# CHAPTER 16
# DISCOVERY OF MULTI-LEVEL SECURITY POLICIES

Christina Yip Chung, Michael Gertz, Karl Levitt
*Department of Computer Science, University of California, Davis, CA 95616, U.S.A.*
{chungy|gertz|levitt}@cs.ucdavis.edu

**Abstract** With the increasing complexity and dynamics of database systems, it becomes more and more difficult for administrative personnel to identify, specify and enforce security policies that govern against the misuse of data. Often security policies are not known, too imprecise or simply have been disabled because of changing requirements.

Recently several proposals have been made to use data mining techniques to discover profiles and anomalous user behavior from audit logs. These approaches, however, are often too fine-grained in that they compute too many rules to be useful for an administrator in implementing appropriate security enforcing mechanisms.

In this paper we present a novel approach to discover security policies from audit logs. The approach is based on using multiple concept hierarchies that specify properties of objects and data at different levels of abstraction and thus can embed useful domain knowledge. A profiler, attached to the information system's auditing component, utilizes such concept hierarchies to compute profiles at different levels of granularity, guided by the administrator through the specification of an interestingness measure. The computed profiles can be translated into security policies and existing policies can be verified against the profiles.

## 1. INTRODUCTION

A major obstacle in securing todays information systems is not the lack of appropriate security mechanisms (see, e.g., [4] for an overview of access control models), but the lack of methods and concepts that allow security administrators to identify security policies. This is because in practice often only a very few policies are known at system design-time. Furthermore, at system run-time existing security policies typically need to be modified or new policies need to be added. In such cases, determining appropriate policy modifications is based on the normal behavior of users. What is needed are concepts and tools that help administrators in identifying security policies of interest and in verifying existing security policies.

Recently several proposals have been made to discover user profiles from audit logs (see, e.g., [2, 7, 6, 9, 12, 13, 14]). Profiles describe the normal behavior of users (groups) regarding the usage of the system and associated applications. Discovered profiles can be used to derive specifications of security enforcing mechanisms based on, e.g., an access control model. The ability to incorporate application specific domain knowledge is critical to the success and usability of these approaches. In [7, 6] we have shown how misuse detection, based on data mining techniques, can benefit from domain knowledge that is incorporated into applications associated with an information system. However, even these approaches turn out to generate too many fine-grained policies. Administrators typically do not want to deal with hundreds of closely related access patterns, but prefer a representation of access patterns at an abstract, more generalized level of description.

In this paper we describe an approach that tries to alleviate this shortcoming by discovering security policies at different levels of detail. The approach is applicable to database systems as well as Web-based information systems. We employ *concept hierarchies* [3] that describe properties (values) of data at different levels of granularity. Concepts modeled in such hierarchies build descriptive parts of user profiles and thus security policies at different levels of detail. Such hierarchies are either provided by the administrator (thus representing some kind of domain knowledge) or can be discovered from data using clustering techniques (see, e.g., [8, 10]). We propose an extension of the concept hierarchy framework in which we organize feature/value pairs (representing concepts) in trees. Our framework is more general than [11] in that it does not impose an arbitrary partial order on the attributes (concepts) in a concept hierarchy.

Our profiler is capable of considering multiple concept hierarchies in discovering profiles. Multiple concept hierarchies have been introduced in the data mining domain for deriving typical patterns of data at different levels of abstraction [3, 10, 11]. We extend the usage of multiple concept hierarchies by allowing different types of concepts in a single hierarchy. Depending on the security policies to discover or verify, the administrator can choose among (combinations of) concept hierarchies and abstract concepts (features) embedded in these hierarchies.

Finally, we introduce the notion of *interestingness measure* as an important means for administrators to guide the profile and policy discovery process. This measurement can be specified by the administrator depending on the type and granularity of policy she is interested in. Our approach provides a valuable tool for *security re-engineering* which utilizes audit logs generated by a database system at run-time.

# 2. CONCEPT HIERARCHIES

Concept hierarchies represent application specific domain knowledge about features of interest for the discovery of security policies. Starting from auditable features, a concept hierarchy can be developed bottom-up by the administrator or data clustering techniques.

In Section 2.1, we describe how concept hierarchies organize feature/value pairs in form of trees. In Section 2.2, we discuss a model to determine whether a given itemset, consisting of feature/value pairs, is of interest. Section 2.3 outlines the usage of concept hierarchies for generalizing itemsets.

## 2.1. PRELIMINARIES

Objects and data are conceptualized through feature/value pairs in which a feature is an object property and value is the value of the property. A feature/value pair is denoted by $\langle F = f \rangle$, meaning that the value of feature $F$ is $f$. We use trees to model concept hierarchies organizing objects and data through feature/value pairs at different levels of abstraction. Each node in the tree corresponds to a feature/value pair. The root of the tree is a special node including all concepts.

While such a framework for a concept hierarchy is simple and intuitive, it captures many real world examples. It is worth mentioning that a concept hierarchy is not limited to only one type of feature. Features can be abstracted to other features. Figure 1 shows a concept hierarchy for the concept `time`, a feature that often is of interest in profiling.

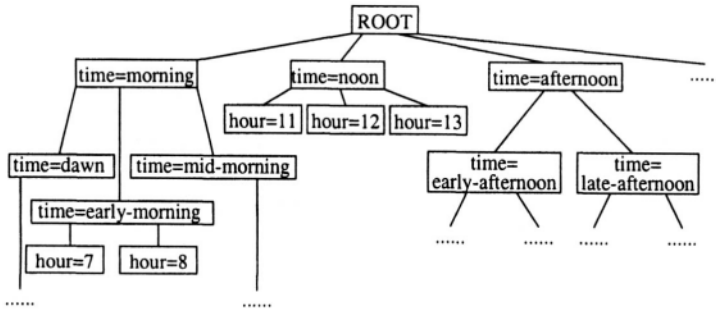


*Figure 1.* Concept Hierarchy on Time

Note that leaf nodes represent auditable features ("raw data" in the audit log), whereas inner nodes represent generalizations of these features at different levels. Formally, a concept hierarchy is a tree where nodes are labeled with feature/value pairs. A subtree with root $\langle F = f \rangle$, having $n$ subtrees rooted with $\langle F_i = f_i \rangle, i = 1 \ldots n$, is denoted by $T(F = f) := (\langle F = f \rangle, T(\langle F_1 = f_1 \rangle), \ldots, T(\langle F_n = f_n \rangle))$.

Let $\mathcal{T}$ be a set of concept hierarchies. *Desc*($n$) denotes the set of nodes that are descendants of node $n$ in $T \in \mathcal{T}$. The depth of a node $n$ in $T$, denoted by *Depth*($n$, $T$), is the number of nodes on the path from the root to $n$. The depth of the root node is 0. The depth of $T$, denoted by *Depth*($T$), is the maximum depth of all nodes in the tree.

We say a feature/value pair $\langle F_i = f_i \rangle$ may be generalized to $\langle F_j = f_j \rangle$ with respect to a hierarchy $T$ if $\langle F_j = f_j \rangle$ is an ancestor of $\langle F_i = f_i \rangle$ in $T$. Since $F_j$ may be different from $F_i$, a feature may be generalized to another feature, e.g., a different type of concept, as indicated in Figure 1, where the feature hour has been generalized to the feature `time`.

## 2.2. INTERESTINGNESS MEASURE

For computing user profiles based on concept hierarchies, it is important to determine whether a set of feature/value pairs might be of interest, i.e., is likely to lead to useful information regarding the usage patterns an administrator is interested in. As a novelty to using concept hierarchies, we introduce the notion of *interestingness measure* to guide the discovery of interesting patterns in an audit session *Audit* containing a set of itemsets. We consider four aspects, (1) *support*, (2) *depth*, (3) *distance*, and (4) *size* of itemsets describing sets of feature/value pairs.

**(1) Support** The support of an itemset gives a measure of how frequent that itemset occurs in an audit session. The higher the support of an itemset, the more regular is the pattern in the audit session. Let *Audit* be an audit session and $I = \{\langle F_1 = f_1 \rangle, \ldots, \langle F_n = f_n \rangle\}$ an itemset. An itemset $I'$ in *Audit satisfies* $I$, denoted by $I \leq I'$, if

$\forall \langle F = f \rangle \in I : \exists \langle F' = f' \rangle \in I' : \langle F' = f' \rangle = \langle F = f \rangle$ **or** $\langle F = f \rangle \in Desc(\langle F' = f' \rangle)$.

The support of an itemset $I$ in *Audit*, denoted by *Sup*($I$), is the number of itemsets in *Audit* that satisfy $I$, normalized by the size of *Audit*:

$$Sup(I) := \frac{|\{I \mid I \leq I' \in Audit\}|}{|Audit|} \in \mathbb{R}[0,1]$$

**(2) Depth** We prefer feature/value pairs of lower level of abstraction since they are more specialized and convey more detailed information. This is captured by the depth of an itemset and describes the depths of its feature/value pairs in a set of concept hierarchies. Let $\mathcal{T}_{\langle F=f \rangle}$ be a set of concept hierarchies containing the feature/value pair $\langle F = f \rangle$. Then

$$Depth(\langle F = f \rangle) := \begin{cases} \frac{1}{|\mathcal{T}_{\langle F=f \rangle}|} \times \Sigma_{T \in \mathcal{T}_{\langle F=f \rangle}} \frac{Depth(\langle F=f \rangle, T)}{Depth(T)} & \text{if } \mathcal{T}_{\langle F=f \rangle} \neq \{\} \\ 1 & \text{otherwise} \end{cases}$$

Intuitively, the depth of a feature/value pair $\langle F = f \rangle$ is its average depth among all concept hierarchies that contain $\langle F = f \rangle$ as a node. The depth of an itemset $I$ is the average depths of its feature/value pairs, i.e.

$$Depth(I) := \frac{1}{|I|} \times \Sigma_{\langle F=f \rangle \in I} Depth(\langle F = f \rangle), \quad \in \mathbb{R}[0, 1]$$

**(3) Distance** We conjecture that usage patterns of users typically involve feature/value pairs that are semantically related. That is, users typically access information that is semantically related. Whether and how information is related is some kind of domain specific knowledge that often can be obtained from the information structures underlying the application, in particular the information schemas.

For example, in Web-based information systems, features such as IP address (called **src**), URL (**rsc**) of a requested page, and time of request (**time**) can often be organized into one or more hierarchies. For example, the feature/value pair $\langle$**src = www.bases.mil**$\rangle$ can be considered as the parent of the feature/value pair $\langle$**src = www.ca.bases.mil**$\rangle$ and so on.

The distance between two feature/value pairs can be defined based on their location in a concept hierarchy. Consider, for example, the feature/value pairs $\mathbf{fv_1} \equiv \langle$**rsc = /user/scott/doc**$\rangle$, $\mathbf{fv_2} \equiv \langle$**rsc = /user/scott/schedule**$\rangle$, and $\mathbf{fv_3} \equiv \langle$**rsc = /user/jones/reports**$\rangle$. $\mathbf{fv_1}$ and $\mathbf{fv_2}$ are more related than $\mathbf{fv_1}$ and $\mathbf{fv_3}$ or $\mathbf{fv_2}$ and $\mathbf{fv_3}$ because $\mathbf{fv_1}$ and $\mathbf{fv_2}$ share a longer common path. Thus, the distance measure between two feature/value pairs in a concept hierarchy $T$ can be defined as

$$Dist(\langle F = f \rangle, \langle F' = f' \rangle, T) := 1 - \frac{Depth(LCA(\langle F = f \rangle, \langle F' = f' \rangle, T))}{Depth(T)}$$

where the function $LCA(\langle F = f \rangle, \langle F' = f' \rangle, T)$ gives the least common ancestor of two nodes $\langle F = f \rangle$, $\langle F' = f' \rangle$ in a concept hierarchy $T$. The higher the depth of the least common ancestor of two feature/value pairs, the smaller is the distance measure, i.e. they are more related.

A similar notion of distance measure can be devised for queries against relations and attributes in relational databases (see [5]).

**(4) Size** We prefer itemsets that contain more feature/value pairs since they reveal correlated information between different feature/value pairs. This is reflected by the size component of the interestingness measure, i.e., the number of feature/value pairs the itemset $I$ contains.

In sum, for the meaningful discovery of itemsets (and thus profiles and policies) we prefer itemsets that are more regular, more specialized, semantically closer and contain more feature/value pairs. Thus, the interestingness measure M($I$) of an itemset $I$ should (1) increase with its support, depth, size and (2) decrease with its distance. A simple

formulation of $M(I)$ that satisfies these criteria thus is

$$M(I) := (Sup(I) + Depth(I) + 1 - Dist(I) + Size(I)) \ / \ 4$$

## 2.3. GENERALIZATION

An important feature of using concept hierarchies in policy discovery is the usage of the abstraction mechanism embedded in the hierarchies. For example, using concept hierarchies, access patterns of user can be generalized to access patterns of user groups, accessed objects can be generalized to classes of objects, depending on their attribute values, and so on. A feature/value pair $\langle F = f \rangle$ in an itemset $I$ can be *generalized* to $\langle F' = f' \rangle$ if it is replaced by $\langle F' = f' \rangle$ in the itemset $I$ and $\langle F' = f' \rangle$ is an ancestor of $\langle F = f \rangle$ in some concept hierarchy. The support and depth components of the interestingness measure consider the effect of generalizing an itemset in all concept hierarchies. We incorporate multiple concept hierarchies into the discovery process by using the change of interestingness measure as a criteria to decide whether an itemset should be generalized.

There are two opposing forces in deciding whether $\langle F = f \rangle$ should be generalized to $\langle F' = f' \rangle$. On one hand, more general itemsets have higher support and hence a higher interestingness measure. On the other hand, it loses depth and hence drives down its interestingness measure.

Let $I'$ be the generalized itemset of $I$ after generalizing $\langle F = f \rangle$ to $\langle F' = f' \rangle$ with respect to some concept hierarchy. The change of interestingness measure is reflected by the change of its four components support, depth, distance, and size.

$$\Delta Sup(I \rightarrow I') := Sup(I') - Sup(I) \qquad \Delta Depth(I \rightarrow I') := Depth(I') - Depth(I)$$
$$\Delta Dist(I \rightarrow I') := Dist(I') - Dist(I) \qquad \Delta Size(I \rightarrow I') := Size(I') - Size(I)$$

The change of interestingness measure of $I$ to $I'$, denoted by $\Delta M(I \rightarrow I')$, then is defined as:

$$(\Delta Sup(I \rightarrow I') + \Delta Depth(I \rightarrow I') + \Delta Dist(I \rightarrow I') + \Delta Size(I \rightarrow I')) \ / \ 4$$

As a general guideline for the discovery of profiles, an itemset $I$ should be generalized to an itemset $I'$ if there is a gain in its change of interestingness, i.e. if $\Delta M(I \rightarrow I') > 0$.

## 3. ALGORITHM

In this section we outline the algorithm underlying the computation of itemsets using concept hierarchies in a profiler. In Section 3.1, we present the basic underlying data structures for managing itemsets. In Section 3.2, we describe the algorithm for computing interesting itemsets.

## 3.1. DATA STRUCTURES

The auditing component associated with a database system records audit data in an audit log implemented as relations in a database. Attributes of the audit log are the features audited and their values. Hence, a tuple in the audit log can be viewed as a set of feature/value pairs, called *itemset*, and a relation as a set of itemsets. Features to be audited by the system are determined by the administrator prior to the user-profiling and policy discovery. The administrator groups audit records into *audit sessions* based on security policy she is interested in. Each audit session consists of audit records sharing some property. For example, an audit session might contain all audit records that correspond to audited actions performed by a particular user. The profiler computes a profile for each audit session.

In the following, we assume the following relations for recording audit and features/value data. The relation `Audit (L1,l1,...,Ln,ln)` stores sets of feature/value pairs (itemsets). Each itemset corresponds to an audit record from an audit session. The relations `Lk(FIID, interest, sup, depth, dist, size, A1,a1,...,Ak,ak)` record itemsets $I = \{\langle A1 = a1\rangle,\ldots,\langle Ak = ak\rangle\}$ up to size k and their interestingness measure (`interest`). Values for the attribute `interest` are based on the components `sup[port]`, `depth`, `dist[ance]` and `size` (see Section 2.2).

Relation `F(FIID, A, v)` stores all interesting itemsets discovered from tuples in relations `L1 ... Ln`. Finally, relation `FMaster(FIID, size, interest,sup,depth,dist)` stores the interestingness measure information about the itemsets in `F`. To relate itemsets in these two relations, each itemset is given a unique identifier, named `FIID`.

## 3.2. BASIC ALGORITHM

The method to compute interesting itemsets based on multiple concept hierarchies is divided into 6 steps, as described below

```
for k = 1 to n do (let n be the total number of features)
  Step 0: check if we need to continue
          if k>1 and Lk is empty then break end if
  Step 1: Generate & store itemsets of size k from itemsets of size k-1
          Lk <- generateltemsets(k)
  Step 2: Compute interestingness measure for each tuple in relation Lk
  Step 3: Generalize itemsets based on concept hierarchies
          for each feature/value pair afv in a set of hierarchies do
            generalizeToAFV(afv.Lk)
  Step 4: Delete non-interesting itemsets from Lk
  Step 5: Record interesting itemsets in relations F and FMaster
end do;
Step 6: Prune non-minimal itemsets
```

Readers are referred to [7] for more details regarding steps 2, 4, 5 and 6. Here we will only briefly describe Step 3, which extends our previous approach by generalizing itemsets in Lk for possible feature/value pairs in concept hierarchies. The order of applying generalizeToAFV to a feature/value pair afv is unspecified.

Procedure generalizeToAFV(afv, Lk) generalizes itemsets in Lk to feature/value pair afv (which stands for AncestorFeatureValue pair) if there is a gain in interestingness measure. First, all itemsets in Lk that consist of descendants of afv are copied to a temporary table Lgen. These feature/value pairs are replaced by their ancestor afv with the change in interestingness measure computed. Second, itemsets with a gain in interestingness measure are generalized in Lk by updating them according to Lgen. Since two or more feature/value pairs in an itemset may be generalized to afv, care is taken to remove redundant pairs from the generalized itemset. If afv is the root of the tree, the feature/value pair is dropped. Third, after Lk is generalized, there may be redundant itemsets which should be pruned. Redundant itemsets in Lk with FIID in FIIDpair are deleted except one itemset. If the feature/value pairs of an itemset are generalized to the root node, the itemset is empty and deleted. In our prototype profiler, all the above computations are performed on tables using SQL statements and stored procedures.

# 4. APPLICATION TO SECURITY

In this section, we describe a feasibility study on extending the profiler by concept hierarchies and compare its effectiveness with a profiler not employing concept hierarchies. We then give some guidelines on how to convert profiles to policies and their usage in detecting misuse. More details can be found in [5].

## 4.1. FEASIBILITY STUDY SETUP

We show the usefulness of our profiler, called *Profiler CH* (CH stands for Concept Hierarchy), by running it over a web audit log gathered at an institution offering online courses. Further, we illustrate the difference of the profiles generated by using another profiler, called *ProfilerNoCH*, which is based on the same technique, but does not consider concept hierarchies. The web audit log records the access patterns of users over four consecutive days. We consider three features in the audit log, src (IP address of host requesting a web page), rsc (URL of requested page) and hour (time page has been requested, see also Fig. 1).

There are two audit sessions. The session AuditSG contains audit records whose feature src corresponds to the domain(s) *.sg and rsc to

pages under `/course/*`. Another session `AuditUIUC` consists of records whose feature values for `src` satisfy `*.uiuc.edu` and values for `rsc` correspond to `/spep-95/*`. Due to space limitation, we only show the profile discovered by ProfilerCH for audit session `AuditSG`:

| FIID | Interest | Sup | Depth | Dist | Size | A1 a1 | A2 a2 |
|---|---|---|---|---|---|---|---|
| 2-10 | 0.51 | 1.00 | 0.21 | 1.00 | 2 | rsc /course/ | src sg |
| 1-27 | 0.51 | 0.46 | 0.40 | 0.00 | 1 | rsc /course/plb1/rice/ | |
| 3-1 | 0.50 | 1.00 | 0.14 | 1.00 | 1 | TIME day | |
| 2-1 | 0.47 | 0.79 | 0.21 | 1.00 | 2 | rsc /course/ | TIME day |
| 2-21 | 0.47 | 0.79 | 0.21 | 1.00 | 2 | src sg | TIME day |
| 1-1 | 0.47 | 0.39 | 0.30 | 0.00 | 1 | TIME noon | |
| 1-5 | 0.47 | 0.38 | 0.30 | 0.00 | 1 | TIME morning | |
| 1-57 | 0.45 | 0.21 | 0.36 | 0.00 | 1 | rsc /course/plb1/tomato/ | |
| 2-25 | 0.43 | 0.54 | 0.26 | 1.00 | 2 | rsc /course/plb1/ | TIME day |
| 1-11 | 0.43 | 0.14 | 0.36 | 0.00 | 1 | rsc /course/cant/current/ | |

## 4.2. DISCUSSION

We evaluate and compare the profilers based on the following criteria: (1) *Coverage*: Can the profiler discover criteria to group audit records into audit sessions? (2) *Simplicity*: How many itemsets are generated? (3) *Novelty*: Can the profiler discover new patterns which are not specified by the criteria that groups audit records into audit sessions?

**Coverage** ProfilerNoCH discovers that requests often come from `src=ce.singnet.com.sg` and `src=po.pacific.net.sg`. The other selection criteria (values for `rsc` are under `/course/*`) are also covered by those itemsets involving `rsc=/course/*`. However, the correlation that `rsc-/course/*` is accessed from `src=*.sg` is not discovered. This is because the data is sparse. Itemsets involving both `rsc` and `src` do not have high enough support and are pruned. This is, however, discovered by ProfilerCH as reflected in itemset 2-10. ProfilerCH discovers the selection criteria because data are aggregated into a higher level of abstraction based on the concept hierarchies. This makes the pattern more prominent and hence is discovered. In sum, both profilers can discover the patterns used to select the data, whereas ProfilerCH does a better job.

**Simplicity** There is a total of 74 itemsets discovered by ProfilerNoCH. Those patterns are represented by only 11 itemsets discovered by ProfilerCH, which is a significant reduction. The concise profile discovered by ProfilerCH helps the administrator in better understanding usage patterns of the user group `*.sg (singapore)`

**Novelty** ProfilerNoCH discovers patterns that were not expected, namely users from user group `singapore` often access pages at specific hours. Similarly, ProfilerCH discovers patterns that were unknown be-

fore. For example, the hours of access are often TIME=noon, morning (covered by TIME=day) and TIME=midnight. Hence, the administrator can add a new security policy stating that users from the group singapore are allowed to access the web pages during the aforementioned times. Alternatively, if someone from group singapore suddenly accesses the web pages at hour=evening, this may be not normal.

Further, ProfilerCH discovers that the resources that are most often accessed are actually /course/plb1/[rice/tomato] and /course/cant/current/. This can aid the administrator in refining the security policy that users from group singapore can access resources /course/ to a finer detail. Other novel patterns discovered include that users from group singapore often access the web pages during day time (TIME=day) and that pages accessed are likely to be /course/plb1/*.

**Inter-User Group Behavior** The conjecture that users from different user groups exhibit different usage behavior is confirmed by comparing the profiles discovered by ProfilerCH on AuditSG and AuditUIUC. A comparison of the two profiles shows that users from group uiuc often access web pages during night (including midnight and late-night), afternoon and late-evening as opposed to day time by users from group singapore. This can be explained by the time zone difference between the US and Singapore.

## 4.3. MAPPING PROFILES TO POLICIES

We consider a policy to be an if-then statement describing a characteristic of a session, i.e., an interesting itemset in the profile. Here we describe how to map an interesting itemset to a policy.

The criteria that aggregates audit records into an audit session gives the precondition of the policy. Feature/value pairs in the interesting itemset are literals in the consequent combined by 'and's. A policy is refined in the following ways: (1) Care is taken to remove literals in the consequent that are logically implied by the precondition to avoid trivial true policies. (2) A policy is simplified by removing literals in the consequent that are descendants of some other literals in concept hierarchies. (3) A refined policy can be obtained by replacing literals in the precondition by those in the consequent that logically imply them. (4) Policies with the same precondition can be aggregated into a single policy by combining their consequents by 'or's. Based on these guidelines, we can derive the following policies for session AuditSG:

```
if src=*.sg and rsc=/course/* then rsc=/course/plb1/rice/*
        or rsc=/course/plb1/tomato/* or rsc=/course/cant/current/*
if src=*.sg and rsc=/course/* then TIME=day or TIME=midnight
if src=*.sg and rsc=/course/plb1/* then TIME=day
```

Anomalies can be detected by comparing the policies derived from a new session with the policies of the corresponding profile(s). A policy from a new session whose precondition matches some policies, but its consequent matches none of these policies is a violating policy. Violating policies represent behavior deviating from normal. A policy from a new session whose precondition matches none of the profile policies is a new policy. New policies signal new usage and a high ratio of violating and new policies in a new session indicates possible misuse of the system.

# 5. CONCLUSIONS AND FUTURE WORK

In this paper we presented an important practical extension to existing approaches for discovering user access patterns and security policies from audit logs associated with diverse types of information systems. We strongly believe that for todays complex databases and information systems, it is vital to employ sophisticated concepts and tools that help administrators in discovering and verifying security policies. Such tools and concepts will play a major role in security (re-)engineering as part of the administration of such complex systems.

We extended existing approaches to user profiling and policy discovery in several ways. We use concept hierarchies that allow administrators to embed domain specific knowledge at different levels of abstraction. The discovery of access patterns and security policies, based on data mining techniques, takes multiple such hierarchies into account, allowing to relate different concepts of interest in the search for interesting patterns. In particular, by introducing the notion of interestingness measure, which considers data semantics, the presented profiler is capable of discovering interesting data access patterns at the right level of abstraction. By considering multiple concept hierarchies describing different features at a time, more complex and non-trivial profiles and policies can be discovered.

We have presented a simple formulation of interestingness measure that satisfy certain criteria based on our experience with different types of audit logs and applications. One direction of future research is to explore other formulations of the interestingness measure, and also to incorporate other aspects of domain knowledge available from applications. Currently, the administrator is responsible for choosing relative weights of the components of the interestingness measure. It would be useful if the system can automatically adjust and fine tune the parameters. We demonstrated the effectiveness of our profiler by running it over a web server access log. We also outlined guidelines on how discovered profiles can be translated into policies.

We also plan to automate the translation of profiles to policies and enforcing mechanisms. For example, in case of relational database systems, profiles can be converted into either appropriate user roles (with associated profiles) or user/application-specific views on data.

## References

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th VLDB Conference*, 487–499, Morgan Kaufmann, 1994.

[2] R. Bueschkes, M. Borning, D. Kesdogan. Transaction-based anomaly detection. In *Proc. of the Workshop on Intrusion Detection & Network Monitoring*, 1999.

[3] Y. Cai, N. Cercone, and J. Han, Attribute-oriented induction in relational databases. In *Knowledge Discovery in Databases*, 213–228. AAAI/MIT Press, 1991.

[4] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1995.

[5] C.Y. Chung, M. Gertz, and K. Levitt. Discovery of multi-level security policies. Technical Report, Department of Computer Science, University of California, Davis, `http://www.db.cs.ucdavis.edu/publications/CGL00a.ps`

[6] C.Y. Chung, M. Gertz, and K. Levitt. DEMIDS: A misuse detection system for database systems. In *Third International IFIP TC-11 WG11.5 Working Conf. on Integrity and Internal Control in Information Systems*, 159–178, Kluwer, 1999.

[7] C.Y. Chung, M. Gertz, and K. Levitt. Misuse detection in database systems through user-profiling. In *2nd Int. Workshop on Recent Advances in Intrusion Detection (RAID'99)*, West Lafayette, Indiana, 1999.

[8] B. Everitt. *Cluster Analysis.* John Wiley & Sons - New York, 1973.

[9] T. Fawcett and F. Provost. Combining data mining and machine learning for effective user profiling. In *The Second International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 8–13, 1996.

[10] J. Han and Y. Fu. Dynamic Generation and Refinement of Concept Hierarchies for Knowledge Discovery in Databases. *AAAI'94 Workshop on Knowledge Discovery in Databases*, 157–168, July 1994.

[11] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. *Proc. of Int. Conf. on Very Large Data Bases*, 420–431, 1995.

[12] W. Lee, S.J. Stolfo, and K.W. Mok. Mining audit data to build intrusion detection models. In *Proc. of the 14th International Conf. on Knowledge Discovery and Data Mining (KDD-98)*, 66–72. AAAI Press, 1998.

[13] R. Mukkamala, J. Gagnon, and S. Jajodia. Integrating data mining techniques with intrusion detection methods. In *Proc. XIII Annual IFIP WG 11.3 Working Conf. On Database Security*, Seattle, WA, July 1999.

[14] R.S. Silken. Application intrusion detection. Technical Report CS-99-17, University of Virginia, Computer Science Department, June 1999.