PANEL:

Which Access Control Technique will Provide the Greatest Overall Benefit?

Moderator: Timothy Fraser
(NAI Labs)

PANELISTS: David Ferraiolo, Mikel Matthews, Casey Schaufler, Stephen Smalley, Robert Watson

The Question Before the Panel:

Considering all factors (for example: quality of protection, performance, compatibility, ease of use), which operating system access control technique will provide the greatest overall benefit to users?

Panelists Positions follow on the next 8 pages.

SACMAT'01, May 3-4, 2001, Chantilly, Virginia, USA
ACM 1-58113-350-2/01/00005.

An Argument for the Role-Based Access Control Model

David F. Ferraiolo
National Institute of Standards and Technology

The fundamental objective of any access control system is to protect system resources against inappropriate or undesired user access. In practice this objective has been met with a mechanism that translates a user's access request through a simple table lookup of the access control matrix [14] – to grant or deny access. Although, access control may in some respects seems straight forward, and mundane, of all disciplines of security, access control stands out as being the least mature and the most problematic. The difficulty lies in the policy or the meaning behind the phrase – "inappropriate or undesired user access." The reality is that access control policies can differ greatly from one organization to another. For instance the military, banking, and healthcare institutions have all defined and formally modeled unique policies. The question is, what is the most advantageous security model to meet these diverse policy needs? From an interoperability perspective, what model should be used to drive access control mechanisms of host and network operating systems, database management systems, and enterprise management systems that form the information infrastructures of government and commercial organizations?

To deal with commercial requirements, newer formal models for access control have been developed that go beyond the simple access control matrix model for the design of more complex security policies [4, 5, 15, 19, 21, 24, 25]. Among these models Role-Based Access Control (RBAC) [6, 7, 10, 17, 20] stands out in its ability to meet a large variety of policy objectives. By making central use of role-based structures and going beyond the simple table lookup of the access control matrix, and the fixed structure of a lattice of security labels, access control models have evolved in their support of an increasing range of access control policies. In particular, RBAC has been shown to be natural in its support of *least privilege* as well as *static* and *dynamic separation of duty* policies [9, 13, 16] known to be important to the business applications of numerous organizations [11, 23]. In addition, RBAC implementations have been shown to afford administrative convenience in visualizing and managing authorizations information in a manor that is natural to the hierarchical privilege and organizational structures of most enterprises [8]. Because roles are global to the role-privilege relations of host operating systems, user privileges can be created, reviewed and deleted through user-role relations.

The concept of roles has been used in software applications for at least 25 years, but it is only within the past decade that RBAC has emerged as a full-fledged mechanism as mature as traditional mandatory access control (MAC) and discretionary access control (DAC) concepts. The roots of RBAC include the use of groups in UNIX and other operating systems, privilege groupings in database management systems [2, 24], and separation of duty concepts described in earlier papers [4, 5, 19]. The modern concept of RBAC embodies all these notions in a single access control model in terms of roles and role hierarchies, role activation, and constraints on user/role membership and role set activation. These constructs are common to the early formal definitions of RBAC proposed by various authors [6, 7, 17, 20]. A comprehensive framework for RBAC models was defined by Sandhu et al. [20], and expanded in subsequent publications [1, 18, 22].

SACMAT'01, May 3-4, 2001, Chantilly, Virginia, USA
ACM 1-58113-350-2/01/00005.

Central to RBAC is the concept of role relations. A role is a semantic construct around which access policy is formulated. Common to RBAC models are four basic elements: Users, Roles, and Permissions (sub-defined as an Operation on an Object); two types of role assignment relations: User/Role, and Role/Permission; as well as static constraint relations imposed on role assignments. Although a role is commonly defined as a job function within the context of an organization, the basic concept of a role allows for the abstraction of users into a number of security related categories that may include among others: users, administrators, organizational units, clearance levels, or integrity levels. In addition, role hierarchies are defined as a partial ordering on the inheritance relation, where role r_1 inherits r_2 if the permissions assigned to r_1 are also assigned to r_2. Similarly, to roles, object sets have been proposed to serve as abstractions of objects into a number of categories, to include: object types, classification levels, integrity levels or object groups.

In addition, operations can be categorized into operation types, for use in defining application specific permissions. For example, *deposit* and *withdraw* operations of a banking application may be applied to the objects contained in the accounts object set. Another example is the administrative operations that are applied to the RBAC sets in creating and maintaining relations and used in delegating administrative permissions from one administrator to another.

Static constraints allow for the specification and enforcement of separation policies during the construction and maintenance of the authorization database. Static constraints can take on many forms to include any combination of user, role, operation, and object sets.

In support of access decisions, a user establishes a session during which the user activates some subset of roles that he or she is authorized. Each session is a mapping of one user to possibly many roles, i.e., a user establishes a session during which the user activates some subset of roles that he or she is assigned. Each session is associated with a single user and each user is associated with one or more sessions. The permissions available to the user are the permissions assigned to the roles that are activated across all the user's sessions. By placing constraints on the activation of roles within or across a user's sessions provides a powerful means of enforcing a wide variety of least privilege and dynamic separation of duty policies. As with static constraints, dynamic constraints can be formulated on any combination of user, role, operation, and object sets.

To configure RBAC relationships in the embodiment of policy, administrative RBAC models and policy specification languages have been separately proposed [1] and in some cases integrated into the RBAC model [17, 22, 23].

To attest to its further flexibility in configuring policy, RBAC models have been extended to show support for One-directional information flow and, Discretionary access control [18], as well as integrated with other constructs in support of history-based policies such as Chinese wall and Workflow policies [3, 12].

Of the many access control technologies currently in development, RBAC models appear to be the most attractive solution for providing security features in large enterprises information infrastructures. RBAC features such as policy neutrality, principle of least privilege, and ease of management make it an especially attractive solution to the complex authorization problem.

REFERENCES

[1] Gail Ahn and Ravi Sandhu. "Role-Based Authorization Constraints Specification." *ACM Transactions on Information and System Security*, Volume 3, Number 4, November 2000.

[2] R. W. Baldwin. Naming and grouping privileges to simplify security management in large databases. In *proc. of the Symp. on Security and Privacy*, pp. 116-132. IEEE Press, 1990.

[3] E. Bertino, P. Bonatti, E. Ferrari. TRBAC: a temporal role-based access control model. In *Proc. of fifth ACM Workshop on Role based access control*, pp. 21-30, 2000.

[4] D. Brewer and M. Nash. The Chinese wall security policy. In *proc. of the Symp. on Security and Privacy*, pp. 215-228. IEEE Press, 1989.

[5] D. Clark and D. Wilson. A comparison of commercial and military computer security policies. In *proc. of the Symp. on Security and Privacy*, pp. 184-194. IEEE Press, 1987.

[6] D. Ferraiolo and R. Kuhn. Role-Based Access Control. In *Proc. of the NIST-NSA Nat. (USA) Comp. Security Conf.*, pp 554-563, 1992

[7] D. Ferraiolo, J. Cugini, and R. Kuhn. Role-based access control: Features and motivations. *In Proc. of the Annual Computer Security Applications Conf.*, IEEE Press, 1995.

[8] D. Ferraiolo, J. Barkley, and R. Kuhn. A role based access control model and reference implementation within a corporate intranet. *ACM Transactions on Information and System Security*, 2(1), 1999.

[9] S. Gaverila and J Barkley. Formal specification for rbac user/role and role relationship management. In *Proc. of third ACM Workshop on Role based access control*, pp. 81-90, 1998.

[10] L. Giuri and P. Iglio. A formal model for role based access control with constraints. *In proc. of the Computer Security Foundations Workshop*, pp. 136-145. IEEE Press, 1996.

[11] V.D. Gligor, S.I. Gavrila, D.F. Ferraiolo. On the Formal Definition of Separation-of-Duty Policies and their Composition. *Proc. Symp. on Security and Privacy*, IEEE Press,1998.

[12] W. Huang and V. Atluri. A secure web-based workflow management system. In *Proc. of fourth ACM Workshop on Role based access control*, pp. 83-84, 1999.

[13] R. Kuhn. Mutual exclusion as a means of implementing separation of duty requirements in role based access control systems. In *Proc. of Second ACM Workshop on Role based access control*, 1997.

[14] B. Lampson. Protection. *ACM Operating Sys. Reviews*, 8(1):18-24, 1974.

[15] C. McCollum, J. Messing, L. Notargiacomo. Beyond the pale of MAC and DAC – defining new forms of access control. In *proc. of the Symp. on Security and Privacy*, pp. 190-900. IEEE Press, 1990.

[16] M. Nyanchama and S. Osborn. The graph model and conflicts of interest. *ACM Transactions on Information and System Security*, 2(1), 1999.

[17] M. Nyanchama and S. Osborn. Access rights administration in role-based security systems. In J. Biskup, M. Morgenstern, and C. E. Landwehr, editors, *Database Security*, *VIII*: *Status and Prospects*, pages 37-56. North-Holland, 1994.

[18] S. Osborn, R. Sandhu and Q. Munawer. Configuring Role-Based Access Control to Enforce Mandatory and Discretionary Access Control Policies. *ACM Transactions on Information and System Security*, 3(2), 2000.

[19] Ravi Sandhu, "Transaction Control Expressions for Separation of Duties." Proc. Fourth Aerospace Computer Security Applications Conference, Orlando, Florida, IEEE Computer Society Press, December 1988, pages 282-286.

[20] R Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-based access control models. *IEEE Computer*, 29(2), February 1996.

[21] R. Sandhu. The typed access matrix model. *Proc. Symp. on Security and Privacy*, pp. 122-136. IEEE Press, 1992.

[22] Ravi Sandhu, Venkata Bhamidipati and Qamar Munawer. "The ARBAC97 Model for Role-Based Administration of Roles." *ACM Transactions on Information and System Security*, Volume 2, Number 1, February 1999, pages 105-135.

[23] R. Simon and M. Zurko. Separation of duty in role based access control environments. In *Proc. of, New Security Paradigms Workshop*, September 1997.

[24] T. C. Ting, S. A. Demurjian, and M. Y. Hu. Requirements capabilities and Functionalities of User-Role Based Security for an Object-Oriented Design Model. In S. Jajodia and C. E. Landwehr, editors, *Database Security*, *IV*: *Status and Prospects*, pages 275-296. North-Holland, 1992.

[25] D. J. Thomsen. Role-based application design and enforcement. In S. Jajodia and C. E. Landwehr, editors, *Database Security*, *IV*: *Status and Prospects*, pages 151-168. North-Holland, 1991.

Position Paper

Mikel L. Matthews
Argus Systems Group, Inc.

1. Question

Considering all factors (for example: quality of protection, performance, compatibility, ease of use), which operating system access control technique will provide the greatest overall benefit to users?

2. Position

It should be obvious that no single access control technique will provide "the greatest overall benefit" in all circumstances. The goal of the access control policy and the nature of the user community will dictate the most appropriate access control component. The difficulty of selecting an access control mechanism is compounded by the fact that historically, nearly all operating systems have incorporated only one access control mechanism, with only a small number of "specialty" operating systems having a second access control feature. Even among computer experts, few have had any significant experience with any access control technique beyond the traditional mechanism involving user identifiers and file access control lists or modes.

To answer the question above, I will briefly discuss four types of access control techniques: DAC (discretionary access control), MAC (mandatory access control), RBAC (role-based access control), and DBAC (domain-based access control).

Discretionary access control is the most popular access control mechanisms, and it is used on all UNIX/UNIX-like systems as well as the Windows NT family of operating systems. DAC provides granularity down to a user or group of users. The user's identifier associated with a process is compared to permission mode bits or an access control list to determine what, if any, access can be had to the object. DAC has been used and will continue to be used by modern day operating systems.

Mandatory access control has been in use for over three decades. MAC is most often thought of in connection with the control of information flow in a multilevel secure (MLS) system, but has been increasingly used commercial systems in highly hostile environments. MAC requires a sensitivity label to be on both objects (files, ipc, etc.) and subjects (processes). Access is allowed or denied based on the relationship between the label of the subject and the object. Unlike a discretionary policy, under a MAC policy the creator and owner of an object does not have control over its security label, and thus cannot allow access or distribute information outside the system security policy.

Role-based access control is another mechanism that has become a popular topic of study and research over the last decade. RBAC systems grant permissions based on roles, which are properties of a user's account and current session. There can be a many-to-many relationship between roles and user accounts, and a user's role set may be dynamic even within a single session. Roles may be used to allow a user to perform high level functions, such as backup and restore, as well as for access control to low level objects and records.

SACMAT'01, May 3-4, 2001, Chantilly, Virginia, USA
ACM 1-58113-350-2/01/00005.

Domain-based access control is a relatively new type of access control mechanism that is based on the concept of *access domains* (AD). Like MAC, DBAC involves labeling both subjects and objects, can be used to impose a security policy on users and programs, and can be used to create compartments or partitions within the system. However, unlike MAC, DBAC does not provide information flow protection, but it does provide separate access rights for different access modes, and permits users and processes to operate outside the control of the DBAC mechanism entirely. DBAC forms the foundation of the Argus Systems Group's PitBull LX product line.

On a system with DBAC, a process can operate in a mode that does not recognize any of the restrictions imposed by ADs. Once a process shifts into "AD aware mode" (or it is created in that mode), all AD access controls will be enforced. No process in AD aware mode can shift out of that mode. This interesting property of DBAC systems allow interesting architectures where certain users (such as administrators) or processes (such as monitoring utilities) can operate entirely outside the scope of users or applications that need to be tightly controlled.

Although the Argus PitBull LX product implements two types of ADs, those for files (FAD) and those for networks (NAD), only file ADs are discussed here. File ADs provide access protection based on a domain and the attributes of the domain. There are three attributes associated with a AD: read, write, and execute access. These were modeled after the DAC attributes to make understanding ADs easier. Processes and files may exist in multiple access domains.

DAC, MAC, RBAC, and DBAC techniques all have their uses on an OS. The quality of protection does differ for each type of access control.

MAC and DBAC can enforce partitioning of systems and can control privileged processes and accounts. These properties are particularly important for systems in high-risk environments where the danger of attack is very high (such as Internet sites) or the value of the assets being protected is very high (such as with military intelligence systems). For users at home who only need to keep from overwriting their own files or system files, DAC seems most suitable. In large distributed, networked systems RBAC mechanisms greatly simplify the overhead of managing access to critical resources or applications.

DAC and DBAC mechanisms tend to be more intuitive and require less expertise to manage than MAC or RBAC systems. For small businesses, home users, and enterprise-wide deployment, DAC and DBAC solutions may be the most cost-effective.

3. Summary

There is no one access control mechanism that provides the greatest overall benefit to users. The user must decide what type of protection is needed and to what degree it is needed. Each type of access control mechanism has a place in today's operating systems. The degree to which protection is needed depends on the requirements for that system. Trade-offs will have to be made as to performance, compatibility, and ease of use for the quality of protection needed for that system.

They Want Froot Loops
Why Industry Will Continue To Deliver Multi-Level Security

Casey Schaufler
SGI

Every parent in America is familiar with heavily sweetened, sugar coated breakfast cereals.[1] Every new parent knows, deep in their heart, that their child will never ever, under any circumstances, be feed that kind of junk. Nonetheless, there is no product market more hotly contested than kid's cereals. How can it be that products no one wants[2] turn out to battle over supermarket shelf space?

The truth of the matter is that parents buy Froot Loops, Capt'n Crunch, Coco Puffs, and Trix[3] for the same reason that computer systems vendors make multi-level secure computer systems. The only child less likely to melt down in a public place in the middle of the morning than one stuffed with Choclat Frosted Shoogar Bombs is the one who has had no breakfast at all. Similarly, the customer you loose to a competitor even though you have a better product is the one who won't come to the table if you can't offer a multi-level secure (MLS) system.[4]

Much has been made in the years since the publication of the Bell and LaPadula sensitivity model about how it doesn't meet real world needs. The commercial facility, the reasoning goes, isn't going to have Marine guards at the front desk stamping documents with big imposing words done up in an intimidating font. It's a kinder, friendlier, world outside the U.S. DoD. If the information stored on the CFOs computer is "accidentally" sent to an investment house in New York it's not like anyone is going to jail, right?[5]

We have a much better understanding of the value of intellectual property now than we did in 1985, when the only people who seemed to care about access control on computers were either military, dealing with classified information, or academic, dealing with undergraduates. Today everyone seems to appreciate the value of their VISA number, in some cases much more so on when it's on a disk drive than when it's in the hands of an underpaid waiter. The masses have discovered computer security, and like the path finders who went before them have fallen head over heals for the sexiest security technology of all, cryptography.

Even as American Express is pushing their latest crypto scheme on national television there exists a set of computing facilities for whom the radiant glow of cryptography is not enough. The realization that you can encrypt messages for transmission, you can encode them for storage, but that you have to bear all when it's time to use them brings back the notion of access control, and today that means strong access control.

When a computer buyer goes looking for a strong access control scheme she doesn't call the National Security Agency or the Computer Science chair at Southeast Arizona State University, she calls the same person who solves all of her information system problems. When the head of systems administration finishes laughing she calls the corporate preferred[6] system Sales Rep, and asks for their solution. The Sales Rep, having never dealt with this kind of thing before, looks in the price book[7] and finds that trusted version they did for a government contract a couple years ago. Problem solved.

Before we start tossing around terms like "Bill of Goods," and "Buggy Whip," do consider that most cases in which strong access control is desired there are a small number of groups who wish to share one expensive[8] resource. Also keep in mind that these groups don't trust each other, or they wouldn't be going out of their way to maintain separation. For these people, a simple model access control is preferred over something with generality, which offers addition configuration choices to confuse the aforementioned, now actively hostile, head of systems administration.

Recent experience in the market indicates that the ideal strong access control scheme should protect the system from the users[9] and separate the user groups either completely or such that one group is superior to the others. The former scheme is an example of MLS categories. The later scheme is MLS levels.

There are those who would have us abandon MLS systems in favor of more general schemes. Alternatives suggested include Domain Type Enforcement (DTE),[10] Role Based Access Control (RBAC), and Pluggable Policy Modules (PPM). Each of these mechanisms secedes in the goal of generality.

Advocates of DTE claim, although they have not demonstrated an implementation, that DTE can be used to emulate Bell & LaPadula sensitivity. Fans of RBAC are inclined to explain why it's great, but don't seem to have put an entire system together

[1] Parents who claim otherwise really need to spend more time with their children!

[2] The power of advertising is admittedly strong, however I have yet to have my child ask twice for an inferior (in the jargon of parenthood, yucky) product. Parents actually do the buying.

[3] All this to get a product plug in, SGIs Trusted Irix product is called Trix by its aficionados.

[4] Customers act like spoilt children in may other ways as well. I will stick to the topic at hand.

[5] Yes, you're correct, this is sarcasm.

SACMAT'01, May 3-4, 2001, Chantilly, Virginia, USA
ACM 1-58113-350-2/01/00005.

[6] As in, I preferred the root canal to the extended staff meeting.

[7] The Sales Rep looks in the price book whenever he has a problem as he has no interest whatever in anything that does not have a commission attached to it.

[8] In 2001 terms, US$3,000,000 is a good lower bound

[9] Trix uses Biba integrity for this, but B&LaP can be used instead

[10] Some fine work done in the NSA to bring this out

using it. The internal issues of PPM, especially regarding locating every place a policy decision might possibly be required, have hampered the credibility of this approach in a real system.[11]

In the real world[12] we find little interest in strong access controls. In the cases where we do find it, the traditional[13] MLS scheme, with it's hierarchical and set based features, usually has one more feature than the customer actually has use for. Would it be fun to experiment with two-man switch, union seniority, or number of patches accepted by Linus policies? Of course it would. But I can't sign up to make whole-grain, fruit juice sweetened, high fiber granola.

The customers are buying Froot Loops.

[11] "You'll never get this past Linus!"

[12] As experienced by Dilbert and me

[13] Classic, Old Fashioned, Stone Age, if you prefer.

Which Operating System Access Control Technique Will Provide the Greatest Overall Benefit to Users?

Stephen Smalley
NAI Labs

Mandatory access controls that are flexible in their support for security policies and that are directly integrated into the service-providing components of the operating system will provide the greatest overall benefit to users. Current mainstream operating systems only provide discretionary access controls and place the burden of security on the individual end users. Even worse, most systems only provide a weak form of discretionary access control in which the discretionary policy can be changed by any code executed by users, regardless of the trustworthiness of that code. These systems are incapable of enforcing the separation of information based on confidentiality or integrity requirements, and they are incapable of protecting users from malicious software. As illustrated by the examples in [6], the absence of operating system mandatory access controls leaves application security mechanisms vulnerable to tampering and bypass, and malicious or flawed applications can easily cause failures in system security when only discretionary access controls are available.

Operating system mandatory access controls can be implemented in a variety of ways. A technique that has gained popularity is the use of kernel space *wrappers* [4, 5, 8]. Wrapper-based techniques can offer several advantages over conventional implementations, such as increased ease of integration and maintenance. They can also be applied to closed source COTS systems that provide a mechanism like loadable kernel modules.

However, wrapper-based techniques also have some serious limitations. As discussed in Section 3.2 of [9], wrapper-based techniques are limited by the existing functional interface that is provided by the system. This limits the abstractions and services that a wrapper-based technique can control. Such techniques are also limited in their ability to make use of internal system state, and frequently must maintain redundant state in order to make decisions. The level of abstraction of the existing interface may also cause difficulties in guaranteeing the uniqueness of objects or in ensuring that the system remains consistent from the time that checks are performed to the time that the service is provided. For example, pathname-based system calls pose problems for wrappers in the areas of object aliasing, multi-component pathnames, and changes in the mapping from pathname to object. Wrapper-based techniques also cannot address subsequent changes to the security policy, particularly the revocation of permissions that are implicitly retained in the state of the system such as open file descriptions, established connections or in-progress operations.

Based on these limitations of wrapper-based techniques, it seems preferable to directly integrate mandatory access controls into the service-providing components of the operating system. Several different kinds of mandatory access controls might be integrated into an operating system. In traditional trusted operating systems, mandatory access controls have been tightly coupled to lattice-based models such as the Bell-LaPadula [1] (BLP) model of multi-level security (MLS) and the parallel model for integrity provided by Biba [2]. This tight coupling has limited their applicability, since the mandatory access controls of these systems do not address other important security requirements such as fine-grained least privilege, protected subsystems and assured pipelines, or dynamic separation of duty.

Abstractly, traditional mandatory access controls provide strong guarantees for the separation of information based on its confidentiality or integrity characteristics. However, these models also require that many important system functions be placed into trusted subjects that operate outside of the constraints of the policy model. Hence, the security of the entire system typically devolves to the security of the trusted subjects, and these systems frequently require many trusted subjects for normal operation. Furthermore, mechanisms for limiting these trusted subjects to least privilege are typically coarse-grained and must be provided separately from the ordinary mandatory access control mechanism.

A different form of mandatory access control known as Type Enforcement [3] (TE) offers several advantages over the traditional model. Security labels are not required to form a partial order, so intransitive relationships can be defined to support protected subsystems and assured pipelines. The security policy logic is defined through a set of separate tables, so the security policy can be easily customized. Controls over program execution and changes in access rights (domains) are explicitly defined in the TE tables, so no separate mechanism is required for this purpose. No trusted subjects that can operate outside of the constraints of the TE tables are needed, since the tables can be configured to grant exactly those access rights that are required for privileged subjects. Users and individual programs can be easily limited to least privilege through the definition of domains and domain transitions.

However, TE also has its limitations. Since the security policy logic is defined through tables and there are no implicit relationships among labels, it would be cumbersome to express a complex BLP or Biba lattice using TE, and it is more difficult to verify that TE tables provide the same guarantees for the separation of information. TE also does not directly address dynamic security policy requirements, which are often needed in real-world environments.

Since no single model is likely to meet all user's needs, operating systems must be flexible in their support for security policies. Policy flexibility requires a mandatory access control architecture that provides clean separation of policy from enforcement and well-defined interfaces for obtaining policy decisions. In order to support dynamic security policy requirements, this architecture must provide a mechanism for supporting policy changes and in particular for revoking permissions, including permissions that are

SACMAT'01, May 3-4, 2001, Chantilly, Virginia, USA
ACM 1-58113-350-2/01/00005.

implicitly retained in the state of the system. The architecture and any implementations of it must allow authorized users to easily customize the security policy, including the addition of new policy components, so that users can express security policies in the manner that is most appropriate to their security requirements rather than requiring all policies to be mapped into a single model.

The architecture must also address a number of acceptability concerns for mandatory access controls. It must ensure that the performance overhead of the mandatory access controls is minimal. Unlike traditional implementations of mandatory access controls, the architecture cannot take advantage of policy-specific information to optimize the enforcement mechanism. The mandatory access controls must operate transparently to applications and users except when access failures occur. Compatibility problems should only occur when the security requirements of a particular security policy conflict with the functional behavior of existing applications.

To provide a high quality of protection, an operating system access control technique must be mandatory, must be directly integrated into the operating system services, and must be able to support a wide variety of real-world security requirements. To provide ease of use, the technique must allow users to express security policies in the manner most appropriate to their requirements. To be acceptable to users, it must not impose a significant performance overhead, and it must operate transparently to applications and users except when access failures occur. A technique that meets these requirements will be of the greatest overall benefit to users. A working example of such an access control technique can be found in NSA's Security-Enhanced Linux prototype[7]. This prototype is an implementation of a flexible mandatory access control architecture called Flask[9] in the Linux operating system.

References

[1] D.E. Bell and L.J. LaPadula. Secure Computer Systems: Mathematical Foundations and Model. TR M74-244, The MITRE Corporation, May 1973.

[2] K.J. Biba. Integrity Considerations for Secure Computer Systems. TR-3153, The MITRE Corporation, May 1977.

[3] W.E. Boebert and R.Y. Kain. A Practical Alternative to Hierarchical Integrity Policies. In Proceedings of the Eighth National Computer Security Conference, 1985.

[4] T. Fraser, L. Badger, and M. Feldman. Hardening COTS Software with Generic Software Wrappers. In Proceedings of the 1999 IEEE Symposium on Security and Privacy, pages 2-16, May 1999.

[5] D.P. Ghormley, D. Petrou, S.H. Rodrigues, and T.E. Anderson. SLIC: An Extensibility System for Commodity Operating Systems. In Proceedings of the USENIX 1998 Annual Technical Conference, June 1998.

[6] P.A. Loscocco et al. The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In Proceedings of the 21st National Information Systems Security Conference, Oct 1998.

[7] P.Loscocco and S. Smalley. Integrating Flexible Support for Security Policies into the Linux Operating System. Technical Report, NSA and NAI Labs, Oct 2000. http://www.nsa.gov/selinux/docs.html.

[8] T. Mitchem, R. Lu and R. O'Brien. Using Kernel Hypervisors to Secure Applications. In Proceedings of the 13th Annual Computer Security Applications Conference, Dec 1997.

[9] Spencer, R. et al. The Flask Security Architecture: System Support for Diverse Security Policies. In Proceedings of the Eighth USENIX Security Symposium, Aug 1999.

Statement for SACMAT 2001 Panel

Robert Watson
NAI Labs and the FreeBSD Project

A variety of mandatory access control technologies have been developed and deployed in the past, in the form of research prototype operating systems, extensions to commercial products, and adaptations of open source systems. Up until now, trusted systems have at best been a small niche market. However, we appear to be at an interesting turning point, as the prospects for improved accessibility of trusted systems seems to be on the rise as a result of the open source movement. There are a number of projects seeking to implement new access control mechanisms based on a variety of free UNIX-like systems, both modeling the systems after existing trusted OS products and exploring new architectures.

One risk associated with the development of a variety of systems is that their access control models will be incompatible, making it difficult or impossible to write portable applications. Selecting a single model for trusted system design would allow the development of consistent APIs and well-integrated applications. Yet despite years of research, we still don't know what the optimal access control mechanism is. Some consumers believe strongly in the MLS confidentiality policy, or Biba integrity policy. Type Enforcement may offer a more flexible vehicle for policy expression. Role-based access control may offer greater parallels between the policy expression and real-world human activities. And DTE offers us a low-management labeling solution building on TE-like concepts. These and other mechanisms provide a wide variety of options with only low levels of compatibility. And none of this is helped by clear deficiencies in deployed discretionary access control systems. The reality is that at this point, when addressing a problem, we turn to a suite of possible solutions, or discover that we're fudging aspects of the application by generalizing a single insufficiently broad policy tool. The natural question that needs to be asked is: what if there is no one right access control model for everyone?

Past research has explored both flexible enforcement and access control policy as a means to address the need for diverse access control mechanisms in operating systems. Of particular interest are those mechanisms that provide fixed enforcement points but flexible policy; there has been increasing exploration of flexible access control environments that offer to abstract the actual policies and decision making away from the enforcement points. These efforts include NSA's Flask model, as well as recent interest in GACI on the Linux platform.

However, the idea of an entirely flexible policy environment raises both substantial challenges from the implementation perspective, especially with a desire to maintain performance, and from the perspective of providing a consistent and well-defined environment to application writers. The inability of applications to adapt to changing security environments has long been a problem in traditional trusted operating systems: applications discover a variety of new and unexpected failure modes, often failing poorly and possibly failing open. Similarly, such over-arching flexibility makes it difficult to develop, test, and deploy solutions due to user and system manager expectations and experience. The problems of too flexible an environment are similar to the problems of highly divergent security solutions in the face of a desire for portable applications. As a result, even if we don't select one "true model" for access control, it makes sense to try and explore and develop constraints for viable models that applications writers will be able to rely on.

SACMAT'01, May 3-4, 2001, Chantilly, Virginia, USA
ACM 1-58113-350-2/01/00005.