

The Pursuit of a Standardized Solution for Secure Enterprise RBAC

- Improve Productivity & Network Security
- Reduce IT Administration Costs and Complexities
- Ensure Regulatory Compliance

About the Company

BeyondTrust is the only provider of Privileged Access Lifecycle Management (PALM) solutions for heterogeneous IT environments. The BeyondTrust suite of products reduces the risks associated with insider sabotage and theft of proprietary data, while documenting accountability to support increasing demands of regulatory compliance required across many industries.

PALM is a technology architecture framework consisting of four continual stages running under a centralized automated platform: Access to privileged resources, Control of privileged resources, Monitoring of actions taken on privileged resources, and Remediation to revert changes made on privileged IT resources to a known good state.

More than half of the companies listed on the Dow Jones rely on BeyondTrust to secure their enterprises. BeyondTrust customers include eight of world's 10 largest banks, seven of world's 10 largest aerospace and defense firms, and six of the 10 largest U.S. pharmaceutical companies, as well as renowned universities. BeyondTrust's customer retention rate is over 90%. The company is headquartered in Los Angeles, California, with East Coast offices in Greater Boston and EMEA offices in London, UK. For more information, visit www.beyondtrust.com.

BeyondTrust Corporation • 30401 Agoura Road • Agoura Hills, CA 91301 • USA

Legal Disclaimer

BeyondTrust, Privilege Manager and PowerSeries are trademarks of BeyondTrust Corporation. This document is for informational purposes only. BeyondTrust offers no warranties, express or implied, in this document. Microsoft, Microsoft Outlook, Microsoft Exchange, Microsoft Internet Explorer, Microsoft Windows, Microsoft Windows 2000, Microsoft Windows XP, and Microsoft Windows Server 2003 are trademarks of Microsoft Corporation. Other names mentioned herein may be trademarks of their respective owners.

© 2009 BeyondTrust Corporation. All Rights Reserved.

Table of Contents

- Role-based Access Control 4
- NIST RBAC Standard 5
 - Isn't RBAC the same as ACLs? 5
 - Flat RBAC Model Rationale:5
 - Hierarchical RBAC Model Rationale:6
 - Constrained RBAC Model Rationale:6
 - Symmetric RBAC Model Rationale:6
- OS RBAC Models..... 6
 - Sun RBAC Features & Weaknesses 6
 - HP-UX Features & Weaknesses 7
 - Linux RBAC Features & Weaknesses 7
- PowerBroker vs. OS RBAC 7
 - How PowerBroker Works..... 8
 - PowerBroker Functionality In-Depth..... 9
 - PowerBroker Customization & Flexibility9
 - Command Requests9
 - Policy Language & ACLs 10
 - Logging & Complete RBAC Security..... 10
- Real-Life Examples when Implementing PowerBroker 11
 - Example 1A 11
 - Example 1B 11
 - Example 1C..... 12
 - Example 1D 12
- Summary 13
- Cited Sources..... 14

Abstract

This white paper focuses on role-based access control (RBAC) and how RBAC is a rich and open-ended technology which is evolving as users gain experience with it. We will also spotlight the accepted NIST model organized into four levels of increasing functional capabilities. The NIST RBAC model has the potential to provide an effective way to manage access controls across a large number of disparate systems. Lastly, we explain how using PowerBroker to implement role-based access control allows an organization to efficiently deploy key security and compliance requirements not always found in operating system (OS) RBAC implementations, including separation of duties and audit trails.

Role-based Access Control

One of the most challenging problems when managing large networks comes from the high costs and complexities of IT security administration. To help counteract these threats, the conception of RBAC (Role-based Access Control) began focusing around multi-user and multi-application online systems that were pioneered in the 1970s. The essential notion of RBAC is establishing permissions based on the functional roles within the enterprise, and then applying a spotlight on aptly assigning users to a role or set of roles. RBAC has evolved into a dominant blueprint for advanced access control mainly because the end-goal is to reduce the complexity and cost of security administration for large or small networked applications.ⁱ

From theoretical inception to the present practical interpretation, most information technology vendors have incorporated RBAC, in some form or fashion, and technology is finding applications in a broad spectrum of industries ranging from health care to defense, in addition to the mainstream commerce systems for which it was designed. Vendors have incorporated RBAC features into their database, system management and operating system products.

Ironically, while there has been a growing global market for a product solution to infuse an RBAC concept into reality, every vendor's products have been independently developed without any attempt at standardizing salient RBAC features.

As a result, RBAC is seen as an amorphous concept interpreted in different ways by various researchers, organizations and system developers ranging from simple to elaborate and sophisticated RBAC models. Without standardized models that can be applied and understood between organizations, it becomes more apparent that these products may never provide the true reduction in cost and complexity that RBAC was originally intended to solve.

After acknowledging the problematic circumstances that could be achieved due to no RBAC regularity, [NIST \(National Institute of Standards and Technology\)](#) produced the NIST RBAC model to provide a blueprint of standardization over a core set of RBAC features. The benefits of this undertaking include a common set of benchmarks for vendors, already developing RBAC mechanisms, to implement into their enterprise operations. It will also give IT consumers, the principal beneficiaries of RBAC technology, a basis for the creation of uniform acquisition specifications.ⁱⁱ

This paper will discuss how integrating Symark's PowerBroker with the basic RBAC implementations of Sun, HP and the Linux community, provides an all-encompassing solution to privileged access control on UNIX and Linux platforms. Each RBAC implementation varies in its capabilities and method of management. In a multi-platform environment, these differences introduce higher administration hours and costs because the various RBAC models are not consistent in administration and operation methodology.

The differences among these implementations also increase the potential for misconfiguration and related security issues. Lastly, most vendors' RBAC solutions are to some extent "host-centric." Meaning maintenance operations may have to be performed on each host. PowerBroker implements a consistent, cross-platform system of access control across all the major UNIX and Linux platforms, providing for centralized and cost-effective access control.

This paper will illustrate how RBAC is a rich and open-ended technology which is evolving as users gain experience with it. We will focus on the NIST model organized into four levels of

increasing functional capabilities. The NIST RBAC model has the potential to provide an effective way to manage access controls across a large number of disparate systems. Using PowerBroker to implement role-based access control allows an organization to efficiently deploy key security and compliance requirements not always found in operating system (OS) RBAC implementations, including separation of duties and audit trails. Lastly, using Symark PowerBroker will bolster one's organization's RBAC model and strengthen its enterprise IT security and compliance.

NIST RBAC Standard

RBAC implements the principle of "least privilege," giving the user the minimum privileges required to perform a given task. This model is the antithesis to the root access model of UNIX and Linux systems, which provides unlimited access to the root account to complete tasks. With today's need for heightened security amongst computing systems, the least privilege model has become a standard approach, providing protection against intended or unintended access and changes to your UNIX and Linux hosts.

Isn't RBAC the same as ACLs?

RBAC differs from ACLs (access control lists) used in traditional discretionary access control systems in that it assigns permissions to specific operations with meaning in the organization rather than to low level data objects. For example, an access control list could be used to grant or deny "write access" to a particular system file, but it would not dictate how that file could be changed. In an RBAC-based model, an operation might be to create a 'credit account' transaction in a financial application or to populate a 'blood sugar level test' record in a medical application. The assignment of specific permissions to perform a particular operation is beneficial, because the operations are granular in meaning within the application.

RBAC has been shown to be particularly well-suited to separation of duties requirements, which ensure that two or more people must be involved in authorizing critical operations. Necessary and sufficient conditions for safety of separation of duties in RBAC have been analyzed. An underlying principle of separation of duties is that no individual should be able to affect a

breach of security through dual privilege. By extension, no person may hold a role that exercises audit, control or review authority over another concurrently held role.ⁱⁱⁱ

Although RBAC is often considered a single-access control and authorization model, RBAC is in fact composed of a number of models, each of which is fit for a specific security management application. As such, the NIST RBAC model has been organized into four separate levels of increasing functional capability, with a specific rationale for its deployment.

The following chart (Fig. 1) illustrates the four level capability and rationale:

Level	Name	RBAC Functional Capabilities
1	Flat RBAC	<ul style="list-style-type: none"> • Users acquire permissions through roles • Must support many-to-many user-role assignment • Must support many-to-many permission-role assignment • Must support user-role assignment review • Users can use permissions of multiple roles simultaneously
2	Hierarchical RBAC	Flat RBAC + <ul style="list-style-type: none"> • Must support role hierarchy (partial order) • Level 2a requires support for arbitrary hierarchies • Level 2b denotes support for limited hierarchies
3	Constrained RBAC	Hierarchical RBAC + <ul style="list-style-type: none"> • Must enforce separation of duties • Level 3a requires support for arbitrary hierarchies • Level 3b denotes support for limited hierarchies
4	Symmetric RBAC	Constrained RBAC + <ul style="list-style-type: none"> • Must support permission-role review with performance effectively comparable to user-role review • Level 4a requires support for arbitrary hierarchies • Level 4b denotes support for limited hierarchies

Figure 1: RBAC Model Variations Organized by Levels

Flat RBAC Model Rationale:

The Flat RBAC model captures the features of traditional group-based access control as implemented in operating systems (OS) through the current generation. As such, it is a widely

deployed and familiar technology. The features required of a Flat RBAC model are obligatory for any form of RBAC and are almost obvious. The main issue in defining this model is in determining which features to exclude. These features are deliberately kept to a minimum, and accommodate traditional but robust group-based access control.^{iv}

Hierarchical RBAC Model Rationale:

The Hierarchical RBAC model adds functionality to the Flat RBAC standard to provide for hierarchies of “Roles,” typically based upon “an organization’s line of authority and responsibility.”^v In addition to a Flat RBAC model, “Role” hierarchies, in the form of arbitrary partial orders, are the single most desirable feature.

Essentially, role hierarchies define an inheritance relationship among various “Roles.” For example, if there are separate “Roles” for providing access to maintain the printers on the accounting hosts and the manufacturing hosts, there could be a “Global Printer Maintenance Role” defined above them in the hierarchy that inherits the permissions granted in each inherited “Role.” Therefore, a user that was assigned the “Global Printer Maintenance Role” would inherit that ability to maintain both the accounting and manufacturing printers. Using this security model will help simplify the maintenance needed in large organizations where there are many different “Roles” that need to be created, and various levels of authority that need to be granted.

Constrained RBAC Model Rationale:

The Constrained RBAC model provides for the separation of duties, which is often mentioned as one of the driving motivations of RBAC. This model provides for both static and dynamic separation of duty relationships. Static separation of duty relationships ensure that conflicting “Roles” can never be assigned to the same user. For example, a user cannot be granted a “Role” to make configuration changes on a host and also be granted a “Role” to review the logs of those configuration changes. Separation of duties is practiced routinely in organizations and should be supported by sophisticated access control products. It is classified after the Hierarchical RBAC model, because most existing products

more often support hierarchies than separation of duties.

Symmetric RBAC Model Rationale:

The Symmetric RBAC model adds a requirement for permission-role review similar to user-role review introduced in Level 1. Thus, the “Roles” to which a particular permission is assigned, can be determined as well as permissions assigned to a specific “Role.” This requirement has been deferred until Level 4 because it can be intrinsically difficult to implement in large-scale distributed systems.^{vi}

OS RBAC Models

This section briefly covers the basic RBAC features that are offered in UNIX / Linux systems. While RBAC is offered, there are many inherent weaknesses found within these models. However, by integrating Symark’s PowerBroker with these OS RBAC models, the time, effort, and cost of maintaining privileged access will be dramatically reduced due to the centralized secure architecture and consistent operations of PowerBroker.

Sun RBAC Features & Weaknesses

Sun Microsystems introduced RBAC into the Solaris 8 models, adopting the functionality from the Trusted Solaris OS. Sun operating systems implements the Flat RBAC model along with support for the Hierarchical RBAC model using “Rights” (Permissions) and “Authorizations” (Sessions). The hierarchy support in Authorizations is based on wildcards and a left-to-right naming scheme, which limits each authorization to a single parent.^{vii} However, it is important to note that there is no direct support for the Constrained or Symmetric RBAC model.

Another limitation in the Solaris RBAC implementation is that the “Roles” do not support host-specific restrictions. Meaning any “Roles” that are not intended for every Solaris host within your environment would need to be maintained individually on each relevant host. This would increase the maintenance burden and the likelihood for many misconfiguration issues.

HP-UX Features & Weaknesses

HP introduced RBAC into the HP-UX 11.23 models, and requires HP-UX 11i v2 (Sept. 2004 or higher) for RBAC to function properly. HP-UX implements the Flat RBAC and Hierarchical models. Hierarchical RBAC support is provided through two means: Using “wildcards” and a left-to-right naming scheme similar to the Solaris RBAC implementation. The main difference between HP and Solaris is HP has the ability to assign “Roles” as “sub-Roles” to other “Roles.” The HP-UX RBAC implementation also supports the ability to designate a default object for each command operation in “Permission.”

There are two major concerns with the HP-UX RBAC implementation. First, as the RBAC configuration files are contained on a host, if all the “Roles” do not properly protect against editing or overwriting of these files there is the potential for un-intended privilege escalation. Secondly, whenever a file is upgraded, all fine-grained privileges on that file are lost and will need to be reconfigured. Again, this would increase the maintenance burden and the likelihood for many misconfiguration issues.

Linux RBAC Features & Weaknesses

RBAC functionality is included in most current Linux distributions as part of the SELinux extensions. SELinux is built into the v2.6 kernel, and is available as kernel patches for kernel versions back to v2.2. Conversely, RBAC in SELinux is quite different from other RBAC implementations, however, in that it is built as an access control management layer on top of SELinux’s TE (Type Enforcement) security. Any potential security found in SELinux is found from securing the host with TE security, then using RBAC as a means to map TE security access for users.

SELinux RBAC implementation provides the functionality similar to the Flat RBAC model, but uses some different constructs. For example, the concepts of “object” and “operations” being combined into “Permissions” are replaced by “TEs” or “Types” and “Domain.” Though different terms are utilized, much of the same functionality is achieved. Hierarchical RBAC can also be achieved with SELinux using “Role-dominance” relationships, where one “Role” can be made the

parent of another “Role,” and inherit the child “Role’s” domains.^{viii}

Although the use of Constraint Definitions can limit the ability to move from “Role-to-Role,” and provide functionality similar to the Constrained RBAC’s static separation of duties, Constrained or Symmetric RBAC models are not supported with SELinux. It is also important to note that SELinux RBAC’s implementation is configured locally on a host. In other words, implementing SELinux RBAC across a large number of hosts will require configuration on each host, along with a customer-defined system to copy “Role” configuration files to the appropriate hosts.

Furthermore, when RBAC is first initialized on a host the host must be rebooted to reinitialize some of the domains. Also, once RBAC is active, any changes to policy require a recompile and reload of the policy into the kernel. Policy also must be reloaded when a user is added to the system that needs greater rights than the generic user identity. This would likely increase the maintenance burden and the likelihood for many misconfiguration issues.^{ix}

PowerBroker vs. OS RBAC

After comparing and contrasting the different UNIX / Linux basic RBAC implementations, it is highly evident that there are fundamental differences between them. The following five points illustrate the advantages gained by integrating Symark PowerBroker into a UNIX / Linux environment:

1. PowerBroker is a cross-platform solution, running on nearly all UNIX and Linux variants:

As noted in the previous section, each of the RBAC implementations is specific to one platform. In a typical large environment, with many platforms of different flavors, each of the different RBAC implementations would need to be installed and maintained. PowerBroker solves the maintenance burden alone for IT administrators.

For example, commands in PowerBroker can also be executed cross-platform (i.e. a privileged request can be submitted on a Linux host, and then be requested to run on a Solaris host).

PowerBroker's granular policy language also makes it possible to evaluate a request made by a user and reschedule it to run on a different host (i.e., HP-UX), all without user knowledge or intervention.

2. PowerBroker can delegate the use of UNIX / Linux accounts other than the standard *root* account:

As noted previously, most of the RBAC implementations only delegate *root* access. The exception is HP-UX RBAC, which can also delegate other privileged accounts as PowerBroker does. However, PowerBroker can go one step further by determining what account a request should actually execute under, and with or without the user's knowledge, converting that task request to run under the proper account.

3. PowerBroker logs access to central, redundant Log Hosts:

The RBAC implementations that do provide logging write the logs locally to a mounted remote drive or to syslog. PowerBroker logs all requested tasks, whether accepted or rejected, and all environmental information. These event and I/O ("keystroke") logs enable an enterprise to meet accountability and enforcement of access controls required by compliance laws. Reports are also exportable in CSV and XML format. In addition, PowerBroker supports over 25 encryption methods of policy and configuration files, log servers and network traffic.

4. PowerBroker policy is controlled centrally, making it less susceptible to unintended privilege:

With PowerBroker, any policy changes do not require any action on the remote hosts. This is a huge advantage when compared to Solaris RBAC's requirement to restart the name service cache or SELinux's recompiling and reloading of policy every time a change is made. As the current PowerBroker policy is evaluated, at each privileged request, as soon as a policy change is implemented, all requests in the PowerBroker implementation immediately follow the new policy.

5. PowerBroker is completely unobtrusive to UNIX and Linux environments:

PowerBroker can be installed, configured, used, reconfigured, and uninstalled without rebooting any hosts, or making any changes in configuration. PowerBroker does not require any special versions of applications to achieve maximum benefits. Therefore, all of PowerBroker's functionality can be achieved without any changes to the software on your machines. Lastly, there is never any notable performance impact when PowerBroker is running on your machines. PowerBroker sessions are similar to telnet sessions, in that they both require minimal system resources. All these advantages contribute to making PowerBroker a more effective solution than OS RBAC implementations, and can only enrich any RBAC model that an enterprise currently has in place. PowerBroker's centralized architecture and consistent operation lower the time, effort and cost of maintaining privileged access.

How PowerBroker Works...

PowerBroker provides policy-driven access control and logging across nearly all UNIX and Linux platforms. Using a centralized policy server called *PowerBroker Master Host*, every request for privileged access through the PowerBroker interface is evaluated by policy.

In this policy process, the request can be either accepted or rejected, event and I/O logging can be turned on or off, or the request can be modified. Modifications can be as simple as removing an incorrect command-line parameter or redirecting the request to run as a different user on a different host.

Access can also be flexibly managed by day, date, time, user ID and/or group membership. PowerBroker's rich policy language provides the capability to address almost any conceivable business or compliance requirement for privileged access.

The following chart (Fig. 2) illustrates PowerBroker's ability to enrich any RBAC model:

Feature	Solaris RBAC	Sudo	Symark PowerBroker
Authorization	✓	✗	✓
Cross-Platform Support	✗	✓	✓
Kerberos Support	✓	✓	✓
Solaris BSM Auditing	✓	✗	✓
RUID	✓	✓	✓
EUID	✓	✗	✓
RGID	✓	✗	✓
EGID	✓	✗	✓
Hierarchical Profiles	✓	✗	✓
Host-Specific Policy	✓	✓	✓
Netgroup Policy	✗	✓	✓
Require Password	✗	✓	✓
Restrict Users	✓	✗	✓
Profile Shells	✓	✗	✓
Control cmd Arguments	✗	✓	✓
Privileges/ Capabilities Aware	✓	✗	✓
Authenticate as Self	✗	✓	✓
Control Sensitive Environment Variables	✓	✓	✓
Control UMASK	✗	✓	✓
Fine-Grained Policy Admin.	✓	✗	✓
Default Profiles for OS Admin	✓	✗	✓

Figure 2 – RBAC Feature Comparisons*

PowerBroker Functionality In-Depth

With PowerBroker, privileged access is not restricted to just the *root* account. PowerBroker can execute requests as any valid UNIX or Linux user, such as accessing application or database accounts. The account under which that user will

run the request can be specified by the user when the request is submitted, and it then can be evaluated and changed during policy processing.

PowerBroker Customization & Flexibility

Although PowerBroker provides strong *root* and command delegation, it is highly customizable and flexible to the needs of an enterprise. Generally, this customization begins with the *pb.settings* file, which lists a number of parameters that can be defined to best suit your security policy. These parameters are stored on each machine in the */etc/pb.settings* file and include the following:

Masters: Allows administrators to define PowerBroker master servers to either request or accept permissions.

Log Servers: Allows administrators to define a single, central server to consolidate all PowerBroker events and I/O logs.

Logging: Allows administrators to define the file names where various data will be logged, including event, I/O and error logs.

Encryption: Enables AES, DES, 3DES, or up to 25 other encryption technologies for encryption of all PowerBroker communication among submitting machines, the PowerBroker Master Server, and executing machines. All policies and log files can be encrypted, further securing PowerBroker authorization.

SSL: Allows administrators to enable PKI (public-key infrastructure) support, using SSL for certificate and key management.

Kerberos: PowerBroker can use Kerberos to authenticate its various components and to exchange encryption-key information.

Firewalls: PowerBroker can operate in very secure environments where firewalls are used to separate clients and servers.

Command Requests

PowerBroker provides multiple interfaces for making privileged access requests, all of which are evaluated by policy and securely logged. The

pbun command can be used to execute single commands or scripts, as well as to open a shell as a privileged user. The PowerBroker shells, *pbsh* and *pbksh*, are secured equivalents of *sh* and *ksh*, respectively.

Each command executed through the shell, as well as the opening of the shell itself, is evaluated by policy established by an organization's security model. PowerBroker provides secured versions of several common UNIX and Linux utilities (*pbvi*, *pbnvi*, *pbmg*, *pbumacs* & *pbless*). An excellent example of this additional security provided is when *pbvi* allows the editing of a file as the *root* or other privileged user, but disallows accessing other files or spawning new processes as the privileged user.

The following diagram (Fig. 3) illustrates how PowerBroker essentially works:

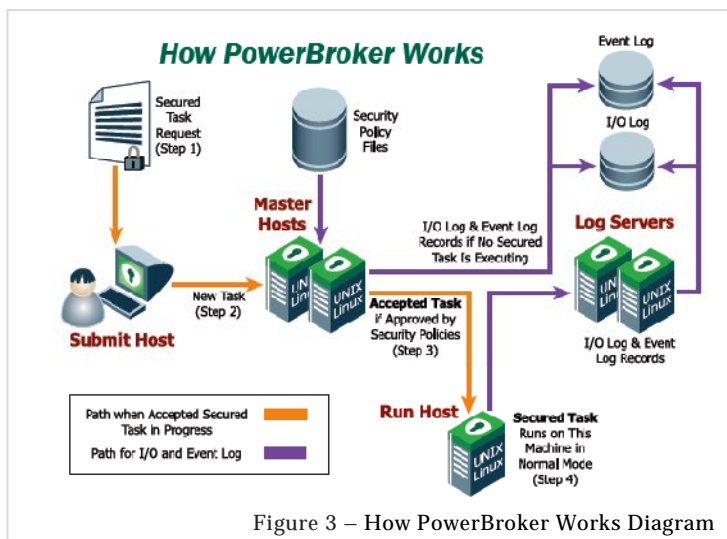


Figure 3 – How PowerBroker Works Diagram

Policy Language & ACLs

PowerBroker policy language can be maintained either using a text editor or through PowerBroker's web-based interface. The Policy Editor within the PowerBroker interface presents the policy in a tree-based hierarchy, automatically broken down into the programmatic functions of the policy. Web-based Smart Editors, which include online command syntax, can also be used to quickly construct policy components. Like any good programming language, PowerBroker's policy language allows compartmentalizing logic in individual policy files,

and then using "include statements" at run-time to implement the compartmentalized logic.

PowerBroker's policy language also includes ACL (Access Control List) syntax. ACLs simplify the definition of access privileges. Using a simple list, a PowerBroker administrator can specify the most commonly used PowerBroker access control mechanisms for users without having to compose PowerBroker policy scripts.

ACLs provide the capability to accept or reject access based on user, command, host the request was submitted on, and host the request will be executed on. The ACL can also be extended with conditional and pre-execution functions written within the PowerBroker policy language. The ACL syntax commands "accept" and "reject" can be freely intermixed in any PowerBroker policy, allowing an organization to begin with a simple ACL-based access control system, and then to add PowerBroker policy language extensions to address business and technical requirements and security models.

Logging & Complete RBAC Security

PowerBroker event and I/O logging is performed on PowerBroker Log Hosts, which can be on the same or a different machine than the PowerBroker Master Host, or any other PowerBroker component for that matter. During the policy processing, the type of logging to be performed, as well as the log file that the entries will be written to can be set according to an enterprise's requirements.

Requests can be logged into different log files based on user, host or any other variable evaluated during policy processing. Authorized users, via either a CLI (Command-line Interface) or web-based PowerBroker interface, can review all log entries.

The PowerBroker architecture of performing policy processing on remote Master Hosts and logging on remote Log Hosts provides an inherent separation of duty relationship between PowerBroker Administrators and PowerBroker users, as many PowerBroker users do not need to have any access to either Master or Log Hosts. This architecture also helps prevent unintended privilege escalation issues by isolating the policy

files from the hosts where the PowerBroker users will be granted access.

Real-Life Examples when Implementing PowerBroker

PowerBroker can be used to construct a Flat, Hierarchical, Constrained, or Symmetric RBAC model that will bring together all the benefits of your security policy. The ACL syntax can be used for PowerBroker's "accept" and "reject" commands. In its basic form, the "accept" statement designates the approval of a privileged request and begins its execution. A "reject" statement designates the disapproval of a privileged request and halts its execution. Typically, "accept" and "reject" statements are included as the decision points in procedural code. These statements can also be single-line approvals or disapprovals.

Example 1A

Suppose you would like to give access permissions to your Backup Administrator, jsmith (user), to run the *tar*, *ufsdump*, *cpio*, *mount* and *unmount* commands (operations), as *root*, on the host *accounting.company.com* (object). However, you want to be sure jsmith does not have access permissions to do anything else on the accounting system. The following command will grant the desired access:

```
accept from
"jsmith",, {"tar", "ufsdump", "cpio", "mount", "unmount"},
"accounting.company.com" with
(runuser="root");
```

Similarly, if you have a DBA (Database Administrator), kadams, and she should have permission to run *vi*, *svrmgr* and *sqlplus* as the Oracle user, the following command would grant that appropriate access:

```
Accept from
"kadams",, {"vi", "svrmgr", "sqlplus"},
"accounting.company.com" with
(runuser="oracle");
```

Notice that both commands assign the *runuser* to the appropriate account. If the *runuser* is not defined, the variable *runuser* will default to the

value of the variable *user* and the command will execute as the user who submitted it. PowerBroker operates on a default reject basis: no access is allowed unless specifically granted. This is consistent with the least privilege model found in the NIST RBAC standardized model. There is another important advantage here. Even though kadams has been granted the ability to run *vi* as a privileged user on the host *accounting.company.com*, you do not have to be concerned with unintended privilege escalation due to incorrectly set permissions, since the PowerBroker policy that is managing her access is located on a remote Master Host that she does not have access to. All her activity is also being logged onto a remote Log Host, creating an audit trail.

Now all this took two lines of code, which is very nice, but it is not really RBAC, is it?

There are "Objects" and "Operations" combining to give "Permission" to a "User," but where are the "Roles" and "Sessions?" This is what will come next...

Example 1B

PowerBroker policy code allows operations on lists of items, instead of individual items. Here we will define a list for each "Role" with the "Users" who should have that "Role." The statement as a whole creates our RBAC Session. We will also throw in a Hierarchy, so the "Sys Admin Role" will inherit the "Permissions" in the "Backup Admin Role," but not the "DBA Role:"

```
RoleSysAdm={"lkelley"}
RoleBackupAdmin={"jsmith"}+RoleSysAdm
in;
RoleDBAdmin={"kadams"};
```

```
accept from
RoleSysAdmin,, "accounting.company.com" with (runuser="root");
accept from
RoleBackupAdmin,, {"tar", "ufsdump", "cpio", "mount", "unmount"},
"accounting.company.com" with
(runuser="root");
accept from
RoleDBAdmin,, {"vi", "svrmgr", "sqlplus"},
"accounting.company.com" with
(runuser="oracle");
```

Now we have the equivalent of a Flat RBAC implementation, with a little dash of Hierarchical RBAC. Let us make this even better.

Example 1C

Our current example is for only one host, but one of the strengths of PowerBroker policy is that we can implement host-by-host restrictions while still maintaining a centralized policy.

```
RoleSysAdm = {"lkelley"}
RoleBackupAdmin = {"jsmith"} +
RoleSysAdmin; RoleDBAdmin =
{"kadams"}; HostsDB =
{"accounting.company.com",
"production.company.com"};
HostsBackup =
{"accounting.company.com",
"sales.company.com"}; HostsAll =
HostsDB + HostsBackup; accept from
RoleSysAdmin,,, HostsAll with
(runuser = "root"); accept from
RoleBackupAdmin,,{"tar", "ufsdump",
"cpio", "mount", "umount"},
HostsBackup with (runuser = "root");
accept from RoleDBAdmin,,{"vi",
"svrmgr", "sqlplus"}, HostsDB with
(runuser = "oracle");
```

We also want to enforce separation of duty as defined in Constrained RBAC. In this simple example, we want to ensure that a person does not hold "RoleSysAdmin" and "RoleDBAdmin" at the same time. We will add a check to our policy to terminate the request and inform the user when that issue is found. PowerBroker policy creates variables holding all the parameters of the request: the user and host submitting the request, the user and host where the request should be run, the privileged command, and so forth. We will simply check that the submitting user is not in both groups:

```
if (user in RoleSysAdmin && user in
RoleDBAdmin) { reject "Separation of
Duty Error: Membership in
RoleSysAdmin and RoleDBAdmin not
allowed at the same time"; }
```

Example 1D

Another PowerBroker strength is logging. PowerBroker logging can be managed by policy. By default, event logging is on and I/O logging is off. We want to turn on I/O logging for the "Sys Admins" and the "Backup Admin" but not for the "DBA's," since Oracle maintains its own logging. In PowerBroker, this is as simple as adding an "iolog" assignment. The ACL syntax for the accept command also supports running multiple expressions before privileged execution, so we can add the "iolog" assignment there:

```
accept from RoleSysAdmin,,,
HostsAdmin with (runuser = "root"),
(iolog = logmktemp("/var/log/pb." +
user + "." + basename(command) +
".XXXXXX"));
accept from RoleBackupAdmin,,{"tar",
"ufsdump", "cpio", "mount", "umount"},
HostsBackup with (runuser = "root"),
(iolog = logmktemp("/var/log/pb." +
user + "." + basename(command) +
".XXXXXX"));
```

Finally, we will extend our access control policy with a feature that none of the RBAC implementations support: time of day-based privileged access. Backup performed during business hours will cause problems. So let us restrict the "Backup Administrator" so he can only have access during non-business hours. The ACL syntax for the "accept" and "reject" statements allows a conditional expression to be evaluated as part of the command. The PowerBroker policy function "timebetween()" returns true or false based on the two time parameters given (in 24-hour time format). If the first time parameter is later than the second time parameter, the "timebetween()" function understands that the time should be evaluated across midnight.

```
accept from RoleBackupAdmin,,{"tar",
"ufsdump", "cpio", "mount", "umount"},
HostsBackup when
timebetween(1700,0800) with (runuser
= "root"), (iolog =
logmktemp("/var/log/pb." + user + "."
+ basename(command) + ".XXXXXX"));
```

Here is the finished policy. It provides functionality similar to Core RBAC; Hierarchical

RBAC; the static separation of duty found in Constrained RBAC; plus time-of-day privileged access management and selective logging functions that neither the RBAC specification nor the RBAC implementations provide for:

```
RoleSysAdm = {"lkelley"}
RoleBackupAdmin = {"jsmith"} +
RoleSysAdmin; RoleDBAdmin =
{"kadams"}; HostsDB =
{"accounting.company.com",
"production.company.com"};
HostsBackup =
{"accounting.company.com",
"sales.company.com"}; HostsAll =
HostsAdmin + HostsBackup; if (user in
RoleSysAdmin && user in RoleDBAdmin)
{ reject "Separation of Duty Error:
Membership in RoleSysAdmin and
RoleDBAdmin not allowed at the same
time"; }
accept from RoleSysAdmin,,,
HostsAdmin with (runuser = "root"),
(iolog = logmktemp("/var/log/pb." +
user + "." + basename(command) +
".XXXXXX"));
accept from RoleBackupAdmin, {"tar",
"ufsdump", "cpio", "mount", "umount"},
logmktemp("/var/log/pb." + user + "."
+ basename(command) + ".XXXXXX"));
```

```
accept from "kadams", {"vi", "svrmgr",
"sqlplus"}, accept from
RoleDBAdmin, {"vi", "svrmgr",
"sqlplus"}, HostsDB with (runuser =
"oracle"); HostsBackup when
timebetween(1700,0800) with (runuser
= "root"), (iolog =
```

Summary

Because of RBAC's obvious value in the IT security administration world, RBAC models, in some fashion or form, have been implemented by the Sun, HP, and Linux communities in various forms. Yet it has been shown that each of these implementations each have an assortment of limitations in meeting an organization's needs.

PowerBroker provides a way to implement role-based access control that provides the needed solution for the limitations found in operating-system RBAC implementations.

PowerBroker's approach enriches the security features of current RBAC implementations, while delivering important advantages in flexibility, separation of duties, audit trails, and strong, best-practices security.

Cited Sources

- ⁱ Kuhn, R., Ferraiolo, D., & Sandhu, R. (2007). The NIST model for role-based access control: Towards a unified standard. *IEEE Security & Privacy*, vol. 5, no. 6.
- ⁱⁱ Kuhn, R., Ferraiolo, D., & Sandhu, R. (2007). The NIST model for role-based access control: Towards a unified standard. *IEEE Security & Privacy*, vol. 5, no. 6.
- ⁱⁱⁱ Ninghui L., Ziad B., and Mahesh V. (2004). "On mutually exclusive roles and separation-of-duty,". *11th ACM conference on Computer and Communications Security*: 42 - 51.
- ^{iv} Kuhn, R., Ferraiolo, D., & Sandhu, R. (2007). The NIST model for role-based access control: Towards a unified standard. *IEEE Security & Privacy*, vol. 5, no. 6.
- ^v Ferraiolo, S., Sandhu, G. RBAC element definitions from "Proposed NIST Standard for Role-Based Access Control". <http://csrc.nist.gov/rbac/rbacSTD-ACM.pdf>.
- ^{vi} Kuhn, R., Ferraiolo, D., & Sandhu, R. (2007). The NIST model for role-based access control: Towards a unified standard. *IEEE Security & Privacy*, vol. 5, no. 6.
- ^{vii} RBAC in the Solaris operating environment. Sun Microsystems.
<http://www.sun.com/software/whitepapers/wp-rbac/wp-rbac.pdf>.
- ^{viii} Smalley, S. Configuring the SELinux policy – policy language and the example policy configuration.
<http://www.nsa.gov/selinux/papers/policy2-abs.cfm>.
- ^{ix} Nakamura. SELinux policy editor RBAC guide. http://seedit.sourceforge.net/doc/2.0/rbac_guide.pdf.
- ^x Moffat, D. (2009). London openSolaris user group security features overview – RBAC detail. Pg. 49.
<http://opensolaris.org/os/community/security/files/losug-security-rbac.pdf>.