Collaborative Research: Models and Mechanisms for Data-Level Security

Chris Clifton (Purdue University) and Arnon Rosenthal (The MITRE Corporation)

Universal access to the WWW has made security a top issue. New problems have arisen in areas such as legally mandated privacy policies and protecting intellectual property. Collaborative arrangements need to be created and ended, rapidly. Systems architectures have become far more complex, so data requires protection in documents, messaging systems, and application servers. Most crucially, there are too few people skilled either in specifying policies, or in the technical means of enforcing them. We need solid new concepts to simplify this environment and to enable building new tools. A grand challenge for data security is to make large-scale administration easy enough that *ordinary* organizations will do it well.

The bulk of security research addresses system protection – analogous to an egg with a hard shell but few interior barriers. A common practice is that one protects *systems* from intrusion; middleware-approved processes have complete access to database data. Finer-grained mechanisms, if present, are frequently coded into applications. The end result is a state of affairs where:

- Systems have little or no capability to prevent insider misuse of information.
- The potential for unintentional corruption of data is high.
- Information access policies are specified in ways that are not automatically enforceable.
- Information-level access control is reimplemented for each application, with potential for errors at each stage.

Better access control mechanisms exist, e.g. view-based security in relational databases, but they are rarely used due to the high cost of specifying and administering policies with these mechanisms. Today, one would need an army of administrators, sophisticated both technically and in the policies that are to be enforced. Since such administrators are costly and scarce, little is done.

Maintaining appropriate fine-grained security is a significant obstacle to achieving a digital government. As government moves to provide greater direct access to individuals, the need for system-enforced access control grows. Eliminating the "government employee in the loop" between data and accessor greatly increases the opportunity for wide-scale violation of security policies. Enforcing controls at the application (or even middleware) level does not scale well – a change in policy may require changes to many applications, with access controls implemented using a hodge-podge of methods.

We propose a better approach: Information access policies are specified *once* – with respect to the information (data and operations) rather than in each stovepipe system. Policies are then enforced automatically across all objects that contain that information. For example, I may have authority to view and change my own address – I will be able to change it whether I access the database directly, through a third-party application, or even through delegating that authority to the US Postal Service with a change-of-address card. But none of those approaches would allow me to view or change someone else's address.

There are three key challenges to achieving this goal.

1. *Ensuring that the policy specification models capture access control needs.* Our primary interest is ensuring that the needs of a digital government (as exemplified by our collaboration with the Internal Revenue Service) are met; however meeting needs of a wider audience will increase the probability that such models become available in commercial products.
2. *Developing policy models that are useable*. Data administrators must specify policies using concepts and tools that they understand. Systems must also support policy administration. We will address single-point introduction/removal of controls, policy maintenance/update, and useful error semantics where policies are incorrect. For example, an access denial message "update failed" or "view is

empty" will not help correct policy problems; a better response is "update failed because of policy P3, administered by Joe". Because we want to develop modular capabilities, some other relevant issues (e.g., authentication, and replication) are outside our scope.

3. *Enforcing such policies using existing technology (or technology evolved from existing systems).* A policy model that can be enforced on existing standard SQL databases is more likely to succeed than one that requires new data and security models. An evolutionary approach – policy models that can be partially enforced on existing systems – may have immediate impact, provided other mechanisms can cleanly address the unenforced portions. Some policy rules may be infeasible to enforce on individual user actions (e.g., "do not email any information about an individual taxpayer's outside the agency") but are worth capturing so sanctions can be enforced, and actions audited.

The Internal Revenue Service has an extremely challenging security environment. Data ranges from high-integrity but non-sensitive (rules and regulations, forms) to highly sensitive (law enforcement actions). The number of users is huge – all taxpayers – and each has a different set of data they are allowed to access. There are also many types of users: IRS employees (trusted with most data, but access must be auditable to catch the occasional abuse), taxpayers, third parties such as paid tax preparers (to whom individuals may have delegated authority), other government agencies, etc. Different security policies must be specified and enforced for each class of user. The number of distinct applications is also large. In addition to internal systems, the IRS makes extensive use of private partners, such as companies offering electronic filing. Realizing the full potential of these partnerships demands a high degree of (secure) information interoperability – which currently demands too much administrative effort. We believe data security approaches that meet the needs of the IRS will meet the needs of most agencies building a digital government (with the possible exception of real-time processes, mobile computing, and data subject to military multi-level security policies –challenges we do not plan to address directly in this project.)

MITRE runs the Center for Strategic Tax Administration and Modernization (CSTAM), a nonprofit Federally Funded Research and Development Center (FFRDC) that is the IRS's major source of vendor-independent systems engineering help. This center assists with problems ranging from specification of security policies, to systems' architectures, to legacy migration. The PIs will work closely with MITRE staff supporting the IRS Office of Cyber Security. This will ensure that the technologies developed in this project are capable of expressing and enforcing the wide variety of security policies defined by the IRS. It also provides a technology transfer path directly to the IRS. The collocation and history of prior collaboration of the PIs and the security experts supporting the IRS ensures the project will maintain a connection to IRS security needs, without demanding that the IRS office of Cyber Security allocate its limited personnel to a long-term research project.

Motivating Examples

To illustrate the world we will work to enable, envision a hypothetical relationship by which the IRS might "outsource" a processing function (e.g., validation of certain corporate tax credits) to a contractor. How do we ensure that taxpayer information and contractor-proprietary information are kept secure, across all the possible interaction mechanisms (messaging, database access, web-based services, email, and so forth)? Specifying and enforcing policy from scratch for each collaboration is unreasonable. We will develop automated techniques to extend existing security policy to the new relationship, based on tool-friendly metadata that makes it easier to capture policy knowledge, to specify each party's obligations, and to manage the end-to-end process. For example:

- *Permissions inferred from explicit policy*. If existing policy says "tax court case law is available to the public", then portions of a particular corporate file that are also contained in a court case can certainly be released to the contractor. Many of these inferences can be invariant as the system implementation changes (e.g., to add a data warehouse). Others apply to physical computations, e.g., if you have the right to submit requests to two servers, then you have "request submission" capabilities needed to

compute a join of Public data on those servers.

- *Richer Types of Permissions*. Sometimes one wishes to grant privileges "with strings attached" to their usage or to granting it onward. One might grant a privilege to waive penalties only for error amounts below a threshold, and as part of an audited process. Other limitations might help express and enforce IRS obligations toward others, for taxpayer privacy or to protect law enforcement data. Finally, some actions may require a composite of several permission grants (e.g., to see corporate research filings, an auditor must be assigned to that corporation's case, and have signed a nondisclosure agreement).

- *Releasability.* Access controls are not sufficient – one needs to let government personnel know what is expected of them (e.g., not to release taxpayer data outside IRS, to release law enforcement data only to Law Enforcers with a documented need to know, and conversely, not to apply updates received by uncertified paths.) Administrators will not be willing to provide instructions about each derived product (e.g., case report), so we want to automatically infer the obligations that apply. This information can then be used to generate auditing or filtering rules, or warnings to the user that they appear to be violating policy.

- *Vulnerability tolerance*. Different situations require different balance between convenience and security. For trusted IRS analysts, it might suffice to have an official policy on data misuse, plus to audit unusual behaviors. Access by contractors might be restricted more securely, by DBMS access controls. To prevent one company from seeing another's filings, one might want defense in depth, e.g., DBMS access controls, plus permitting access only by a few carefully written functions. Finally, information about informants might be kept on a separate, more protected physical system.

- *Need for access*. Allowing greater access than is needed increases the opportunity for misuse of information. But assuming that currently granted privileges constitute the maximum tolerable means we may unnecessarily lose functionality. Most security models concern only controls and permissions; good management requires knowing the need for access.

Prior Work

Data security technology has not advanced rapidly in recent years. The research community is small, and has had little impact on data management products. The research advances that have made it into product have been spin-offs from the wider arena of system security. Below, we briefly discuss system security (relevant topics only), data security in general, and data security for distributed systems.

Systems' Security Overview

Operating systems have long been a focus of security research. Many ideas from systems security are used in data security (including our work). Operating systems basically support three mechanisms (with interactions among them):

- **Users and groups.** Each user is identified to the system, and may belong to one or more user groups.
- **Administrative roles.** Some users may be granted certain administrative rights – generally a fixed hierarchy, but sometimes more detailed and dynamic.
- **Access Control Lists.** Each object in the system (file / directory / program) has associated users and groups that may access it, along with the permissions each has on the object.

But system security is not enough. Most OS security work concentrates on providing a "hard shell" – only authorized users have access to the system. However, once inside the system, mechanisms for more detailed security enforcement are limited. There is no *defense in depth.* The difficulty is fundamental -- one can administer controls only on "objects" that are visible in the system model, e.g., files. Fine

granularity (e.g., columns) and content-based protections are unavailable. Early systems had strong security products (e.g., IBM's RACF), but these assumed centralized systems

Object and application servers can enforce security policies, using CORBA's framework for intercepting method calls, or using separate products (e.g., Netegrity's Site Minder) to rule on requests' appropriateness. These products emphasize interception and passing information around, but do not provide high level models for administration. Policies are code fragments, either in the vendor's proprietary language or in a full programming language such as Java. Neither leads to easy administration. While standards initiatives exist (e.g., A2ML), there is no standard with penetration comparable to SQL.

Document management raises many of the same security issues as databases. [Gladney97] contains excellent descriptions of requirements and of implementation techniques. Much of the attention goes to efficiency of run-time enforcement (physical data structures and algorithms); we expect instead to build over what DBMSs provide.

Data Security Research

A grand challenge for data security is to make large-scale administration easy enough that *ordinary* organizations will do it well. We are far from that point, today, and the goal is receding as our systems increase in complexity.

General Database Security
Data security research has produced rigorous theory for multilevel secure systems (e.g., [Winslett94]), and good syntheses of the issues in large systems. But researchers' impact has fallen far short of our impact on query processing, transactions, indexing, foreign functions, and many other areas. Security papers constituted roughly 2% of the 2001 submissions to the leading database conferences (SIGMOD and VLDB). No regular research papers were accepted (though one PI's panel on defining a new security research agenda was accepted).

SQL provides a strong standard that is widely implemented and understood. Unlike many other models, it formalizes how administrative authority is passed ("with grant option") and revoked (to maintain a custodial chain leading to each permission). But the SQL security model has had no conceptual extensions in the past 20 years, except for the recent addition of role-based access controls [Sandhu99] (which originated outside databases), plus mechanisms for cooperating with other servers (e.g., authentication from an LDAP directory).

There are specific areas of SQL that require extension. SQL supports almost no inference of permissions among tables (the one exception being for a view's owner). As a result, applications must be written explicitly in terms of tables where the anticipated user has permission – excessively early binding. (An alternative, also only partly satisfying, is to give permissions to the application-writer and have her delegate).

Over the years, the SQL standard has accreted low level details to provide security on new SQL constructs (e.g., objects in SQL99). The standard is specified as detailed semantics of individual operations, and partly as a result is so complex that it can be a barrier to further progress. DBMSs suffer similarly from complexity.

Policy specification techniques beyond SQL will be relevant to us. A series of papers culminating in [Bertino99] defines a calculus of negative permissions with two levels of priority and rules for overriding. We agree with their argument that some such facility is needed, but will seek one that scales up with less complexity. [Jajodia00] proposes "provisions", preconditions that the system state and history must satisfy

before a request can be allowed. There is little comparison with trigger-based alternatives. The action provisions are attractive, but since a security system should be highly predictable, we prefer to move the choice among alternative user -visible actions into other tools. [Sandhu99] provides a rich treatment of roles, providing useful controls on delegation. While not connected explicitly to database systems, they would be useful for implementing composite controls on data objects.

Distributed Database Security

Distributed database security involves all the issues that must be negotiated among systems, including schema integration, user/role sets, degree of autonomy, authentication mechanisms and trust, assigning enforcement responsibilities to servers, combining policies from various authorities, and delegation of authority. The range of tasks is set out in [Jonscher94, De Capitani di V.97, Castano97].

These works aim at specifying ground rules for *pairwise* cooperation, e.g., between a component system and a federation interface. All issues are addressed at system granularity, in terms of a particular federated architecture. They provide limited integration, e.g., no theory to deduce end-to-end effects (even in the many cases where *some* deduction is possible). To summarize, they provide broad toolkits, rather than creating modular abstractions that can tackle each issue thoroughly. We will address a narrower range of issues (just policies, delegation, and enforcement), but will cover distributed systems beyond federations. We emphasize fundamental metadata that governs how each system cooperates can be fine-grained, and supports a rigorous (rather than heuristic) inference theory.

Proposed Work

Existing security management products (e.g., for document management [Gladney97]) and many research prototypes (e.g., for XML authoring [Bertino00]) provide toolkits for a broad range of issues. We will take another approach – abstraction. The database research community has prospered by doing a superb job of providing a few abstractions (Table, Transaction, and subsidiary ones such as Index, Foreign-Function). We will proceed similarly. We aim not to reach the sky, but to raise the ground, i.e., the level of abstraction above which humans tackle the remaining difficult problems.

Each of our modules aims to do a thorough job on a tractable problem. We believe that the toughest problems are better omitted from the base level, general-purpose facilities. Our modules aim to simplify and automate tractable pieces of the real problem, while not making these tough issues (e.g., aggregation, inference threats, metadata secrecy) more difficult.

Our goals are to:

- *Simplify administration:* The abstractions will separate the issues (such as information versus server policies). Not only does this allow each to evolve without impacting the other, but also it allows information factors to be administered by domain experts, restricting technical administrators to server policies.
- *Simplify enforcement:* Given an access control policy, and an indication of what vulnerabilities and run-time costs are tolerable, we will explore ways to generate an implementation automatically (with defense in depth, where appropriate).
- *Simplify implementation:* We want vendors to develop tools to support these tasks, but must compete for their limited development resources. We show how progress can be achieved using small implementation steps that each requires a modest adjustment by customers.

Over the past several years, we have produced models to do the above, in several areas:

- Generating access controls from simpler forms of information
- Passing *limited* administrative authority

- Generating access rights on derived products from those on the sources
- Managing database access controls so that they complement middleware access controls

We now propose to extend these models to suit real applications, and to produce prototype implementations

This research will develop models and implementations that address the following problems with current approaches:

- *Permission Semantics Suitable for Distributed Systems*. Granting access to particular information should be done once, not once for each copy or variant of the same information. This will build on existing work by Rosenthal and others.
- *Managing Permission Sets on Each Object.* Policies applying to a piece of information are more complex than simply "access always allowed" or "access always denied". One may grant permissions "with strings attached", possibly to enforce explicit obligations to protect data. Access controls need to reflect this, automatically determining if a *particular* access is allowed under *all* applicable policies.
- *Integrating Information Releasability with Access Controls.* Some policies are difficult to enforce with access controls: Can I show data I receive to others? Can I put it on my home computer? Capturing these policies in the same framework as access permissions will ease enforcement and make it easier to inform the user of their obligations.
- *Protection Requirements: Vulnerability Tolerance.* Policies should describe not only what is allowed, but also the needed strength of enforcement, i.e., degree of resistance to various threats.
- *Security is about Satisfying Legitimate Needs for Access.* Locking up the data and throwing away the key keeps it secure – but isn't good policy. Capturing and expressing need for service as well as need for security will ensure that security isn't an obstacle.

For each of these, we will develop a model and enforcement mechanism. Wherever possible, we will simplify technology transfer by exploiting or extend SQL security functionality, doing our best to avoid creating slightly different features (as occurred in [Bertino99, Castano97, De Capitani di V.97]). Our first choice will be to transform our model's security policies into SQL grants, view definitions, and triggers, which can be installed into today's robust, efficient DBMSs. This approach minimizes impact on existing applications. Enforcement approaches may also exploit extensions that industry is very likely to provide, e.g., using Public Key Infrastructure (PKI) to pass a wider range of attributes that describe a request.

In other cases, extensions to the SQL model may be more appropriate – potentially leading to issues in query processing and execution. The work will be sufficiently abstract that the techniques will also apply to other data management systems, such as distributed object managers and semistructured data.

Permission Semantics Suitable for Distributed Systems

Security can be greatly improved if data access permissions are appropriately consistent across systems, and are at appropriate granularity. Organizations have not done a good job even of protecting individual databases at appropriate granularity, and today's enterprise and multi-enterprise systems are even farther from that goal. One Oracle VP (W. Maimone, personal communication) estimates that only 30% of databases use SQL's security features to manage security directly. Others use roles whose granularity is application (e.g., SAP), and even these generally grant at table granularity.

SQL (like nondatabase models) has several weaknesses in coordinating permissions for related data:

- SQL controls access to SQL tables (materialized, or derived from specific SQL tables) rather than to information (e.g., taxpayers' Income).
- Even when content is identical or derivable, there is no formal consistency between tables' permissions (except for a view's owner), so one cannot save work by automatically maintaining that consistency.

- A table's administrator cannot grant permission to use it just for computing a view or procedure (except if they are the view definer and administer *all* tables needed for the computation).

As organizations move to make information more widely available, through multiple interfaces and with copies on multiple machines (e.g. in warehouses and federations), the difficulty increases in two ways:

- *Size:* There are more object types (i.e. tables and columns) to administer. SQL grants apply to a single physical table, so each copy or derived interface is administered separately. This complexity also makes it harder for planners to determine whether their users have enough permissions to get their jobs done.
- *Consistency:* Permissions on derived products should, intuitively, be consistent with the sources, but with current models, there is little hope of giving clear guidance to data administrators. Assertions about pairwise autonomy can increase nonlinearly, and require the administrator to understand two systems.

We will extend SQL's security model in three ways:

- We will *factor* the permission decision into parts that can be separately done and maintained, each at an appropriate granularity.
- An *inference theory* will infer those permissions that are justified by content derivability.
- We will define "use within view" permissions' semantics and inference theory, and explore administration pragmatics.

Our steps so far convince us that the goals are practical and useful. In [Rosenthal01a], factoring led to asking administrators to make simpler, smaller decisions, enabling separation of skill and reducing the impact of each change. We provided a rigorous inference theory that takes permissions explicitly asserted on (base and view) tables, and soundly infers additional permissions. No information can be accessed that could not be obtained from the explicit permissions (so we do not complicate the technical difficulties of protecting from inference and aggregation).. We were able to simulate each administration use case in [Cas97, Cap97] by granting or not granting "grant option" to the other sites' administrators, on each table. The theory for factoring seems nearly complete; the next step is to exploit it and examine usability.

Under this project we will devise and prototype algorithms to implement the inference specified in [Rosenthal01a]. Permission inference algorithms have two modes: to check whether the theory permits an individual request, and (more difficult) to derive all permissions on a table (for a large set of users or roles). Initially, we will look at acyclic derivation graphs. The amount of effort invested in ugly details (cycles in derivation graphs, views whose definitions are secret, execute permissions on views) will be determined by the needs of our government partners. We will also prototype tools that transform policy into access controls that work on off the shelf systems, e.g. translating policy into a collection of SQL views and grants that enforce the policy.

We showed the utility of a "use within view" construct in [Rosenthal00b], for situations where one cannot appoint an administrator with rights to all inputs. For example, in application hosting (e.g., a federal system that hosted applications from multiple agencies or states) the service provider's staff might have no permissions on the participants' data that resides on their system. Or imagine creating a list of tax evasion suspects' bank accounts, where the bank will not grant law enforcement unlimited access to deposit records, and law enforcement will not grant access to suspect lists. In conventional warehouses, participants' security depends on both warehouse DBMS correctness and the efforts of warehouse table administrators; in our scenarios, these table administrators have no permissions they can abuse.

We will prototype a Permission Manager (*PM*) tool, as a major extension to today's remote administration tools (from Oracle, Computer Associates, etc.) The PM should accommodate both factored and unfactored

permissions (since one cannot expect all DBMSs or users to change instantly). It should give a full picture of access controls, whether on SQL tables, DBMSs, or network access.

Finally, we will attempt to provide metrics of administration effort with the SQL model and with ours, examining the number of factors in each decision, the number of decisions that are done manually, and the amount that needs to be considered when a situation changes. We will explore factors at different granularities and the effects of differing costs for technical experts, business policy experts, and individuals that can play both roles.

Managing Permission Sets on Each Object

The previous section addressed capabilities among objects related by derivation, especially for distributed systems. Here we address extensions that seem appropriate for the basic permission model. First, we continue our investigation of ways to limit the damage that a malicious or careless administrator can do. Second, we address the reality that many SQL operations (e.g., Create Attribute) require a conjunction of several SQL permissions (rather like the view operations discussed above).

Permissions with Limitations

In standard SQL, a user who has a privilege is able to use it in any circumstance. If they have grant option, they can grant onward to anybody. Yet there are many situations where we cannot trust every grantor with such power.

For example, one may wish to restrict a user's use or onward grant of a privilege to certain days or hours, to non-members of certain groups (for separation of duty), to tables above a threshold size, to tax judgements below a threshold amount, or to requests that are received along a trusted path. Limitations seem particularly important in coalitions of systems, where it may be difficult to impose sanctions for improper use. They also can reduce the scope for damage by an intruder who misuses legitimate users' privileges.

In [Rosenthal00a] we began a model in which one limits individual Grants, not users' privileges. The model appears to have several major advantages (especially when the object owner has delegated grant authority to others): clean lines of authority (you can limit only what you give); less administration (no explicit grant is needed to give you that ability to limit); greater generality (as opposed to any priority-based model), and locally-understandable behavior (rather than weighing positives and negatives from all sources). We propose to carry the model further.

Several techniques are available to limit discretion, including negative permissions [Bertino99], grant-limitation predicates [Rosenthal00a], overriding permission factors [Rosenthal01a], or embodying the limitations in SQL views to which one grants access. Traditional research questions include extending the theory (to support roles, reactivation and dynamic reevaluation of predicates), a fuller treatment of limitations on views (ruled out in [Bertino99]), policies that modify operations' execution [Chapin00, Jajodia00], and efficient implementation. We will attempt to simulate the access control portion parts of the other models [Bertino99, Jajodia00] using limitation predicates. Unlike the other models, ours can act *directly* to limit Grant option – we need to see if this is an advantage.

The theoretical study will begin with simulation arguments – can technique X simulate technique Y? What enhancements are needed? For example, it appears that to simulate our grant-limitation predicates, negative permissions would need two features unavailable or barely available in many DBMSs: metadata about individual grants, plus efficient maintenance of transitive closures. On the other hand, negative permissions offer more power (and complexity) than our Overriding factors. Views are attractive because DBMSs already implement them, but their query modification approach does not distinguish "access

denied" from "empty result", and may not allow alternative paths to give a permission. The theoretical study will also examine auditability, and whether the technique has clear lines of authority.

The pragmatic questions are equally important. Would users see limitations as important, and would they do the administrative work to impose them? How much generality is needed? How does this model compare with security policy managers for enterprises or for digital libraries [Gladney97]? Finally, what tools are needed to make it all usable?

The pragmatic study will examine IRS use cases for limiting permissions. It will also examine how easily one can express (and understand) the answers to some key questions, such as "should this request be accepted?" or "how will the set of permissible actions change after this grant or revoke?" By attaching limitation predicates to a grant, we can refine this "all or nothing" mechanism. The difficulty of getting a robust implementation will also be examined – only the SQL view approach is supported in today's DBMSs. For this reason, comparisons with the view model (with and without our inference improvements) will receive high priority.

Composite Permissions

The SQL99 security specifications have become very large, covering dozens of resources (e.g., schemas, attributes, datatypes, and collating sequences) and an increasingly complex object data model. Oracle goes even farther, offering dozens of additional, nonstandard resource privileges. This sprawl inhibits conceptual progress, complicates implementation, and discourages consistent treatments. Query optimizers reached a similar state around 1990. To overcome the difficulty, the next generation were built as formally structured rule systems, permitting a great increase in capability. We believe that similarly tractable metamodels can simplify SQL security enough to enable the leap to more complex environments.

Reading the SQL standard, one sees that many permissions depend on prerequisite permissions [San97], or (ignoring order) many operations are enabled only after one has received a composite of independent permissions. In SQL99, each operation on each permission type (e.g., Grant, Revoke) is specified independently but with obvious parallelism. We hypothesize that the learning curve would be reduced, and implementation and extension would be significantly simpler, if one just specified the set of prerequisites plus declarative instructions on customization, and *generated* treatments for grant, revoke, and permission checking.

Integrating Information Releasability with Access Controls

Obligations arise when an organization accepts data from a customer or partner or other individual, and promises to protect it from access, release, or unauthorized change. IRS needs to specify obligations for outside recipients of its data, and to track IRS's obligations for data received from taxpayers and from other government and non-government partners. Issues include "can I show it to others?", and "can I put it on my home computer?" In both cases, one must document the obligations explicitly, and ideally, enforce them automatically. (Multilevel secure systems enforce a very inflexible set of obligations, often at high cost.) We will explore ways to unify the treatment of obligations through combining access control and releasability models.

Data releasability policies are the dual of access controls, concerning "what information can be sent 'out'", rather than "what requests can be accepted". The releasability research we have seen has generally concerned high-assurance implementation approaches, e.g., guards [Wiederhold00] or means of attaching policies to the moving data. We propose to examine the issue at the policy level.

We hypothesize that releasability policies can be unified with ordinary access control policies. Instead of having two models and requiring that administrators provide two sets of policy metadata, a unified treatment will reduce effort and improve consistency. For operations that do not change the data (Read,

send email), both deal with "what information can the system release (to a client in the first case, the email recipient in the second). In both cases, the policy can be either based on the kind of information (e.g., values in a certain column), on the content (no dirty words), or both. An update analogy will also be explored (to restrict use of external low-integrity data).

Enforcement might also be shared, especially where normal commercial levels of vulnerability can be tolerated. For example, a hospital's "dirty words" release policy might prevent release of either database outputs or emails that contain certain terms (e.g., "malpractice" or "HIV"), setting them aside for human review. This corresponds (approximately) to giving permission on a view that excludes such words. Popular enforcement techniques, guard controls and DBMSs, have different vulnerabilities, and the combination can be more secure than the individual. Guards cannot easily determine if a query will release sensitive information – one cannot filter *true.* DBMSs are too large to be highly secure, and traditional database controls on tables or even attributes do not catch sensitive information that users have placed in an innocuous Remarks field.

We will devise models and design algorithms for systems containing both access controls and releasability policies. We will formalize relationships between them, to provide inference and reduce administrative work. We will explore how our enforcement techniques extend to releasability policies.

Protection Requirements: Vulnerability Tolerances

A policy specification goes beyond stating what access one chooses to allow; one also needs to understand how firmly the controls' implementation must resist attackers. Depending on the users' reliability and the damage of having data wrongly cross the boundary, enforcement may rely on users' discretion plus auditing, filtering outgoing data automatically or manually, DBMS access controls, or keeping sensitive data on a separate system at a secure location. Reducing vulnerability may entail major losses of flexibility and efficiency.

Enforcement techniques must go beyond allowing or denying access. For policies with a high vulnerability tolerance, audit mechanisms may be appropriate. The question is, how do we support the audit? For example, suppose we allow some access controls to be overridden in a "declared emergency" – with the understanding that such accesses are likely to be audited. What should be logged and (equally important) how can we simplify analysis of this fire hose of information? We will investigate auditor needs, and develop mechanisms that allow flexibility to trade off efficiency requirements with auditor needs.

Domain security experts determine access permissions, and desired tradeoffs between vulnerability and cost. Technologists need to devise models for the policy to be captured, and methods for capturing it. We need to express the vulnerabilities of popular implementation techniques for individual steps of a computation, and from these, to derive estimates of the vulnerability of the entire computation, and its cost. In the long term, many of these will be packaged and used by many organizations. Vulnerability estimates will need to be reviewed frequently, in the light of changing technology. Having them separated from domain experts' decisions will make this easier.

We expect to focus on the framework for enforcement, rather than on detailed analyses of particular design constructs' vulnerabilities (e.g., vulnerability of DBMSs to hackers versus vulnerability to end user browsing). We emphasize threats to information (confidentiality and integrity) rather than denial of service. The vulnerability of an execution strategy can be taken as a "low water mark" of the most vulnerable critical step; an adjustment will be made if there is defense in depth (e.g., tests at multiple stages) and no single step is critical. The protection model would be complemented by a risk analysis model (beyond our scope), which describes the loss due to a successful attack, and the likelihood of different sorts of attacks.

We will use MITRE's security experts to capture knowledge for a range of techniques. The Internal Revenue Service is currently considering an approach that examines authentication policies in terms like the above (executives telling needed resistance for each policy, technologists maintaining knowledge of state of the technology, technologists devising implementations manually.)

Eventually, we will formalize and seek algorithms for the more difficult problem of deploying applications in a way that meets vulnerability requirements. For example, if one wants low vulnerability to an attack by software experts who have access to other parts of the system, physical separation (onto different machines with limited communication) helps, but is inflexible and costly. If one wants protection from insider abuses, a DBMS can make subtler discriminations. The long-term goal is to include vulnerability as a factor in deploying functions to the various servers (what can be co-located) and in determining execution strategies (What data can be safely transmitted? What servers can be relied on to enforce access controls or to apply views that filter data?) To do this research, we will draw on experience from distributed query processing (especially strategy enumeration) and database physical design/file deployment.

Security is about Satisfying Legitimate Needs for Access

Existing permissions may not capture the actual needs for service – they may model what was convenient to support with yesterday's technology. Technical limitations may cause some data that does not need protection to be lumped with more sensitive information; they may also cause users to be given permissions broader than their need to know (e.g., all data in a table for which they need two columns). Good management requires that we document desires for service, and use that information to design the policies.

When there is a gap between the protection requirements and the need for service, there is room for alternatives in how to enforce the policies. In [Rosenthal01b] we explored synthesizing "least privilege" enforceable by the servers in question, such that all needs are met. Other tradeoffs are possible, and analysis techniques are also needed for auditing (though synthesis tools seem likely to get a broader market). We will start with a model that assumes designers have assigned functions to servers, and the security analyst must decide "what access controls may be placed on each function". Initially, we will seek "least privilege", providing the tightest controls that are known to still allow necessary function to execute. Later we will consider execution costs, trading them against vulnerability.

We now discuss a technical approach to this task. A model of desire for service has two parts. First, one should specify what users desire, i.e., the ability to execute certain functions that are of value to them. Second, because controls or concerns may be on implementation objects such as data tables, one needs to know the *implements* relationship, i.e., which lower level functions that are subject to access controls are needed to implement the first batch. In [Rosenthal01b] we refined the *implements* relationship to accept limitation predicates, e.g., ability to (PurchaseItem where cost<$5000.)

There are two pragmatic problems: First, how can one describe – and capture descriptions of – the implements relationship. While functions' behavior cannot be predicted perfectly – the halting problem intervenes– often one can obtain usable bounds. Capture is easy in a few cases (e.g., SQL views). Program-slicing techniques can handle some limited cases of general code (if source code is available). Data mining techniques (below) are promising. But the theory using the bounds needs to be robust, since we will usually have approximations to both user desires and *implements*. Second, one must model the servers that enforce permissions: What tests can they execute? Can and will they request data from other systems for complex predicates? Who trusts them? Will the test predicates themselves violate security? What are the execution costs?

Data Mining to Learn Security Policy

Regardless of the advances in modeling security requirements, administrators will rarely specify the desired access controls *precisely* (i.e. capturing fine distinctions of who should see which granules of data). Instead, one is likely to provide coarse statements that identify data that needs strong protection from outsiders, and to provide little discrimination among insiders. For this realm, an alternative is to base policy on common practice – how the data is generally used. (This idea has a long history, and the existence of "easement by implication" and "easement by prescription" hold that the way real estate is used can give legal rights to that use.) Establishing policy based on common use may lower organizational resistance to new policies – a "top-down" policy that (perhaps unintentionally) prevents what people are used to will likely face problems. We propose to exploit the capability of data mining to discover generalized profiles, in order to provide templates for policy. Security administrators would need only to refine the templates and determine enforcement, rather than defining policy from scratch.

Multiple techniques will be needed. For example, a single profile isn't enough – an upper bound and lower bound will provide much more information in developing tradeoffs between legitimate access needs and security expectations. Profiles at different levels of granularity are also important. A course-grained profile will be most useful for policy-makers. However, a fine-grained profile can provide useful information to auditors

There has been research in profiling to support intrusion detection [Bueschkes99, Fawcet96, Lee98, Mukkamala99]. There has even been work on profiling database access, both for intrusion detection [Chung99], and to learn security policy [Chung00]. The novelty in this proposal is based on the tie between the learning process and security models. For example, a security model may specify three "strengths of enforcement": *Always enforce*, *enforce except in emergencies*, and *suggest for audit*. This leads to a need to learn three categories of profiles: Accesses that never happen, those that happen rarely but are clustered with other rare accesses, and those that appear occasionally with little apparent correlation.

The interaction between known constraints enforced by existing policy, known policy not directly tied to enforced constraints, and the pattern of accesses demands additional flexibility in data mining. The PIs prior work in data mining on text has resulted in new techniques that enable such flexibility, and methods that utilize that flexibility [Tsur98, Clifton99].

Another constraint on profiling of database accesses is the size of data relative to the number of accesses: many data items may be only accessed a few times (except as part of summary queries). Too few, and the results of data mining are meaningless [Clifton00]. One solution is to group data items such that total accesses to all items in the group are sufficient. This grouping is another form of mining – and may use access patterns, but will also use content values and known security constraints.

Constrained data mining has other applications as well – for example, it may help address finding "needles in a haystack", a problem faced by intelligence analysis in the face of asymmetric threats. Working in the context of security models provides a well-defined set of constraints on data mining, both in terms of results produced and known "facts" (existing policies). This gives a good environment for advancing data mining technology, and ensures that the advances will have immediate impact to a critical problem.

Educating the Next Generation of Developers

We have too few people who can use even today's data security mechanisms. The SQL Grant structure is quite powerful – but developing an appropriate set of grants to enforce a particular policy demands a very specialized set of skills. The most common alternative (where data-level permissions are enforced at all) is to use more general programming skills. Instead of putting the controls on the data, the controls are

programmed into the application, and then repeated in the next application and the next, intertwined with application code. The result is inconsistency and a maintenance nightmare. But for the short term, it means that you only need an application programmer, not an application programmer AND a database security specialist. For tomorrow's enterprise systems, comprising databases, messaging, application servers, web servers, etc., the need for single specification will be even greater.

There are efforts underway to improve the state of education for information security. The National INFOSEC Education and Training Program [NIETP] aims to "provide the foundation for a dramatic increase in the population of trained and professionalized security experts." However, the ability of our educational system to supply such professionals will be limited – the demand for computer scientists of all types far outstrips our ability to train them. The Department of Commerce's Office of Technology Policy estimated that in 1998 there were 2,179,000 "core IT" workers, with more than 2 million new needed by 2008, over 75% of which require at least a bachelor's degree [Sargent00, Mears98]. However, through the 1990s, Computer and Information Science enrollments held at around 25000/year [EDStats00]. This demand for an order-of-magnitude growth in computer professionals of all types will drastically cramp our ability to dramatically increase the number of security experts. More likely, the vast majority of systems will continue to have security mechanisms developed by the programmers that created them.

This research will provide mechanisms that are easier to administer – simpler knobs and more relevant gauges. Tasks (and the skills required) will be separated between domain experts, security tool developers, system developers and administrators, and data administrators. To make this work, we need to ensure that developers know how to build systems that use these mechanisms. To this end, Professor Clifton will develop a module on database security mechanisms that can be incorporated into existing database courses, and also for courses in system security. This module will initially concentrate on how to utilize SQL security mechanisms to enforce security policy, as an "add-on" to existing database and information systems courses. As the research progresses, the course module will incorporate the models and mechanisms that show the most promise of wide acceptance. This course module will be made available to the wider community through various channels, initially the National INFOSEC Education and Training Program's Center of Academic Excellence in Information Assurance Education at Purdue.

Technology Transfer

Transitioning research results from the laboratory to government applications is a challenging problem. This proposal has a novel and unique transfer path – through interaction with MITRE staff supporting the IRS Systems Modernization effort. The PIs will work closely with MITRE staff that support the IRS Office of Cyber Security, in particular Richard Graubart. They have experience with research in this area, and with transitioning it to practice. In the initial stages of the project, this interaction will concentrate on bringing knowledge of IRS security needs and policy specification mechanisms into the research. As models are developed, the MITRE staff involved will test them against actual IRS policies.

We will develop user interface mock-ups to test the ability of the models to express IRS policies, and extend these to prototypes as the project progresses. These prototypes will be made available to MITRE staff assisting in the definition of IRS security policies – where appropriate, MITRE staff can use the developed models and tools to specify security policy for IRS systems. In cases where the tools are particularly effective at capturing IRS policy, and the IRS systems support appropriate access control mechanisms, the PIs and project staff will assist in converting the captured policies into enforcement mechanisms, either using the prototype analysis tools or manually. To support the transition of technology between Purdue and MITRE, there will be regular communication and visits between the staff involved on the project. In addition, one of the students funded by the project will spend their summer at MITRE.

As the research matures, MITRE staff will be able to apply the results to their existing IRS-funded projects. In other cases, vendor support will be needed to make technology transfer a reality. As neither MITRE nor Purdue is in the commercial software business, both will license or freely provide the results of this research to the vendors producing new IRS systems. MITRE has a history of releasing research prototypes as freeware and open-source software [MITRETT]. In addition, the PIs will give talks to the wider vendor community to push for inclusion of the results of this research into commercial products.

A more traditional transfer route – to software vendors and standards committees– will also be employed. The PIs lecture at major vendor research labs (Microsoft, IBM Almaden, HP Labs) and have cooperated in the past with the ANSI/ISO SQL standards committee (see supporting documents). In the past, our ideas have influenced query processors for IBM's DB2 (recursion and outerjoin) [Carey99] and Microsoft SQL Server.

Results from Prior NSF Support

Although these awards fall outside the five-year window for required inclusion of results from prior NSF support, they are given for completeness.

Prof. Clifton

NSF Award number 9210704, $90,000, September 1 1992 – August 31, 1995, *Shared Software Database*.

The objective of this project was to develop techniques for searching for and sharing software using wide-area networks. The project produced a novel technique for matching specifications of code modules with implementations, based on structural information automatically extracted from Computer-Aided Software Engineering (CASE) tools. Thus a user of a CASE tool could design their program, and automatically be provided with implemented code that appeared to be similar to the modules in the design. All indexing information was automatically extracted from the design and code – no "special" description was needed for reuse.

The techniques developed for this also had impact in the arena of heterogeneous database integration. SemInt, a Semantic Integration tool relying on schema information automatically extracted from databases, was produced under this project. SemInt used machine learning techniques to identify similar attributes in different databases. This was a novel approach to a very difficult problem in data integration.

One student obtained his Ph.D. while funded by this grant, and four Master's students were involved in the work (one of whom received funding from the grant).

Publications resulting from this award:

Wen-Syan Li and Chris Clifton, "SEMINT: A Tool for Identifying Attribute Correspondences in Heterogeneous Databases Using Neural Networks", *Data and Knowledge Engineering* 33(1), Elsevier Science, Amsterdam, April 2000.
Wen-Syan Li, Chris Clifton, and Shu-Yao Liu, "Database Integration Using Neural Networks: Implementation and Experiences", *Knowledge and Information Systems* 2(1), Springer-Verlag, London, February 2000.
Peter Scheuermann, Wen-Syan Li, and Chris Clifton , "Multidatabase Query Processing with Uncertainty in Global Keys and Attribute Values", *Journal of the American Society for Information Science* 49(3), John Wiley & Sons, Philadelphia, PA, March 1998.
Wen-Syan Li and Chris Clifton, "Dynamic Integration in Multidatabase Systems", *Journal of Database Management* 7(1), IDEA Group Publishing, Harrisburg, PA, Winter 1996.

Peter Scheuermann, Wen-Syan Li, and Chris Clifton, "Dynamic Integration and Query Processing with Ranked Role Sets", *Proceedings of the First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, Brussels, Belgium, June 19-21, 1996.
Chris Clifton and Wen_Syan Li, "Classifying Software Components Using Design Characteristics", *The Tenth Knowledge-Based Software Engineering Conference*, November 12-15, 1995, Boston, MA.
Wen-Syan Li and Chris Clifton, "Semantic Integration in Heterogeneous Databases using Neural Networks," *20th International Conference on Very Large Data Bases*, Santiago, Chile, September 12-15 1994.

Dr. Rosenthal

NSF Award number 7701753, $40,000, January 1 1978 – June 30 1980, *Dynamic Programming and Other Decomposition Algorithms*.

We succeeded in abstracting several algorithms and several lower bounds on computational complexity, in ways that preserved their power and improved generality. These were published in a series of discrete algorithms papers. We began with dynamic programming, and expanded to cover other algorithms that produced many results for the price of one. The general results sometimes had greater power than published special case results. After leaving academia, I collaborated for several years with ex-students to complete the work.

Publications resulting from this award:

P. Helman, A. Rosenthal, "A Comprehensive Model of Dynamic Programming," *SIAM J. on Discrete and Algebraic Methods*, 6(2), Apr. 1985.
A. Rosenthal, "Dynamic Programming is Optimal for Nonserial Optimization Problems," *SIAM J. on Computing*, 1982.
A. Rosenthal, "Optimal Algorithms for Sensitivity Analysis in Associative Multiplication Problems," *Theoretical Computer Science*, 1981.
A. Rosenthal, "Optimal Mass Production," *Discrete Applied Mathematics*, 1980.
A. Rosenthal, "Dynamic Programming is Optimal for Certain Sequential Decision Processes," *J. Mathematical Analysis and Application*, 1980.
A. Rosenthal, J. Pino, "A Generalized Algorithm for Centrality Problems on Trees," *JACM*, Apr. 1989.
H.D. Ratliffe, A. Rosenthal, "Order-Packing in a Rectangular Warehouse -- A Solvable Case of the Traveling Salesman Problem," *Operations Research* 31,2, May-June 1983.
M. Conrad, A. Rosenthal, "Limits on the Computing Power of Biological Systems," *Bulletin of Mathematical Biology*, 1981.
A. Rosenthal, "Series-Parallel Reduction for Difficult Measures of Network Reliability," *Networks*, 1981.
A. Rosenthal, "Decomposition Methods for Fault Tree Analysis," *IEEE Transactions on Reliability*, 29(2), June 1980.
A. Rosenthal, "Approaches to Comparing Cut Set Enumeration Algorithms," *IEEE Transactions on Reliability, April 1979.*
A. Rosenthal, "Notes on Closed Form Solutions for Delta-Star and Star-Delta Conversion of Reliability Networks," *IEEE Transactions on Reliability, August 1978.*