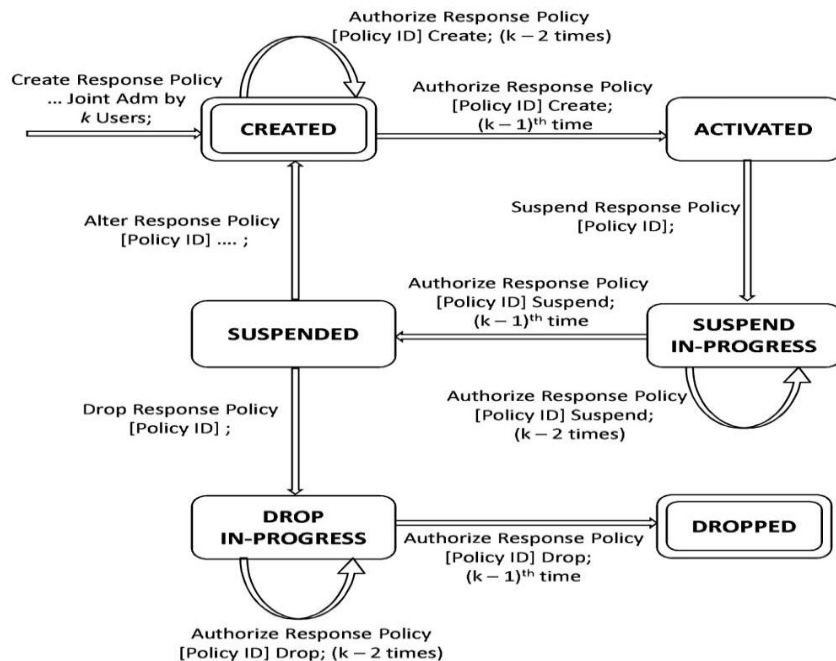
The next step is to create the secret key shares for each of the *l* DBAs. For this purpose, the DBMS sets a0= d and randomly assigns a_i from {0,…..,m-1}for 1≤i≤k-1.The numbers $\{a_0 \ldots a_{k-1}\}$ define the unique polynomial p(x) of degree k - 1,

$p(x)= \sum^{i=k-1} a_i x^i$. For $1 \leq i \leq l$ the server computes the secret share, si, of each DBA, DBAi, as follows:
$s_i = p(i) \mod m$

The secret shares can be stored in a smartcard or a token for every DBA, and submitted to the DBMS when required to sign a policy. The other alternative, that we implement in JTAM, is to let the DBMS store the shares in the database encrypted with keys generated out of the DBA's passwords. Thus, during the registration phase, each DBA must submit its password to the DBMS for encrypting its secret key shares. Using this strategy, whenever a DBA needs to sign a policy for authorization, it submits its password which is used by the DBMS to decrypt the DBA's secret share, and use that to generate the correct signature share. The three key observations regarding the registration phase of JTAM are as follows: First, the security parameters, that is, the public-private key pairs, and the secret shares, need to be generated for every k value($2 \leq k \leq l-1$) , and not for every policy. This means that any policy that uses the same value of k will have the same security parameters. Second, the private key d is only used temporarily to generate the secret key shares and is not stored by the DBMS. Third, the registration phase needs to be performedas an ACID database transaction.

III. Lifecycle of a Response Policy Object

In this section, we describe the signature share generation, the signature share combining, and the final signature verification operations, in the context of the administrative lifecycle of a response policy object. The steps in the lifecycle of a policy object are policy creation, activation, suspension, alteration, and deletion. The lifecycle is shown in Fig. 1 using a policy state transition diagram. The initial state of a policy object after policy creation is CREATED. After the policy has been authorized by k -1 administrators, the policy state is changed to



Policy state transition diagram

Attacks and Protection

In this section, we describe possible attacks on JTAM and strategies to protect from them. Recall that the threat scenario that we address is that a DBA has all the privileges in the DBMS, and thus it is able to execute arbitrary SQL commands on the sys_response_policy catalog.

Signature Share Verification

It is possible for a malicious administrator to replace a valid signature share with some other signature share that is generated on a different policy definition. However, such attack will fail as the final signature that is produced by the signature share combining algorithm will not be valid. Note that by submitting an invalid signature share, a malicious administrator can block the creation of a valid policy. We do not see this as a major problem since the threat scenario that we address is malicious modifications to existing policies, and not generation of policies themselves.

Final Signature Verification

A final signature on a policy is present in all the policy states except the CREATED state. As described earlier, the final signature is verified using the public key (n, e)corresponding to the value of k. The public key is assumed to be signed using a trusted third party certificate that cannot be forged. Thus, if a malicious DBA replaces the server generated public

© 2013, IJARCSSE All Rights Reserved

key, the final signature will be invalidated. Apart from verifying the final signature immediately after policy activation, suspension, and drop, the signature must also be verified before a policy may be considered in the policy matching procedure. Such strategy ensures that only the set of response policies, that have not been tampered, are considered for responding to an anomaly. Note that RSA signature verification requires modular exponentiation of the exponent e. The overhead to carry out such modular exponentiation increases with the number of bits set to one in the exponent e. As we show later in our experiments, an appropriate choice of e, such as 3, 17, or 65,537 can lead to a very low signature verification overhead. However, the cumulative overhead of verifying signatures on every policy during the policy matching procedure may be high. An alternative strategy is to create a dedicated DBMS process that periodically polls the sys_response_policy table, and verifies the signature on all policies.

Session key Verification

Whenever the DBA want to do any of the one the listed operation like create, activate, suspend, drop each time a session key generated by the pseudo random generator and send to the DBA. The DBA has to submit the key then only the rest of the DBA associated with the particular action will come to about the change. Likewise all the DBA associated with the policy have to give the approval that is k-1. Then only the change will apply. In these each time the session key changes and sends to the DBA via mail or phone number.

Malicious Policy Update

A policy may be modified by a malicious DBA using the SQL update statement. However, all policy definition attributes that need to be protected are hashed and signed; therefore any modification to such attributes through the SQL update command will invalidate the final signature on the policy.

Malicious Policy Deletion

An authorized policy may be deleted by a malicious DBA using the SQL delete command. However in JTAM, a policy tuple is never physically deleted; only its state is changed to DELETED. Thus, a signature on the policy-count can be used to detect malicious deletion of policy tuples. The detailed approach is as follows: When the Create Response Policy command is executed, the DBMS counts the number of policy tuples present in the database. It increments such policy-count by one to account for the new policy being created. A hash is taken on the new policy-count and state =VALID, and a signature share is generated on such hash. The signature share, policy id of the policy being created,the k value of the policy being created, and the initial state = INVALID are all stored in the sys_response_policy_count catalog as shown in Table 9. These values replace the tuple that is already present in the table. Note that the policy id that is inserted in the sys_response_policy_count table represents the latest policy that has been created. During policy activation, the DBMS first checks if the policy id present in sys_response_policy_count matches the id of the policy currently being activated. If the check succeeds, it counts the number of policy tuples in the database, and generates a signature share on the hash of the policy-count, and state = VALID. If the check fails, no signature share is generated. Such strategy ensures that always the correct policy-count is signed as multiple policies may be in CREATED stage at the same time. The final signature on the policy-count is generated when the (k – 1)th administrator activates the policy. The state of the policy-count signature is then changed to VALID. The dedicated DBMS process that verifies the individual policy signatures also verifies the signature on the policy-count. If a policy tuple is deleted, the signature on the policy-count is invalidated.

IV. Policy Matching

In this section, we present our algorithms for finding the set of policies matching an anomaly. Such search is executed by matching the attributes of the anomaly assessment with the conditions in the policies

Base Policy Matching

The policy matching algorithm is invoked when the response engine receives an anomaly detection assessment. For every attribute A in the anomaly assessment, the algorithm evaluates the predicates defined on A. After evaluating a predicate, the algorithm visits all the policy nodes connected to the evaluated predicate node. If the predicate evaluates to true, the algorithm increments the predicate-match-count of the connected policy nodes by one. A policy is matched when its predicate-match-count becomes equal to the number of predicates in the policy condition. On the other hand, if the predicate evaluates to false, the algorithm marks the connected policy nodes as invalidated. For every invalidated policy, the algorithm decrements the policy-match-count9 of the connected predicates; the rationale is that a predicate need not be evaluated if its policy-matchcount reaches zero.

Ordered Policy Matching

The search procedure in the base policy matching algorithm does not go through the predicates according to a fixed order. We introduce a heuristic by which the predicates are evaluated in descending order of their policy-count; the policy-count of a predicate being the number of policies that the predicate belongs to. We refer to such heuristic as the Ordered Policy Matching algorithm. The rational behind the ordered policy matching algorithm is that choosing the correct order of predicates is important as it may lead to an early termination of the policy search procedure either by invalidating all the policies or by exhausting all the predicates. Note that the sorting of the predicates in decreasing order of their policy-count is a pre-computation step which is not performed during the runtime of the policy matching procedure.

Response Action Selection

© 2013, IJARCSSE All Rights Reserved

In the event of multiple policies matching an anomaly, we must provide for a resolution scheme to determine the response to be issued. We propose the following two rank based selection options that are based on the severity level of the response actions:

1. Most Severe Policy (MSP).

The severity level of a response policy is determined by the highest severity level of its response action. This strategy selects the most severe policy from the set of matching policies. Note that the response actions described previous sections are categorized according to their severity levels. Also, in the case of interactive ECA response policies, the severity of the policy is taken as the severity level of the Failure Action.

2. Least Severe Policy (LSP).

This strategy, unlike the MSP strategy, selects the least severe policy. In our implementation, we provide the DBA with an option to switch between the two choices.

Query limitation

When ever the user try to access the databases the decrement counter set by the DBA associated with the table verifies the maximum limit simply said to be as the usage. That particular usage still have the allocations left to use that user request will proceed further . if the user have no allocations left or complete the limit means the request will not be proceeded further .the user has to wait until the DBA again gives the approval.

In such case the DBA will get an alert about the user's requirement. After that the DBA will analyze the requirement given by the user. Then issue a suitable response to the user after the approval of k-1 DBA it come in to usage.

V. Conclusion And Future Work

In this paper, we have described the response component of our intrusion detection system for a DBMS. The response component is responsible for issuing a suitable response to an anomalous user request. We proposed the notion of database response policies for specifying appropriate response actions. We presented an interactive Event-Condition-Action type response policy language that makes it very easy for the database security administrator to specify appropriate response actions for different circumstances depending upon the nature of the anomalous request. The two main issues that we addressed in the context of such response policies are policy matching, and policy administration. For the policy matching procedure, we described algorithms to efficiently search the policy database for policies matching an anomalous request assessment. We extended the PostgreSQL open-source DBMS to implement our methods. Specifically, we added support for new system catalogs to hold policy related data, implemented new SQL commands for the policy administration tasks, and integrated the policy matching code with the query processing subsystem of PostgreSQL. The experimental evaluation of our policy matching algorithms showed that our techniques are efficient. The other issue that we addressed is the administration of response policies to prevent malicious modifications to policy objects from legitimate users. We proposed a JTAM, a novel administration model, based on Shoup's threshold cryptographic signature scheme. We presented the design and the implementation details of JTAM, and reported experimental results on the efficiency of the policy signature verification mechanism.

References

[1] A. Kamra and E. Bertino, "Design and Implementation of Intrusion Response system,"

[2] A. Kamra, E. Terzi, and E. Bertino, "Detecting Anomalous Access Patterns in Relational Databases," J. Very Large DataBases (VLDB), vol. 17, no. 5, pp. 1063-1077, 2008.

[3] A. Kamra, E. Bertino, and R.V. Nehme, " Responding to Anomalous Database Requests," Secure Data Management, pp. 50- 66, Springer, 2008.

[4] A. Kamra and E. Bertino, "Design and Implementation of SAACS: A State-Aware Access Control System," Proc. Ann. Computer Security Applications Conf. (ACSAC), 2009.

[5] R. Mogull, "Top Five Steps to Prevent Data Loss Leaks. Gartner Research (July 2006)," http://www.gartner.com, 2010.

[6] A. Conry-Murray, "The Threat from within. Network Computing(Aug.2005),"http://www.networkcomputing.com/showArticle. jhtml?articleID=166400792, July 2009.

[7] F. Fabret, F. Llirbat, J.A. Pereira, I. Rocquencourt, and D. Shasha, "Efficient Matching for Content-Based Publish/Subscribe Systems,"technical report, INRIA, 2000.

[8] M.K. Aguilera, R.E. Strom, D.C. Sturman, M. Astley, and T.D. Chandra, "Matching Events in a Content-Based Subscription System," Proc. Symp. Principles of Distributed Computing (PODC), pp. 53-61, 1999.

[9] A. Conry-Murray, "The Threat from within. Network Computing (Aug. 2005)," http://www.networkcomputing.com/showArticle. jhtml?articleID=166400792, July 2009.

[10] J. Widom and S. Ceri, Active Database Systems: Triggers and Rules for Advanced Database Processing. Morgan Kaufmann, 1995.

[11] R. Gennaro, T. Rabin, S. Jarecki, and H. Krawczyk, "Robust and Efficient Sharing of RSA Functions," J. Cryptology, vol. 20, no. 3, pp. 393-400, 2007.

© 2013, IJARCSSE All Rights Reserved

[12] V. Shoup, "Practical Threshold Signatures," Proc. Int'l Conf. Theory and Application of Cryptographic Techniques (EUROCRYPT), pp. 207-220, 2000.
[13] C.K. Koc, "High-Speed RSA Implementation," Technical Reporttr-201, Version 2.0, RSA Laboratories, 1994.
[14] E.N. Hanson, M. Chaabouni, C.-H. Kim, and Y.-W. Wang, "A Predicate Matching Algorithm for Database Rule Systems," Proc. ACM SIGMOD, vol. 19, no. 2, pp. 271-280, 1990.
[15] J.A. Pereira, F. Fabret, F. Llirbat, and D. Shasha, "Efficient Matching for Web-Based Publish/Subscribe Systems," Proc. Int'l Conf. Cooperative Information Systems (CooplS), pp. 162-173, 2000.
[16] V. Ganapathy, T. Jaeger, and S. Jha, "Retrofitting Legacy Code for Authorization Policy Enforcement," Proc. IEEE Symp. Security and Privacy, pp. 214-229, 2006.
[17] H.-S. Lim, J.-G. Lee, M.-J. Lee, K.-Y. Whang, and I.-Y. Song,"Continuous Query Processing in Data Streams Using Duality of Data and Queries," Proc. ACM SIGMOD, pp. 313-324, 2006.

© 2013, IJARCSSE All Rights Reserved