A SURVEY OF ACCESS CONTROL MODELS

INTRODUCTION

Computer systems and the information that they create, process, transfer, and store have become indispensable to the modern enterprise. In today's on-demand, always connected, data-driven world—and especially in light of the transformation of entire national economies from manufacturing-based paradigms to knowledge-based ones—many organizations rightly count their information systems among their most important assets. Organizations often use these IT systems to store and process vast quantities of sensitive data, which, if disclosed, could be potentially damaging to an organization. At best, an organization may be embarrassed by an unauthorized disclosure; at worst, it may lose its competitive stance in the market if the information were a proprietary trade secret, or may be sued if the information were confidential customer information. Some companies have gone out of business when the damage from an unauthorized access proved too great for them to weather.

Loss of competitive advantage, or even going out of business, is, of course, a very grave situation. However, it is not the worst outcome imaginable from an unauthorized access to an information system. IT systems now integrate with—and even control—critical national infrastructure components, such as the hardware components responsible for the safe operation of power plants, chemical manufacturing facilities, and transportation systems. Controlled access to these types of systems is critical because of the very real potential for loss of life or massive environmental and infrastructure damage that improper or malicious operation could cause.

Organizations use access control mechanisms to mitigate the risks of unauthorized access to their data, resources, and systems. Several access control models exist. Their corresponding access control mechanisms—the concrete implementations of those access control models—can take several forms, make use of different technologies and underlying infrastructure components, and involve varying degrees of complexity. In some cases, the more complicated models expand upon and enhance earlier models, while in other cases they represent a rethinking of the fundamental manner in which access control should be done. In many cases, the newer, more complicated models arose not from deficiencies in the security that earlier models provide, but from the need for new models to address changes in organizational structures, technologies, organizational needs, technical capabilities, and/or organizational relationships.

The business-to-business (B2B) relationships that enable organizations to successfully execute their missions, for example, sometimes require users or systems from one business to access resources from business partners. Simpler access control models often cannot adequately meet the complex access control requirements that such relationships require, and so more granular, powerful, dynamic models and

mechanisms are needed to address these new realities. In short, increasingly complex data access and sharing requirements drive the need for increasingly complex access control models and mechanisms (see Figure 1). The rest of this paper discusses current and future access control models—including access control lists, role-based access control, attribute-based access control, policy-based access control, and risk-adaptive access control—and the infrastructure needed to support them.

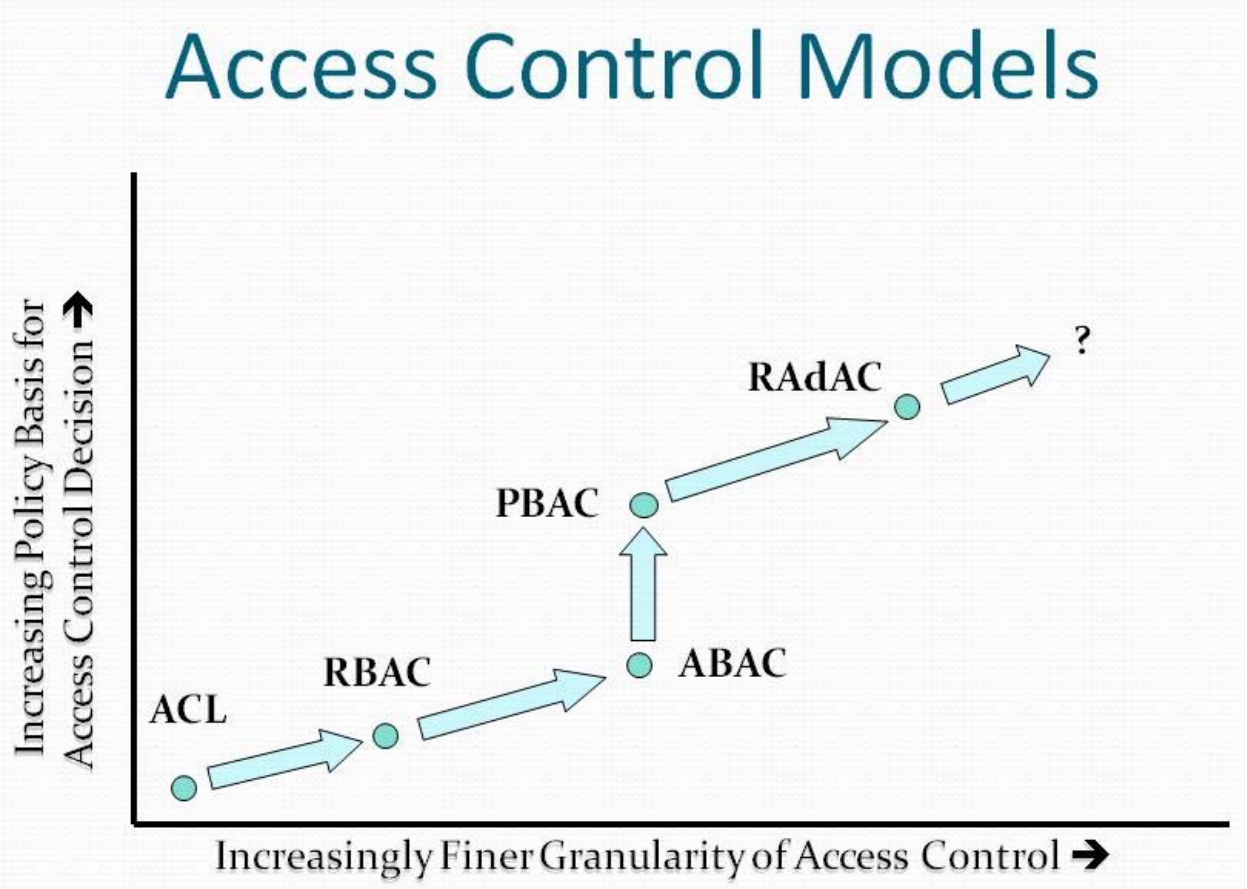

Figure 1--Selected Access Control Models

ACCESS CONTROL LISTS

Access Control Lists (ACLs) are the oldest and most basic form of access control. They gained prominence in the 1970s with the advent of multiuser systems where the need to limit access to files and data on shared systems became necessary. Not surprisingly, some of the earliest access control lists were implemented on UNIX systems. Later, as multiuser operating systems for personal use became popular, the idea of ACLs was introduced into them as well. Many modern operating systems, whether UNIX, UNIX-like, or Windows, make use of ACLs at some level, although the access control mechanisms used to protect system resources have become more complex in recent years.

The concept of an ACL is very simple: each resource on a system to which access should be controlled, referred to as an object, has its own associated list of mappings between the set of entities requesting access to the resource and the set of actions that each entity can take on the resource. For example, each file on a file system might have an associated data structure that holds the list of users that the operating system as a whole recognizes, along with a flag which indicates whether each user may read, write, execute, delete, or modify the file (or some combination of these). Whenever a user tries to perform any of these actions on the file, the operating system checks the file's ACL and determines whether the requested action—appending data to the file, for example—is allowed. If the action is allowed for that user, the data is appended; if not, the append operation fails. This may also apply to groups of objects—a directory, or a group of processes for example—and current ACL implementations often provide users with sufficient privileges to modify the ACLs associated with objects.

Although ACLs are commonly associated with operating systems on a single system—they are, in fact, ubiquitous across all modern operating systems at one level or another—the ACL concept has also been implemented in other contexts. They have, for example, been used in network contexts wherein the target resource to which access is sought rests on remote systems. Some applications also maintain access control lists to determine which users are able to view certain data elements (some relational database management systems, for example, may make use of ACLs as a simple way to implement data views, so that one user has a different view of a subset of data than another).

The relative simplicity of ACLs means they do not need much underlying technological infrastructure to work. They are prevalent across all modern operating systems, even among many of the most basic ones, so every organization that makes use of an operating system almost certainly has an ACL implementation by default. At the application level, where developers may feel the need to implement access control list functionality, this can be done using basic built-in map-type data structures common in many programming languages—such as dictionaries in the Python programming language—or maps in the Java programming language, and relatively simple functions.

In cases where ACLs need to be managed for hundreds of thousands or millions of users, and where in-memory data structures do not scale well, databases may be used to store some of the ACL data.

While widely used, ACLs do have their limitations. The ACL for a particular file, process, or other resource must be checked every time the resource is accessed, and this can be an inefficient means of providing access control. Furthermore, ACLs control not only user access to system resources; they also control application and system access as well. So in a typical computing session, the files a user tries to access perform ACL lookups, the applications he tries to open perform ACL lookups, the files and applications those applications open and modify perform lookups, and the system applications perform lookups, and so on.

ACLs can also be difficult to manage in an enterprise setting where many people need to have different levels of access to many different resources. Selectively adding, deleting and changing ACLs on individual files, or even groups of files, can be time-consuming and error-prone.

ROLE-BASED ACCESS CONTROL

Role-based Access Control (RBAC) is a newer access control model than the ACL paradigm. Unlike ACLs, access to a resource is determined based on the relationship between the requester and the organization or owner in control of the resource; in other words, the requester's role or function will determine whether access will be granted or denied.

Role-based Access Control addresses some of the shortfalls of the ACL model, while presenting some new and interesting opportunities. For example, one limitation of the ACL model is that it treats every user as a distinct entity with distinct sets of permissions for each resource. This means that ACLs are resource-focused. ACLs have to be set for each resource (or group of resources) separately, a cumbersome process when large groups of resources are involved, or when different people need to be able to access different resources. The fact that ACLs are generally set a by resource's owner and are not always centrally managed only complicates matters, since a fair amount of coordination and planning has to be done to ensure that the correct people have the correct access to the correct resources. In short, the largest single pitfall of the ACL model is that it has limited scalability at the enterprise level.

Since RBAC determines access based on roles, and since more than one person can have the same role (the role of software engineer, for example), RBAC allows for the grouping of individuals into categories of people who fulfill a particular role. This means that one set of access control permissions on a particular resource—the source code tree for a new piece of software, for instance—can be set once for all members of the software engineering department.

RBAC also allows users to be members of multiple groups. An accountant can be a member of the “employees” group, thereby gaining access to resources to which all employees need access, but would also be a member of the “accounting” group, which provides access to the company’s spreadsheets and financial reports. RBAC also creates the potential for hierarchies of permissions and inheritance, wherein more restrictive permissions override more general permissions.

Variants of the RBAC model have been implemented at different levels. As with ACLs, there is at least rudimentary support for groups and roles in most modern desktop operating systems, so at a minimum the infrastructure for RBAC is available to operating system users. Windows 2000 and later operating systems added the concept of “groups,” which facilitate RBAC. Among some of the groups available in these operating systems are the “Administrator” group, members of which have complete control over the operating system, “Power Users,” which have fewer privileges than administrators, but still operate with elevated privileges, and normal users, which have limited privileges on the system.

RBAC is also increasingly being implemented at the application level, particularly in enterprise settings where it is commonly implemented as a component of enterprise middleware. Implementing RBAC at the application level creates new opportunities for scalability and versatility because a single middleware product can be used to control access to many systems and resources. Tivoli Identity Manager, for example, has an RBAC component which treats someone’s role as a part of his/her identity. The Identity Manager then mediates access to a large variety of operating system services and enterprise resources.

Most implementations of RBAC-based middleware require additional infrastructure components. Many, for example, require directory services, such as Microsoft Active Directory or Sun Java System Directory Server, in addition to relational database management systems, such as offerings from Oracle or IBM. The need for additional middleware or infrastructure components varies depending on the vendor; several vendors offer complete infrastructure solutions in support of their access control products.

Despite its many advantages (particularly when compared with the ACL model), RBAC has its own disadvantages. One of the most significant is the fact that dividing people into categories based on roles makes it more difficult to define granular access controls for each person. It is often necessary to create more specific versions of roles or devise other mechanisms to exclude specific individuals who fall into a particular role, but do not necessarily need to have the full rights accorded to other members of a group. Consider a large organization, UltraMegaCorp, which has subsidiaries in multiple locations. It might be necessary to segregate IT personnel across the various locations. Furthermore, it might be necessary to organize IT systems so that certain resources reside in particular locations with their own administration personnel. With such a setup, a generic “administrator” role might need to be decomposed into sub-roles based on the type of resource to be administered, and perhaps also based on the location that the

resource serves. Giving a SharePoint administrator from the Colorado site office the ability to administer a SharePoint deployment at the New York headquarters might make sense; on the other hand, it might not. However, creating a generic “SharePoint Administrator” role does not allow for easy differentiation between the two use cases; opportunities exist for one administrator to have unnecessary or undesired access to particular systems. What is needed in this case is the ability to differentiate individual members of a group and to selectively allow or deny access based on a granular set of attributes. The Attribute-based Access Control (ABAC) model was designed to fulfill this requirement.

ATTRIBUTE-BASED ACCESS CONTROL

Attribute Based Access Control (ABAC) is an access control model wherein the access control decisions are made based on a set of characteristics, or attributes, associated with the requester, the environment, and/or the resource itself. Each attribute is a discrete, distinct field that a policy decision point can compare against a set of values to determine whether or not to allow or deny access. The attributes do not necessarily need to be related to each other, and in fact, the attributes that go into making a decision can come from disparate, unrelated sources. They can be as diverse as the date an employee was hired, to the projects on which the employee works, to the location where the employee is stationed, or some combination of the above. One should also note that an employee’s role in the organization can serve as one attribute that can be (and often is) used in making an access control decision.

A typical ABAC scenario involves a requester who attempts to access a system either directly or through an intermediary. The requester will have to directly or indirectly provide a set of attributes that will be used to determine whether the access will be allowed. Once the requester provides these attributes, they are checked against the permissible attributes and a decision will be made depending on the rules for access. If UltraMegaCorp implemented an ABAC model for access to their distributed infrastructure, for example, they could create access control rules that state that a person who tries to access a particular administration interface for a critical router in New York must present credentials with a division attribute of “5,” which corresponds to the IT division, a title of “senior network engineer,” and a location attribute of “New York.” If any of these attributes do not match, access to the server will be denied.

A key advantage to the ABAC model is that there is no need for the requester to be known in advance to the system or resource to which access is sought. As long as the attributes that the requestor supplies meet the criteria for gaining entry, access will be granted. Thus, ABAC is particularly useful for situations in which organizations or resource owners want unanticipated users to be able to gain access as long as they have attributes that meet certain criteria. This ability to determine access without the need for a predefined list of individuals that are approved for access is critical in large enterprises where the people may join or leave the organization arbitrarily.

Unlike RBAC and ACLs, readily available operating systems do not inherently support the ABAC model. Instead, such access control is most often implemented at the application level, with an intermediary application that helps to mediate access between a user or application and the resource to which access is requested. For relatively simple implementations, large databases or other infrastructure are not necessary and the application logic for allowing access based on attributes is all that is required. In more complicated environments, however, the need for databases becomes critical, particularly if some of the attributes that go into making a decision include organizational or personal information. For example, if a person's role in the organization were used as one of the attributes that determines access, a database and directory services infrastructure become indispensible.

One limitation of the ABAC model is that in a large environment with many resources, individuals, and applications, there can be disparate attributes and access control mechanisms among the organizational units. It is often necessary to harmonize access control across the enterprise in order to meet enterprise governance requirements. Policy-based Access Control (PBAC) enables organizations to have a more uniform access control model throughout the organization.

POLICY-BASED ACCESS CONTROL

Most organizations have some kind of policy and governance structure in place to ensure the successful execution of the organization's mission, to mitigate risk, and to ensure accountability and compliance with relevant law and regulations. The internal security posture of most companies and organizations has traditionally been out of the purview of law and regulation, although banking, government-related bodies, and critical infrastructure are some notable examples of organizations where the government has exercised its authority to push for tighter security controls. With the institution of regulation and legislation in several industries, such as Gramm-Leach-Bliley (GLBA) for financial services, Health Insurance Portability and Accountability Act (HIPAA) for healthcare, and Sarbanes-Oxley (SOX) for corporations, many organizations are discovering that they need to put into place tighter policies and uniform controls across the enterprise in order to stay in compliance. They need to create and enforce policies that define who should have access to what resources, and under what circumstances. They also need to put in place mechanisms so that access can be easily audited because these Acts hold the organizations' executives responsible for their subordinates' actions. Policy-based Access Control (PBAC) is an emerging model that seeks to help enterprises address the need to implement concrete access controls based on abstract policy and governance requirements.

In general, PBAC can be said to be a harmonization and standardization of the ABAC model at an enterprise level in support of specific governance objectives. PBAC combines attributes from the resource, the environment, and the requester with information on the particular set of circumstances under which the access request is made, and uses rule sets that specify whether the access is allowed under

organizational policy for those attributes under those circumstances. In an ABAC-only model, the attributes required to gain access to a particular resource are determined on a local level and can vary greatly from one organizational unit to the next. For example, one organizational unit might determine that access to a sensitive document repository requires credentials with a username, organizational role and password; another unit might require that the credentials necessary to access its repository also include a digital certificate issued by a trusted Certificate Authority. If documents are transferred from the latter repository to the former one, they lose the protection afforded by the digital certificates, and thus can be more easily compromised. Under the PBAC model, the organization would likely have one policy governing access to all resources that meet particular sensitivity criteria, and this policy would be enforced across the board for all attempts to access the resource, no matter where the documents are housed at any given point.

Although PBAC is an evolution of ABAC, it is a much more complicated model. Since the attributes have to be maintained across the enterprise, it is necessary to design and deploy enterprise-level systems to accommodate PBAC. This includes databases, directory services, and other middleware and management applications, all of which must be seamlessly integrated. In contrast to the other access control models, PBAC requires not only complicated application-level logic to determine access based on attributes, but also a mechanism to specify policy rules in unambiguous terms. It is extremely important that policies be unambiguous; otherwise, there is the potential for unintended, unauthorized access to a resource with which a particular policy is associated. The eXtensible Access Control Markup Language (XACML) is based upon XML, and was developed as a way to specify access control policy in a machine-readable format. Unfortunately, policy creation can be complicated and the use of XACML does not necessarily make the task of creating, specifying, and enforcing good access control policy any less difficult.

There is also a need to ensure that the entire enterprise uses the same attributes for access, and that all of the attributes are from an authoritative source. In simple terms, an Authoritative Attribute Source (AAS) is the one source of attribute data that is authorized by the organization and that overrides all other attribute sources. Ideally, policies should be able to specify which sources of attributes are authoritative for the particular policy, and there should be mechanisms to verify that the attributes provided by a requester come from the AAS. Although seemingly simple in theory, in practice it can be very difficult to establish one authoritative attribute source. This is especially true in situations in which different enterprises must work together, and must implement access control among them. One organization might consider a particular repository of attribute data authoritative, but another partner may consider the repository inadequate. Thus, like policy, the establishment of an authoritative attribute source that all partners can agree upon is not necessarily an easy task.

RISK-ADAPTIVE ACCESS CONTROL

Organizations are not static; they constantly evolve and respond to a variety of stimuli, which can include legal requirements, economic and financial realities, market challenges, a variety of risk factors, and leadership styles. Their dynamic nature means that the policies that guide them must also be adaptable; this naturally extends to the organization's security and access control requirements as well. The security threats that organizations face are also dynamic, so they must constantly assess the risk to their IT infrastructure and the associated data. Even the more advanced access control paradigms, such as ABAC and PBAC cannot adequately address the need for dynamism and changes in the risk levels. The Risk-Adaptive Access Control (RAdAC) model was devised to bring real-time, adaptable, risk-aware access control to the enterprise.

RAdAC represents a fundamental shift in the way access control is managed. It extends upon other earlier access control models by introducing environmental conditions and risk levels into the access control decision process, in addition to the concept of "operational need." RAdAC goes beyond the traditional reliance on static attributes and policies. It combines information about a person (or machine's) trustworthiness, information about the corporate IT infrastructure, and environmental risk factors and uses all of this information to create an overall quantifiable risk metric. RAdAC also uses situational factors as input for the decision-making process. These situational inputs could include information on the current threat level an organization faces based on data gathered from other sources, such as CERTs or security vendors.

After all of this information is gathered, it is compared against access control policy. The access control policy could include directives for how access control should be handled under a variety of situations and with a variety of risk levels. For example, under normal operating conditions a person may be able to log into an IT system with a username and password per the normal operating policy, but under heightened conditions, RAdAC could enforce a second, stricter policy that also requires a digital certificate for two-factor authentication. Under RAdAC, policies would be able to specify the set of circumstances under which ordinarily strong access control could be relaxed to allow access based upon a determination of operational need to access a resource. This means that RAdAC allows operational need to override security risk, if necessary, and if the policies allow the overrides. RAdAC also takes a probabilistic, heuristic approach to determine whether the access should be granted under the circumstances. The heuristics include a historical record of access control decisions and machine learning. This means that a RAdAC system will use previous decisions as one input when determining whether access will be granted to a resource in the future.

The infrastructure required to support a RAdAC implementation is understandably large and complex given the number of inputs that would be required to autonomously make a reasonable access decision based on risk metrics. Among the diverse array of systems and inputs necessary to make RAdAC work are: key management services, a situational awareness service, user information, metadata and attributes for resources,

policies that help to determine access control, information about enterprise IT systems, and a repository of access decision data. Some of these RAdAC components are data repositories, such as databases; others, such as those that provide user data, are directory services; many are full-fledged applications in their own right. Some of these applications can be standalone; in many environments, a better approach might be to implement them as loosely-coupled services. A service-oriented architecture approach to RAdAC implementation helps to limit vendor lock-in and promotes modular architecture. This in turn enables enterprises to add and remove modules as needed, or to select from among different modules that encapsulate similar functionality. A SOA approach also facilitates standards-based information exchange, which is critical to the success of inter-enterprise access control decision-making.

Despite the power and attractiveness that RAdAC provides, actually implementing the model will be daunting. There are numerous obstacles to successful implementation, many of them technical in nature. First, integrating the many systems involved in RAdAC will be a challenge, given that they are diverse and data exchange among them has not been standardized. Second, RAdAC, like PBAC, relies on digital policies in order to help to determine whether access should be allowed. This will require standardized ways of exchanging policy, and a means to unambiguously define these digital policies so that the RAdAC system can correctly interpret them. As with the PBAC model, XACML is a potential solution to this problem, but it needs to reach a higher level of maturity before it can be included in a RAdAC solution. Third, trustworthy sources of user information will need to be made available to the system; the need for trustworthy information about the status, capabilities and security posture of various parts of the IT infrastructure is equally important. Trusted Platform Modules (TPMs), which are hardware components that can attest to the trustworthiness of a system, are one potential source of information; behavioral analysis might be able to do the same for users. Unfortunately, both TPMs and automated behavioral analysis have a long way to go before they can be reliably implemented and integrated into a RAdAC scheme. Fourth, since a part of RAdAC's functionality is the ability to adapt to changing environmental conditions, unambiguous mechanisms to describe the various environmental conditions that need to go into an access control decision is also necessary. Again, standardized data exchange formats will determine how robust, adaptable, and useful the environmental data will be in a RAdAC implementation. A fifth challenge for RAdAC is the reliance on heuristics in the assessment of whether to allow access to a system. Machine learning, genetic algorithms, and heuristics have come a long way, but they still have a long way to go before they can be included in RAdAC.

Finally, RAdAC faces a variety of non-technical challenges, including those of policy and law. Does deploying RAdAC in certain environments violate the law? Who is accountable if a security breach were to occur, given that the decisions to allow or deny access to a system are automated? Are the system owners, the RAdAC implementers and administrators, and/or the RAdAC system designers ultimately responsible if a breach were to occur? These questions must be addressed before RAdAC can be widely deployed, and certainly before organizations feel comfortable allowing RAdAC to control access to their sensitive information.