CHAPTER 26

ALGORITHMS AND EXPERIENCE IN INCREASING THE INTELLIGIBILITY AND HYGIENE OF ACCESS CONTROL IN LARGE ORGANIZATIONS

Marc Donner - Morgan Stanley
David Nochlin - Morgan Stanley
Dennis Shasha - New York University
Wendy Walasek - Morgan Stanley

Abstract: Security managers in large organizations must manage the access of tens of thousands of employees on diverse data. Databases store the access control information but the security officers use essentially manual techniques to determine who has too much access and why. We call this task the Audit Problem. The security research community has offered promising frameworks such as role-based access control, but these still leave open the problems of designing the roles and determining group memberships and of demonstrating that there are substantial benefits to be reaped from making a change.

In this paper, we propose a data-mining approach that includes an algorithm that starts with a set of atomic permissions of the form (user, asset, privilege) and derives a smaller but equivalent set (user group, asset group, privilege group). The asset and privilege groups so identified constitute promising roles. The users so identified constitute useful groups.

In this paper we report on actual experience with actual corporate access control data. We built a production role-based access control authorization service, storing the tables in a relational database and transmitting queries as XML 'documents' over MQ message queues.

Our experiments show that the proposed algorithm can reduce the number of permission assertions by a factor of between 10 and 100. With such a reduction, the Audit Problem is brought from the absurd to the plausible.

1. THE PROBLEM

1.1 The players

In any complex environment, there are three sets of players in the access control arena. On the one hand there are the business managers who are constrained by fiduciary responsibility and by specific regulations to establish an appropriate control regime for information. On the other hand there are the software implementers and support staff for whom success is measured by the rapid deployment of powerful software and by its stable operation. Between the two is a collection of auditors, both internal and external, whose role is to ensure that the implementations are at least approximately compliant with the regulations.

These relationships are illustrated in this figure:

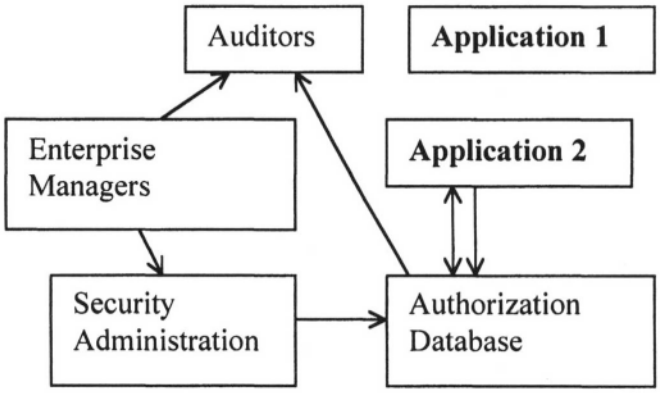


Figure 1.

1.1.1 Enterprise Managers [Policy]

To senior enterprise managers the imperatives of security are risk management and regulatory compliance. How, at minimal cost, can they maintain an acceptable risk profile and comply with the various laws, rules, and regulations governing the enterprise? Enterprise managers know that complexity breeds failure, and so they want clear simple statements of principle. An ideal policy is a one or two page statement in English that unambiguously lays out the goals and objectives and establishes clear guidance to employees seeking decisions.

1.1.2 Software Developers and Integrators [Application 1 and Application 2]

Developers and integrators are typically faced with demands for rapid delivery of software and systems that provide the maximum in functionality and performance at the lowest cost. Most software is produced or procured for a specialized purpose and serves a subset of the organization that is, by nature of being smaller, naturally more coherent. This reduces the perceived need by both the enterprise line staff and the developers for stringent security machinery.

1.1.3 Security Administrators [Security Administration]

Low in the organizational food chain and often neglected in planning access control systems, the security administrators are the keys to the day-to-day interpretation of the security policies and their effective reduction to practice. They have to understand the policies, they need tools that they can use effectively, and their needs must be reflected in the overall access control architecture if it is to be effective. To illustrate this point, imagine a house with excellent locks on every window and door, but installed so inconveniently that to lock or unlock them requires that the homeowner carry a ladder and a box of tools to each window and door. How likely is it that the doors and windows will be locked or unlocked at the appropriate times?

1.1.4 Auditors

Senior managers and a wide variety of external parties, from customers to industry regulators, look to the auditors to reconcile the ideally clear and simple statements of policy and the necessarily complex and messy implementations that emerge from day-to-day activities. Auditors, in turn, look for powerful analytical tools to help them reduce the complexity of the world they discover so that they may answer the questions that are put to them.

1.1.5 Related Work

To quote from a report on this problem at NIST [1], "one of the most challenging problems in managing large networked systems is the complexity of security administration. Today, security administration is costly and prone to error because administrators usually specify access control lists for each user on the system individually. The principle of role based access control (RBAC) is that security should be mapped to an organization's structure. With Role-based Access Control, security is managed at a level that corresponds closely to the organization's structure. Each user is assigned one or more roles, and each role is assigned one or more privileges that are permitted to users in that role."

The notion of authorization hierarchies using roles is certainly present in the SQL databases [2].

In SQL, the owner of a resource can give certain operational rights (e.g. insert, delete, and update) to certain users or groups of users (called roles) on certain objects where an object can be a column, table, or entire database, or even can be defined as a query. The syntax is:

```
GRANT privileges
ON objects
TO roles
```

Non-SQL approaches to the problem include the product from Computer Associates called TopSecret. That system allows positive and negative authorizations, permitting grouping of users into collections, wild cards in object names, and a hierarchy of privileges NONE > ALL > WRITE > READ where > means "takes precedence over". Assets and permissions are grouped into profiles that are distributed to users. Finally, there is a notion of single and variable length wild card characters, so that for example AB*C matches ABXYTC.

Jajoida et alia [3] have proposed a framework for positive and negative access control policies. The framework consists of languages and techniques for making positive and negative access control policies consistent.

The novelty of our work derives mainly from the fact that we must actually solve this problem on large data. In a large bank that originates significant electronic funds transfers people who have excessive permissions could cause significant financial harm. Existing approaches within our organization to reducing such risks are manual and consist of interviewing people and managers about their needs. Our approach here is to aid that work by increasing the intelligibility of the permissions people have.

Why could we define no previous experimental study of access control or algorithms to increase the intelligibility of security? Here's what Prof. Jajodia observed in a personal communication: "There is no study because the general feeling is that groups and roles are helpful. Everybody accepts that." In a way, this paper is meant to show how much help such a technique can provide.

2. A DATA MODEL

2.1 Glossary

Term	Definition
Asset	An ***asset*** is an enterprise thing like an account or a technical thing like an application program or system file.
Asset group	An asset ***group*** is a potentially arbitrary administrative grouping of ***assets***. The groupings are established for enterprise purposes, for instance a set of accounts can be grouped together because they are owned by the same party or are controlled by the same portfolio manager. An asset ***group*** is an ***asset***, so these definitions support potentially arbitrary grouping and clustering.
Person/User [People]	A ***person*** or ***user*** here is a bona fide individual. Each system assigns each person with which it has a relationship a unique ID called the ***userid***. Notice that the establishment of confidence that a particular person should be entitled to act as a particular userid is the responsibility of the ***authentication system***, an external component that provides that service uniformly to all systems within the organization.
User group	A ***user group*** is a potentially arbitrary grouping of ***people***. The groupings are established for

	enterprise purposes, for instance the set of ***people*** authorized to order trades in an account might be such a group.
Privilege	A ***privilege*** is the right to perform some action. In object oriented programming terminology, a ***privilege*** is the right to invoke a specific method. A method might enable you to read data from a specific file or set of files.
Privilege group	A ***privilege group*** is a named collection of ***privileges***.

2.2 The Central Data Structure

Historically, access control databases have provided the capability to assign specific individuals specific rights to specific assets. Some have provided the ability to aggregate one or another of these atomic entities, but in general little aggregation has been available. This has resulted in administrative and analytic headaches as the number of assertions in the access control databases have grown. At Morgan Stanley Dean Witter in

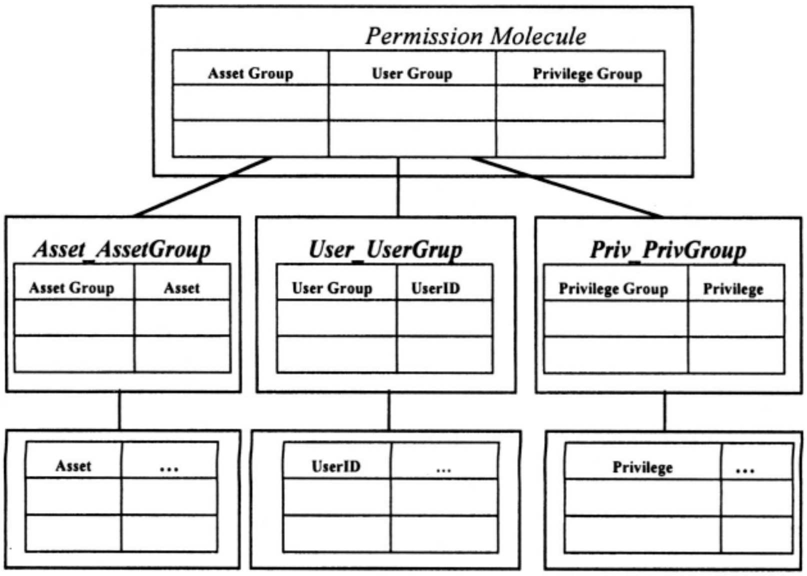


Figure 2 [User_UserGroup and (p,r) is in Priv_PrivGroup]

the Institutional Securities business we use a product named "Top Secret," now from Computer Associates, which permits the grouping of asset-privilege combinations. The operational database for our Top Secret environment has nearly 500,000 assertions. The number is so large that we cannot practically analyze it.

It was our hypothesis when we undertook this work that by providing aggregation for each of the three entities, users, assets, and privileges, we could reduce the number of assertions in the central fact table. As you will see in the result section below, that is exactly what we found.

3. ALGORITHMS AND PROOFS

The data model introduced in section 2 is a framework for grouping assets into asset groups, users into user groups, and privileges into privilege groups. The algorithm of this section transforms a set of atomic relationships into the most economical possible set of groupings.

3.1 Reduction Algorithm for Inferring Molecular Rules:

Input: A set of rules at the atomic level (asset, user, and privilege). This is called **PermissionsAtom.**

Output: A set of rules at the molecular level (asset group, user group, and privilege group). This is called **PermissionsMolecule.**

A description of the groupings

- From asset to asset group (**Asset_AssetGroup**),
- From user to user group (**User_UserGroup**), and
- From privilege to privilege group (**Priv_PrivGroup**).

Correctness Conditions:

If (a, u, p) is in PermissionsAtom, then there exists a row (c, g, r) in PermissionsMolecule such that (a,c) is in Asset_AssetGroup, (u,g) is in User_UserGroup, and (p,r) is in Priv_PrivGroup.

3.1.1

If (c,g,r) is in PermissionsMolecule then for all a, u, p such that (a,c) is in Asset_AssetGroup, (u,g) is in User_UserGroup, and (p,r) is in Priv_PrivGroup, (a, u, p) is in PermissionsAtom.

3.2 Atom to Molecule Reduction Algorithm:

We explain this algorithm along with the following running example. We start with the following atomic table.

Table 1. [Atom to Molecule Reduction Algorithm]

Asset	User	Privilege
a1	u2	p1
a1	u3	p2
a2	u1	p2
a2	u1	p1

Note: We now present the reduction steps in a particular order. As we show later, the order of reduction matters, so this ordering is only one among several (six, to be precise). But all reductions use the same basic construction.

3.2.1 Asset Reduction Step

For each distinct user-privilege pair (u, p), find the set of assets:

```
S(u,p) = {   PermissionsAtom.asset |
             PermissionsAtom.User = u and
             PermissionsAtom.Privilege = p
         }
```

For each such set, create a new name, e.g. assetgroup_u_p. Then create a new table, denoted tab1, consisting of these asset groups and their unique user-privilege pairs. Formally, tab1 has three columns (asset group, user, and privilege), where user-privilege together form a key.

In set theoretic notation:

```
tab1 (asset group, user, priv) =
 { ( S( u, p), u, p) |
   u belongs to PermissionsAtom.user and
   p belongs to PermissionsAtom.priv
 }
```

In words: create triples consisting of each possible user u, privilege p, and the associated set of assets, which will be grouped into a new asset group name. In addition create an association between each created set of assets and an asset group name.

In our running example, this would give the following (here, we keep the original sets instead of replacing them by asset group names):

Table 2.

tab1

Asset Group	User	Privilege
{a1, a2}	u1	p1
{a1}	u2	p1
{a1}	u3	p2
{a2}	u1	p2

Asset	Asset Group
a1	c1
a2	c1
a1	c2
a2	c4

Thus, c2 is the name for {a1} whereas c1 is the name for {a1, a2}. So, replacing the sets by their names, we would get another form for tab1:

Table 3.

tab1

Asset Group	User	Privilege
c1	u1	p1
c2	u2	p1
c2	u3	p2
c3	u1	p2

3.2.2 Privilege Reduction Step

The next step consists of grouping the privileges into privilege groups. The strategy is to perform groupings based on user and asset group to create sets of privileges.

```
tab2(asset group, user, privilege group) =
    { ( c, u, { tab1.priv |
                    tab 1.user = u and
                    tab1.asset_group = c
                }
    ) |
        u belongs to tab1.user and
        c belongs to tab1.asset_group
    }
```

Again, the constraint is that no two rows in tab2 may have the same asset group and user combination of values.

In our running example, this does not diminish the number of rows:

Table 4.
tab2

Asset Group	User	Privilege Group
{a1, a2}	u1	{p1}
{a1}	u2	{p1}
{a1}	u3	{p2}
{a2}	u1	{p2}

3.2.3 User Reduction Step

The next step consists of grouping users into groups. This is the final step and creates the molecular level of grouping that we seek.

```
    PermissionsMolecule(
            asset group,
            user group,
            privilege group) =
             { ( c,   { tab2.user |

    tab2.privilege_group = r and
    tab2.asset_group = c
       },r) |
    r belongs to tab2.privilege_group and
    c belongs to tab2.asset_group
  }
```

In our running example, this does not diminish the number of rows either:

***Table 5.* [*PermissionsMolecule*]**

Asset group	User group	Privilege Group
{a1, a2}	{u1}	{p1}
{a1}	{u2}	{p1}
{a1}	{u3}	{p2}
{a2}	{u1}	{p2}

3.3 Observations:

3.3.1

The general step of reducing based on C given a table T(G1, G2, C) is to form the set

```
T'   ={ ( g1, g2, { T.C |
          T.G1 =g1 and
          T.G2 = g2
        }
     )|
     g1 belongs to T.G1 and
     g2 belongs to T.G2
   }
```

with the constraint that no two rows in T' have the same g1 and g2 values, in combination. (So, many rows may have g1 in the G1 column and many may have g2 in the G2 column, but only one row will have g1 and g2.)

3.3.2

The transformation satisfies the correctness conditions. We will show this through the use of generalized tables in which there are sets as field values.

3.3.3 Definitions:

The **atomic permission table** has a single user, a single asset, and a single privilege in each row.

A **partly molecular table** has a set of users, a set of assets, and a set of privileges in each row. Any of these sets can be singleton, so even the atomic permission table can be modeled as a partly molecular table in which every field in every row is singleton. A partly molecular table is isomorphic to the intermediate tables in our construction. Each such table has atoms in each field, but some of those atoms represent sets, (e.g. an asset group represents a set of assets). Partly molecular tables are non-first-normal form representations of those sets.

Table 6.

Example:

The partly molecular table for the result of the first reduction is:

Asset	User	Privilege
{a1, a2}	{u1}	{p1}
{a1}	{u2}	{p1}
{a1}	{u3}	{p2}
{a2}	{u1}	{p2}

The **row expansion** of row r in a partly molecular table is the cross product of its sets.

The **expansion** of a partly molecular table is the union of its row expansions.

A **row one-column expansion** of row r on column C is a set of rows of size ||r.C||, each row having one value of r.C and the values contained r for the remaining columns.

The **one-column expansion** of a partly molecular table is the union of its row one-column expansions.

Table 7.

Example:

If we have a table:

Asset	User	Privilege
{a1, a2}	{u1, u3, u4}	{p1,p2}
{a1, a3, a4}	{u2, u4}	{p1}

Then the one-column expansion based on column User would give:

Asset	User	Privilege
{a1, a2}	{u1}	{p1,p2}

{a1, a2}	{u3}	{p1,p2}
{a1, a2}	{u4}	{p1,p2}
{a1, a3, a4}	{u2}	{p1}
{a1, a3, a4}	{u4}	{p1}

Table 8.

The **expansion** (to singleton sets) would give:

Asset	User	Privilege
{a1}	{u1}	{p1}
{a1}	{u1}	{p2}
{a2}	{u1}	{p1}
{a2}	{u1}	{p2}
{a1}	{u3}	{p1}
{a1}	{u3}	{p2}
{a2}	{u3}	{p1}
{a2}	{u3}	{p2}
{a1}	{u4}	{p1}
{a1}	{u4}	{p2}
{a2}	{u4}	{p1}
{a2}	{u4}	{p2}
{a1}	{u2}	{p1}
{a3}	{u2}	{p1}
{a4}	{u2}	{p1}
{a1}	{u4}	{p1}
{a3}	{u4}	{p1}
{a4}	{u4}	{p1}

Observation:

Any ordering of one-column expansions that includes all columns will yield the same set of rows as an expansion.

The operation **unsetting** removes the curly brackets when all sets are singleton. In the example, unsetting yields the following table:

Table 9.

Asset	User	Privilege
al	ul	pl
al	ul	p2
a2	ul	pl
a2	ul	p2
al	u3	pl
al	u3	p2
a2	u3	pl
a2	u3	p2
al	u4	pl
al	u4	p2
a2	u4	pl
a2	u4	p2
al	u2	pl
a3	u2	pl
a4	u2	pl
al	u4	pl
a3	u4	pl
a4	u4	pl

The operation **setting** is the inverse of **unsetting**.

Suppose that Ta is an atomic table and P is a partly molecular table. We say that P **conserves** Ta if Ta = the unsetting of the expansion of P. That is, up to reordering Ta is the same as the unsetting of the expansion of P.

Lemma: If P1 is a partly molecular table that conserves Ta and P2 is a reduction of P1, then P2 also conserves Ta.

Proof: Our strategy is to show that reduction is the inverse of expansion. (We assume three columns, though the same argument will apply to any number of columns greater than one.)

When doing a reduction from P1 to P2, we partition P1 by its grouping columns, e.g. user and asset, and compute the set in its grouped column, e.g. privilege, corresponding to each partition. Call the grouping columns G1 and G2 and the grouped column C. Without loss of generality, assume that G1 and G2 are the first two columns. In P2, no two rows have the same G1 and G2 values. Further, no two sets of rows have the same G1 and G2 values in P1 so we can treat the reduction that produces a single row of P2 independently from the reduction for any other row of P2.

Consider the row having values g1 in G1 and g2 in G2 and the corresponding rows in P1 having g1 and g2 as members. Let set c of items from column C be all the items from column C in rows having (g1, g2) as the entries in G1 and G2, respectively.

So, the row (g1, g2, c) in P2 is the reduction of the set of rows in P1 having g1 and g2 as their first two field values. A row one-column expansion of (g1, g2, c) based on C results in ||c|| rows all having g1 and g2 as their first two field values and having a different element in c in the third field of each row.

A one-column expansion based on C is the inverse of reduction based on C. This implies that P1 is the expansion of P2 based on C. Hence, the expansion of P2 has the same set of rows as the expansion of P1.

Therefore P2 conserves Ta because P1 conserves Ta.

Theorem: The molecular Permissions Table that results from the Reduction Algorithm conserves the atomic Permissions Table.

Proof: By induction on the column reductions and from the lemma.

Recursion on this procedure does not help.

It never helps, i.e. reduces the size of the permissions molecule, to do further grouping beyond the above construction. We explain why by concrete example, but the reader will be able to generalize the example.

Suppose that it were useful to group, say, privilege groups into bigger privilege groups. Then there would have to be two rows in PermissionsMolecule with values (g, c, r) and (g, c, r') where privilege groups r and r' are distinct but group g and asset group c are the same.

When privilege groups are created, they are created based on a partition of user-asset group pairs. Users from such pairs later are put into groups. Therefore, no two privilege groups can be associated with the same user and asset group or the same user group and asset group.

3.3.4 Choosing different orderings of reduction may help.

Recall our running example:

Asset	**User**	**Privilege**
a1	u1	p1
a1	u2	p1
a1	u3	p2

a2	u1	p2
a2	u1	p1

If we reduce asset first, we get the following groupings after the first step:

Asset group	User	Privilege
{a1, a2}	u1	p1
{a1}	u2	p1
{a1}	u3	p2
{a2}	u1	p2

None of the other reductions shrinks this set any further.
However, if we reduce user first, we get:

User group	Asset	Privilege
{u1, u2}	a1	p1
{u3}	a1	p2
{u1}	a2	p2
{u1}	a2	p1

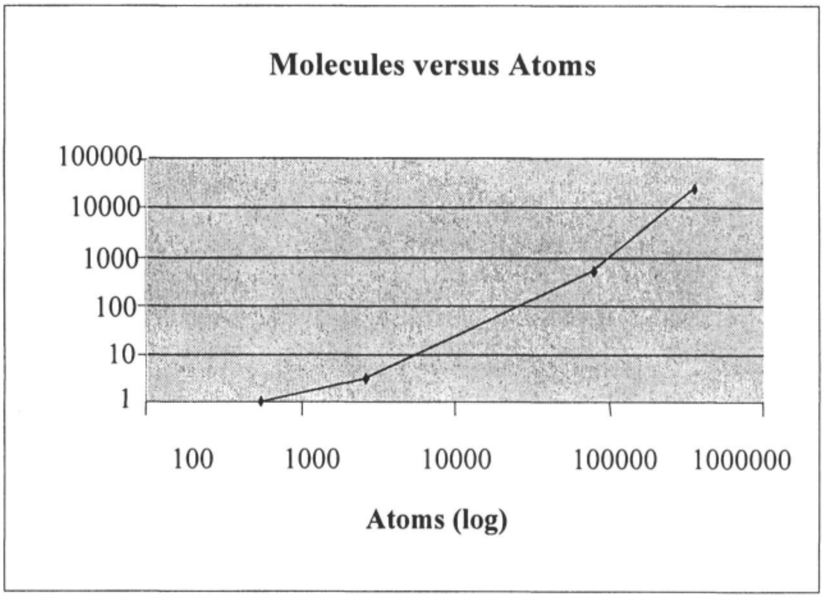
Now, if we reduce privilege next, we get:

User Group	Asset	Privilege Group
{u1, u2}	a1	{p1}
{u3}	a1	{p2}
{u1}	a2	{p1, p2}

This gives us three rather than four rows in the result.

4. QUANTITATIVE RESULTS

Reducing atomic relationships to molecular relationships seems to be a plausible idea, but does it really help? We set out to answer this question using several actual sets of authorization data in daily use within Morgan Stanley Dean Witter. The numbers we cite are:



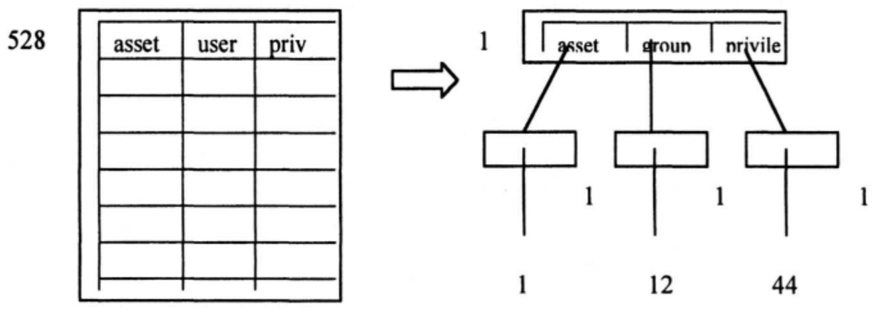
4.1.1 **the initial size of the atomic permissions table,**

4.1.2 **the size, following execution of the algorithm, of the molecular permissions table,**

4.1.3 **the sizes of each of the grouped entities: assets, userids, and privileges,**

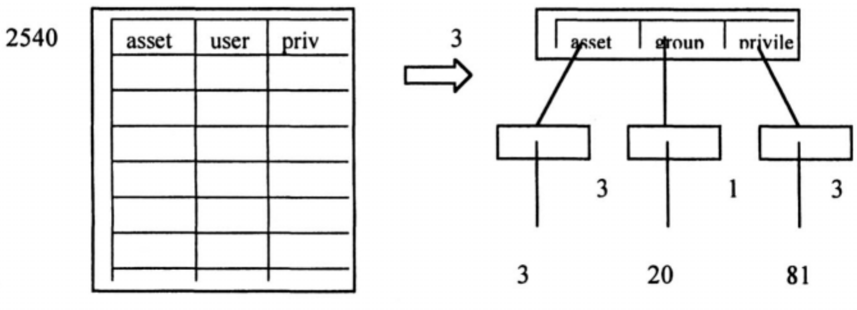
4.1.4 **the number of groups: asset groups, user groups, and privilege groups**

4.2 Experiment 1:



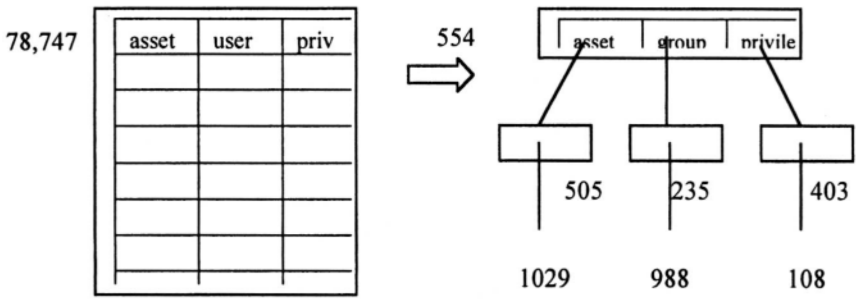
528 rows in PermissionsAtom became 1 row in Permissions Molecule.
- 1 asset grouped into 1 asset group
- 12 users grouped into 1 user group
- 44 privileges grouped into 1 privilege group

4.3 Experiment 2:



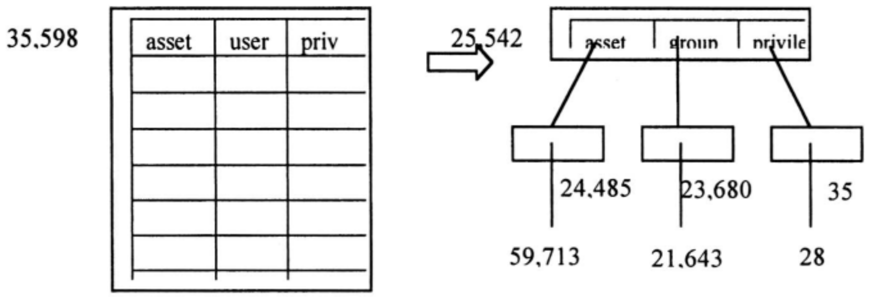
- 2540 rows in PermissionsAtom became
- 3 rows in PermissionsMolecule with
- 3 assets grouped into 3 asset groups
- 20 users grouped into 1 user group
- 81 privileges grouped into 3 privilege groups

4.4 Experiment 3:



- 78,747 rows in Permissions Atom became
- 554 rows in PermissionsMolecule with
- 1029 assets grouped into 505 asset groups
- 998 users grouped into 235 user groups
- 108 privileges grouped into 403 privilege groups

4.5 Experiment 4: Top Secret Configuration



- 354,598 rows in Permissions Atom became
- 25,542 rows in PermissionsMolecule with
- 59,713 assets grouped into 24,485 asset groups
- 21,643 users grouped into 23,680 user groups
- 28 privileges grouped into 35 privilege groups

5. FUTURE WORK

We began this article by highlighting the plight of the auditors who must reconcile the representation of a security policy with regulations. We suggested that presenting a human being with hundreds of thousands of permission facts (354,598 in our case) was simply too unwieldy. Our algorithm reduced this number of facts to 25,542, definitely a step in the right direction but not yet a solution.

Achieving a solution requires "improving the hygiene" of the authorization data, i.e. data cleaning to eliminate typos, reduce inconsistencies and so on. An algorithm for data cleaning is an open problem, but we think the following ideas are worth exploring:

Suppose that our grouping to the molecular level produces several sets of users that are large and are mutually similar. The fact that they are not identical might be an indication of inconsistent treatment of these users. The same holds for assets and for privileges.

Sometimes we may be able to infer that permissions are missing.

Consider the following example of atomic permissions:

A	B	C
a1	b1	c1
a1	b1	c2
a1	b2	c1
a1	b2	c2
a1	b3	c1
a1	b3	c2
a2	b1	c1
a2	b1	c2
a2	b2	c1
a2	b2	c2
a2	b3	c1
a2	b3	c2

As it stands this can be reduced to one row:

AG	BG	CG
{a1,a2}	{b1,b2,b3}	{c1,c2}

Now, we notice that any break in that symmetry, for instance by removing one row, will cause the groups to break up quite a lot. For instance, if we remove the last row to give:

A	B	C
al	bl	cl
al	bl	c2
al	b2	cl
al	b2	c2
al	b3	cl
al	b3	c2
a2	bl	cl
a2	bl	c2
a2	b2	cl
a2	b2	c2
a2	b3	cl

Then grouping will lead to more rows in the result:

AG	BG	CG
{al,a2}	{bl,b2,b3}	{cl}
{al,a2}	{bl,b2}	{c2}
{al}	{b3}	{c2}

In this case, we'd want to infer that we were missing:

a2	b3	c2

Additional information can also help the cleaning process. We have found the following heuristics to be useful for example when the organizational structure is supposed to imply homogeneous privileges: Find people whose permissions are inconsistent with most (say 80%) of their co-members in the supposedly homogeneous group. Alternatively, find permissions that most members of a group have and consider those to be a core. Question any permission outside the core.

Human judgement plays the critical role in managing security. Our hope is to provide tools that help people manage this sometimes-overwhelming task. This paper is one step in that direction.

6. REFERENCES

[1] "Role Based Access Control," D. Ferraiolo and R. Kuhn, NIST, http://hissa.ncsl.nist.gov/rbac/

[2] "Guide to the SQL Standard," H. Darwen and C. J. Date, 1997, Addison-Wesley, ISBN 0201964260.

[3] "A Unified Framework For Enforcing Multiple Access Control Policies," S. Jajoida, P. Samarati, V. S. Subrahmanian, and E. Bertino, Proceedings of ACM SIGMOD International Conference on Management of Data, 1997, pp 474-485