International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.2, Issue.1, Jan-Feb 2012 pp-412-416 ISSN: 2249-6645

INTRUSION RESPONSE SYSTEM TO AVOID ANOMALOUS REQUEST IN RDBMS

Akila.L[1], Mrs.DeviSelvam[2]

[1]II M.E CSE,Sri shakthi Institute Of Engineering and Technology,Anna University,Coimbatore
[2]Asst.prof CSE,Sri shakthi Institute Of Engineering and Technology,Anna University,Coimbatore

Abstract:

The intrusion response component of an overall intrusion detection system is responsible for issuing a suitable response to an anomalous request. In the existing system, Intrusion Detection mechanism consists of two main elements, specifically tailored to a DBMS: anomaly detection (AD) system and an anomaly response system. In anomaly response system conservative actions, fine-grained actions, and aggressive actions methods are used. The proposed system mainly concentrates on response policies by using policy matching and policy administration. For the policy matching problem, we propose two algorithms that efficiently search the policy database for policies that match an anomalous request. We also extend the PostgreSQL DBMS with our policy matching mechanism, and report experimental results. The other issue that we address is that of administration of response policies to prevent malicious modifications to policy objects from legitimate users. We propose a novel Joint Threshold Administration Model (JTAM) that is based on the principle of separation of duty. The key idea in JTAM is that a policy object is jointly administered by at least k database administrator (DBAs), that is, any modification made to a policy object will be invalid unless it has been authorized by at least k DBAs out of L. We present design details of JTAM which is based on a cryptographic threshold signature scheme, and show how JTAM prevents malicious modifications to policy objects from authorized users.

Index Terms: PostgreSQL DBMS, Joint Threshold Administration Model, Threshold signatures.

I. INTRODUCTION

Our approach to an ID mechanism consists of two main elements, specifically tailored to a DBMS: an anomalydetection (AD) system and an anomaly response system.The first element is based on the construction of databaseaccess profiles of roles and users, and on the use of such profiles for theADtask.Auser-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 for additional information and experimental results. The second element of our approach—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. 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 user.

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[4], and search through the policy database to find the matching policies. We implement our policy matching scheme in the PostgreSQL DBMS [7], and discuss relevant implementation issues. We also report experimental results that show that our techniques are very efficient.

II.AN OVERVIEW OF RELATED WORK

Administration model is based on the well known security principle of separation of duties (SoD). SoD is a principle whereby multiple users are required in order to complete a given task. As a security principle, the primary objective of SoD is prevention of fraud (insider threats), and user generated errors. Such objective is traditionally achieved by dividing the task and its associated privileges among multiple users.

However, the approach of using privilege dissemination is not applicable to our case as we assume the DBAs to possess all possible privileges in the DBMS. Our approach instead applies the technique of threshold cryptography signatures to achieve SoD. A DBA authorizes a policy operation, such as create or drop, by submitting a signature share on the policy. At least k signature shares are required to form a valid final signature on a policy, where k is a threshold parameter defined for each policy at the time of policy creation. The final signature is then validated either periodically or upon policy usage to detect any malicious modifications to the policies.

The key idea in our approach is that a policy operation is invalid unless it has been authorized by at least k DBAs. We thus refer to our administration model as the Joint Threshold Administration Model (JTAM) for managing response policy objects. To the best of our knowledge, ours is the only work proposing such administration model in the context of management of DBMS objects.

The three main advantages of JTAM are as follows: First, it requires no changes to the existing access control mechanisms of a DBMS for achieving SoD. Second, the final signature on a policy is nonrepudiable, thus making the DBAs accountable for authorizing a policy operation. Third, and probably the most important, JTAM allows an organization to utilize existing man-power resources to address the problem of insider threats since it is no longer required to employ additional users as policy administrators.

III. CREATION OF INTRUSION RESPONSE SYSTEM

The main contributions of this paper can be summarized as follows:

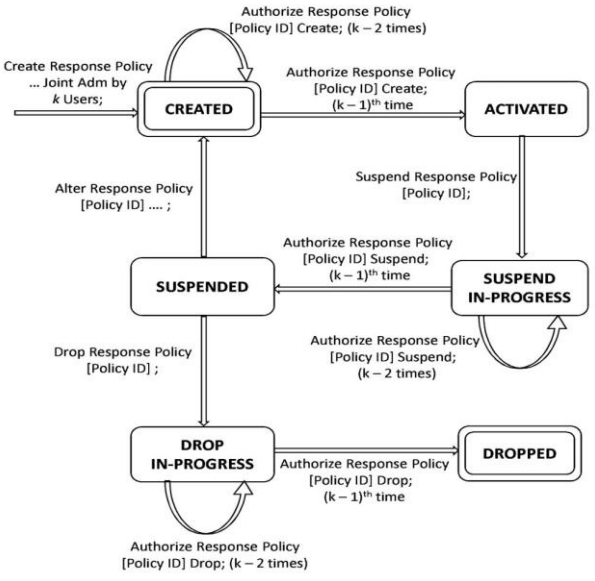

Fig .1. Policy state transition diagram.

1. We present a framework for specifying intrusion response policies in the context of a DBMS.
2. We present a novel administration model called JTAM for administration of response policies.
3. We present algorithms to efficiently search the policy database for policies that match an anomalous request.
4. We extend the PostgreSQL DBMS with our response policy mechanism, and conduct an experimental evaluation of our techniques.

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 ACTIVATED. A policy in an ACTIVATED state is operational, that is, it is considered by the policy matching procedure in its search for matching policies. If a policy needs to be altered, dropped or made nonoperational, it must be moved to the SUSPENDED state. The transition from the ACTIVATED state to the SUSPENDED state must also be authorized by k _ 1 administrators, before which the policy is in the SUSPEND IN-PROGRESS state. Note that a policy in the SUSPEND IN-PROGRESS state is also considered to be operational. From the SUSPENDED state, a policy can be either moved back to the CREATED state or it can be moved to the DROPPED state. A single administrator can move a policy to the CREATED state from the SUSPENDED state, while a policy drop operation must be authorized by k _ 1 administrators (before which the policy is in the DROP IN-PROGRESS state). We begin our detailed discussion of a policy object's lifecycle with the policy creation procedure.

OVERALL PROCESS OF INTRUSION RESPONSE SYSTEM

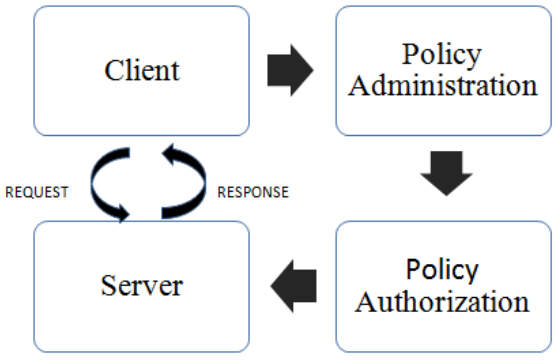

Fig .2.Flow of process

Client requests to the server for performing their operations in the database. For that, the server does the following policy methods like policy administration, policy authorization to find whether the client is intruder or not. After performing the policy methods, the server responds to the client. By the way the intruders are found. The working principle of each components in the fig.2 are explained below,

A.SERVER

We address in the context of such response policies are that of policy matching and policy Administration 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.

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.

Interactive response policy language makes it very easy for the database administrators to specify appropriate response actions for different circumstances depending upon the nature of the anomalous request. The two main issues that we address in context of such response policies are that of policy matching, and policy administration. 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 for additional information and experimental results.

B.POLICY ADMINISTRATION

An administration model referred to as the JTAM. The threat scenario that we assume is that a DBA has all the privileges in the DBMS, and thus it is able to execute arbitrary SQL insert, update, and delete commands to make malicious modifications to the policies. Such actions are possible even if the policies are stored in the system catalogs.3 JTAM protects a response policy against malicious modifications by maintaining a digital signature on the policy definition. The signature is then validated either periodically or upon policy usage to verify the integrity of the policy definition.

JTAM is that we do not assume the DBMS to be in possession of a secret key for verifying the integrity of policies. If the DBMS had possessed such key, it could simply create a HMAC (Hashed Message Authentication Code) of each policy using its secret key, and later use the same key to verify the integrity of the policy.

C.POLICY AUTHORIZATION

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[1].

A DBA authorizes a policy operation, such as create or drop, by submitting a signature share on the policy. At least k signature shares are required to form a valid final signature on a policy, where k is a threshold parameter defined for each policy at the time of policy creation.

The final signature is then validated either periodically or upon policy usage to detect any malicious modifications to the policies. The key idea in our approach is that a policy operation is invalid unless it has been authorized by at least k DBAs. We thus refer to our administration model as the Joint Threshold Administration Model (JTAM) for managing response policy objects[3].

It requires no changes to the existing access control mechanisms of a DBMS for achieving SoD. Second, the final signature on a policy is non reputable, thus making the DBAs accountable for authorizing a policy operation. Third, and probably the most important, JTAM allows an organization to utilize existing man-power resources to address the problem of insider threats since it is no longer required to employ additional users as policy administrators.

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. 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 dropping the request, disconnecting the user or revoking/denying the necessary privileges.

Signature

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[8].

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.

D.CLIENT

Often clients and servers communicate over a computer network on separate hardware, but both client and server may reside in the same system. A server machine is a host that is running one or more server programs which share their resources with clients. A client does not share any of its resources, but requests a server's content or service function. Clients therefore initiate communication sessions with servers which await incoming requests.Note that implementing the confirmation actions such as a re authentication or a second factor of authentication require changes to the communication protocol between the database client and the server. The scenarios in which such confirmation actions may be useful are when a malicious subject (user/process) is able to bypass the initial authentication mechanism of the DBMS due to software vulnerabilities (such as buffer overflow) or due to social engineering attacks (such as using someone else's unlocked unattended terminal).

Interactive response with the user is not required; the confirmation/resolution/failure actions may be omitted from the policy.

IV. PERFORMANCE EVALUATION

We perform three sets of experiments. The first two experiments report and compare the overhead of the policy matching algorithms. The third experiment reportsresults on the overhead of the signature verification mechanism in JTAM.

In the first experiment, the anomaly assessment is set such that the number of matching policies for an anomaly is kept constant at four. The number of predicates, and correspondingly the number of policies, are varied in order to assess the policy matching overhead time. Fig. 1 shows the policy matching overhead for the two algorithms as a function of the number of predicates.
Fig. 2 reports the number of predicates skipped as a function of the number of predicates. As expected, the policy matching overhead time increases linearly with the increase in the number of predicates in the policy database. Interestingly, the number of predicates skipped in both the algorithms is almost same.

Thus, counter-intuitively, the ordered policy matching algorithm does not lead to a decrease in the number of predicate evaluations. In fact, for larger number of predicates, the policy matching overhead of the ordered predicate algorithm is higher than that of the base policy matching algorithm.

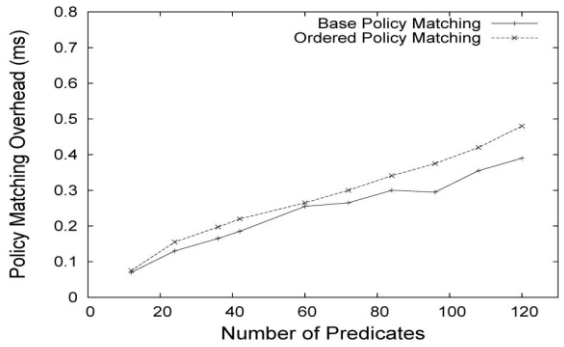

Fig 1. Experiment 1: Number of predicates versus policy matching overhead.

Such increase in matching overhead may be explained by the fact that the predicates evaluated by the ordered policy matching are more computationally expensive than the ones evaluated by the base policy matching algorithm. The key observation from this experiment, however, is that predicate ordering based on the policy-count parameter has no benefits in terms of decreasing the overhead of the policy matching procedure.

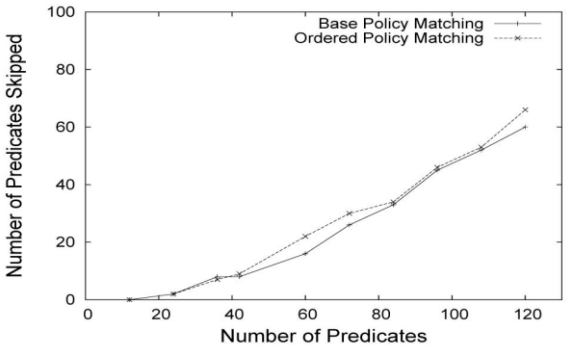

Fig. 2. Experiment 1: Number of predicates versus number of predicates skipped.

In the second experiment, we keep the number of predicates in the policy database constant at 60. The number of policies is also kept constant at 20. The number of matching policies is varied in order to assess the policy matching overhead. Fig. 3 shows the policy matching overhead for the two algorithms as a function of the number of matching policies. As expected, the policymatching overhead increases with the increase in the number of matching policies.

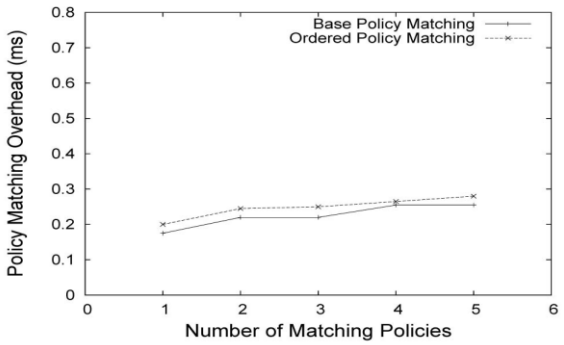

Fig 3. Experiment 2: Number of matching policies versus policy matching overhead

Moreover, in this experiment as well, the overhead of the ordered policy matching algorithm is higher than that of the base policy matching algorithm.

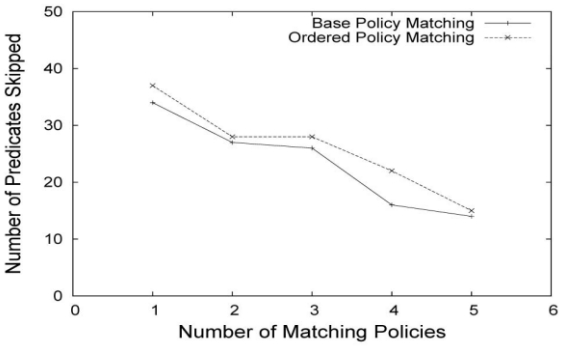

Fig. 4 Experiment 2: Number of matching policies versus number of predicates skipped

Fig. 4 reports the variation in the number of predicates skipped by varying the number of matching policies. For both the algorithms, the number of predicates skipped by the search procedure decreases for increasing numbers of matching policies. Such result is expected since an increase in the number of matching policies leads to an increasing number of predicate evaluations.

Overall, the fist two experiments confirm the low overhead associated with our policy matching algorithms.

They also show that predicate ordering based on the descending policy-count parameter has no significant impact on reducing the overhead of the policy matching procedure.

Therefore, a better strategy is to create a dedicated DBMS process that periodically polls the policy tables, and verifies the signature on all the policies.

V. CONCLUSION

In this paper, we have described the response component of our intrusion detection system for a DBMS. 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. 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 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 are currently in the process of implementing the intrusion detection algorithms in the PostgreSQL DBMS as part of our overall intrusion detection and response system in a DBMS.

REFERENCES

[1] 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.

[2] Campailla, S. Chaki, E. Clarke, S. Jha, and H. Veith, "EfficientFiltering in Publish-Subscribe Systems Using Binary Decision Diagrams," roc. Int'l Conf. Software Eng. (ICSE), pp. 443-452, 2001.

[3] V. Ganapathy, T. Jaeger, and S. Jha, "Retrofitting Legacy Code for Authorization Policy Enforcement," Proc. IEEE Symp. Security and Privacy, pp. 214-229, 2006.

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

[5] 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.

[6] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone, Handbook of Applied Cryptography. CRC Press, 2001.

[7] " Postgresql 8 . 3. The Postgresql Global Development Group " http://www.postgresql.org/, July 2008.

[8] V. Shoup, "Practical Threshold Signatures," Proc. Int'l Conf. Theory and Application of Cryptographic Techniques (EURO1RYPT), pp. 207- 220, 2000.