Best Practices for Creating Simple, Effective Unix/Linux Least Privilege Policies

PowerBroker for Unix & Linux
# Table of Contents

Executive Summary .................................................................................................................. 3

Understanding Why Privileged Access Management (PAM) is Important ............................. 3

Regulatory Requirements and Privileged Access – a Primer .................................................... 3

General Policies for Unix/Linux Least Privilege ................................................................. 4

Writing System Administrator Policies ................................................................................. 4

  System administrator policy – Level 3 (highest level) ......................................................... 6
  System administrator policy – Level 2 (mid-level) .............................................................. 11
  System Administrator Policy – Level 1 (junior administrators) ......................................... 13

Conclusion ............................................................................................................................. 16

The PowerBroker Privileged Access Management Platform ................................................. 17

About BeyondTrust .................................................................................................................. 18
Executive Summary

In many IT organizations, there are administrators who need broad access to Unix/Linux systems to execute their work, but who require some restrictions from performing certain functions that may impact system availability, or over which oversight is desired (for example for compliance purposes). This tech brief is meant to be an introduction to policy writing, and best practices around developing policies within your environment so you can gain better control over your Unix/Linux environment while still ensuring your administrators are productive.

It is important to understand that these are only examples of how this can be accomplished, and is in no way a recommendation for policies within your environment. Every customer use case is different, and these posts merely demonstrate the flexibility and control available within PowerBroker for Unix & Linux policy language. Before we jump into policy, let’s quickly review why privileged access management (PAM) is important for Unix/Linux systems.

Understanding Why Privileged Access Management (PAM) is Important

Privileged access management is often driven by regulatory requirements that require businesses to know ‘who’ is doing ‘what’, ‘where’, ‘when’, and ‘why’ on systems they own, manage, or control. That is a very simple definition, but as you develop policy, those key words will help you to understand and reason through policies.

Understanding the drivers behind why PAM is so important in your organization is the first step toward writing effective policies. Although it is primarily driven by regulatory, audit, or general compliance requirements, it is important to understand that any or all of those can be behind specific reasons for policy development. Next, we’ll look at specific PAM requirements in common regulations.

Regulatory Requirements and Privileged Access – a Primer

Globally, there are many regulatory requirements that govern the delivery of PAM solutions. Ultimately, these regulations drive toward the same goal – protecting information for which an organization is responsible. Best practices dictate that companies, when faced with several regulations, should adhere to the highest standard that applies. In so doing, they will meet or exceed the lesser requirements. The most commonly recognized requirements are contained within one or more of these laws:

- **Sarbanes-Oxley (SOX)** – This is a US law that requires accurate reporting of financial information for all publicly held companies. Protecting financial databases from being altered (Privileged Access and Audit) can be considered under section 404 as an audit point.

- **Payment Card Industry / Data Security Standards (PCI/DSS)** – This is an over-arching requirement for all organizations that handle financial information, and describes processes and procedures
required to safely handle this data. Although this is not a law, it is treated as a mandate in most companies where financial data is handled.

- **Health Insurance Portability and Accountability Act (HIPPA)** – This is a US law that provides requirements related to the handling of health-care information. The most common interpretation of Title II requires companies to protect from disclosure all patient related information. In cases where database or system access is provided, audits must be available to prove compliance.

- **Health Information Trust Alliance (HITRUST)** – This is a privately held company that collaborates with healthcare, technology, and information security leaders to provide a prescriptive set of controls to harmonize regulations and standards. Similar to the PCI/DSS organization above, the directives from HITRUST do not carry the weight of law, but the underlying requirements may.

It is important to note that there are several overlapping requirements that can bind a company to multiple regulatory requirements. For instance, a small company that handles financial transactions and expects to abide by PCI/DSS requirements may also have a healthcare plan for their employees. How this healthcare related information is handled is a requirement of HITRUST/HIPAA regulations in addition to PCI/DSS, so elements of both may apply. Organizations should regularly consult with their legal, regulatory, and compliance teams to determine how and which regulations apply to their business.

**General Policies for Unix/Linux Least Privilege**

When writing PowerBroker policies, you have access to many resources on systems. Among these are centralized authentication (Active Directory, LDAP, or local group and identity information), DNS to positively identify both the source and destination servers, and other network resources that may be configured in an enterprise. It is a best practice to make as much use of these external sources as possible. This will help to avoid having to hard-code specifics into policy. It also adheres to the philosophy of ‘separation of duties’ where the policy writer is not responsible for delivering policy and provisioning identities or servers.

**Writing System Administrator Policies**

System administrators are responsible for making sure that systems are available and healthy; they monitor processes and ensure that applications are performing as designed, and they maintain systems to keep them patched and up to standards.

It is important that policies be developed based on the experience of system administrators that balances the risk of inappropriate activity against system maintenance. As system administrators become more experienced, they can be progressively granted more privileges on systems. For instance,
you may want to restrict a junior system administrator from being able to stop a network interface, or reboot a server without a higher-level administrator confirming that is the correct solution.

This method also opens the door to having oversight on system administrators where a second pair of eyes may be required to execute the most privileged commands.

When building the system administration policies, it is best to begin with a base policy that has the least restriction (Level 3), and use that as a template to add restrictions for the other levels. This may seem counter-intuitive, but having a solid template, and locking it down further for the other levels is the easiest way to proceed.

The code below is meant to be demonstrative, rather than efficient. The purpose is to demonstrate the logical construct of a policy, and to show examples for later use.

**Best Practice: Back up files**

When writing policy, or modifying existing policies, files should be backed up, and with each change, policy should be confirmed with the built-in ‘pbcheck’ command to test syntax and policy structure. All policies should be thoroughly tested prior to implementation.
SYSTEM ADMINISTRATOR POLICY – LEVEL 3 (HIGHEST LEVEL)

When writing the first policy we will assume that the most senior system administrators will have access to ‘any’ system, at any time, to perform any command. Session keystroke logging will be started, and a user notification will be displayed and acknowledged by the user.

Best Practice: Comment liberally

When writing policy, it is important that you use liberal comments. This serves several purposes, including:

1. In a busy environment, there are often several people involved in writing policies. Adding comments helps anyone else who may follow to understand the intent and logical construct of your policies.

2. From time to time, auditors, or a compliance team may request policies for review. Adding comments will help them to understand – even if they are not technical.

3. Once established, policies may go unaltered for years. Having comments will assist greatly when you look back and try to figure out what the design considerations were when it was developed.

Best Practice: Use headers, author, date

Policies should have a header that describes their purpose, the author of the policy, the date it was created, a record of any changes made with the date of the change, and who altered it.

Suggested Format:

```
# Level 3 System Administrator Policy
# Permits senior system administrators to execute any command
# on any server.
#
# Author: Bob Policywriter
# Original Implementation: 01/01/2017
#
# Changelog:
# 01/10/2017 – Fred PolicyWriter – Added something at the
#              request of the administrators (CHG0010000)
```
Best Practice: Improve change management

Proper change management, and keeping records of changes will greatly improve audit results, as well as training. Habitually keeping meticulous records of policy changes may be useful if policies ever come under scrutiny during a legal or compliance investigation as well. It is better to leave a detailed audit trail than to leave questions.

Policy will follow the ‘Who’, ‘Where’, ‘What’ pattern described above. In this first policy, this will be called out. In later policy examples, this will be assumed.

# Who
# In the development policy, system administrators are all
# members of the LDAP group ‘L3ADMINs’
# Define the group that this policy will apply to
L3AdminGroup = “L3ADMINs”;
#
# Where
# In the development policy, we’ll pretend that there are four
# *NIX servers in our environment.
# Define Server list where policy will apply
UnixServers = “{abc123, abc124, abc125, abc126}”;  

  • Note the curly braces around the list above. By creating a
    list of servers, we can refer to elements within the list when
    validating the request. See the Policy Language Guide for
    more information on lists and how to use them.

# What
# In the development policy, the senior administrators can run
# any command on any of the servers listed.
# Commands = “{*}”; -- All commands are authorized.

  • Since the exact command is not necessary in this policy (all
    commands are authorized), there is no need to define anything
    here. This will be a placeholder for later policies. In this
    case, a comment to say that all commands are authorized is all
    that is necessary.

# # Define the condition where this will apply using built-in
# variables that come when a request is processed.
if ( (L3AdminGroup in groups) && (runhost in UnixServers) ) {

} # End if group and host are in the list
Best Practice: Comments on conditional statements

Policies can become very large and complex – putting comments on the closing braces on ‘if’, ‘for’, and other conditional statements will make it much easier to keep policies organized.

The following code will go within the ‘if’ statement above. It is broken out here so each line can be explained in detail. Bear in mind that within policy you can control all aspects of a user’s environment when authorizing commands.

Step 1 will be to set up the session log for this task:

```plaintext
# Set up I/O (keystroke) log for this session
# Identify the root directory where iologs will be stored
iolog_dir="/iologs";
# Set the HHMM for log file naming
logtime=strftime("%H:%M");
# Set the log file name
iolog=iolog_dir + "pbul_iolog." # All logs will start with ‘pbul_iolog’
   + "admin-session." # Description for session
   + strftime("%d-%d-%d", month, day, year) + "." # MM-DD-YYYY
   + logtime + "." # HH:MM
   + split(runhost,".")[0]; # Short name of the host
   + basename(command) + "."; # Command entered
   + ".XXXXX"; # Unique serial number
```

The filename will expand out to:

```
/iologs/pbul_iolog.admin-session.01-01-2017.12:00.abc123.su.999999
```

• Note that how the session logs are named is entirely up to the customer. The example above just demonstrates how variables can be used to construct a logical file name.

Step 2 will be to set some basic user environment variables:

```plaintext
# Set user environment for this session
runcwd="/tmp"; # Directory to run command from
rungroup="!g!"; # Group for user
rungroups={"!G!"}; # All groups for user
setenv("SHELL", "!!!"); # Set user shell
setenv("HOME", "!~!"); # Set user home
setenv("USER", RunUserName); # Set username
setenv("USERNAME", RunUserName); # Set Name i.e. ‘bob jones’
setenv("LOGNAME", RunUserName); # Set logname variable
setenv("PWD", runcwd); # Set directory to start in
setenv("PATH", "/bin:/usr/bin:/sbin:/usr/sbin"); Set PATH
```
• Most of the above variables are not necessary for most policies. Setting those variables was a way to demonstrate how the user environment can be controlled within a session. For a complete list of variables available, consult the Policy Language Guide.

Step 3 is to set command execution parameters:

```plaintext
# Set command execution
runuser = "root"; # All commands for this policy run as ‘root’
# Notify user that session logging is on
print("WARNING: This session is being centrally recorded.");
print("Session Recording Filename:", iolog);
```

• Note – In some jurisdictions, displaying a notice to privileged users is sufficient, in some it is not required, and in others, it is necessary that they acknowledge the warning. It is up to the customer to determine what is required for their business, or industry, and to develop policies accordingly.

```plaintext
# Accept the command, and proceed with execution
accept;
```

• When an ‘accept’ is received within policy, an event is logged, and in the case of this policy, a session log is started.

The entire policy for the senior system administrators:

```plaintext
# Level 3 System Administrator Policy
# Permits senior system administrators to execute any command
# on any server.
#
# Author: Bob Policywriter
# Original Implementation: 01/01/2017
#
# Changelog:
# 01/10/2017 – Fred PolicyWriter – Added something at the
#              request of the administrators (CHG0010000)
#
# Who
# In the development policy, system administrators are all
# members of the LDAP group ‘L3ADMIN’S
# Define the group that this policy will apply to
L3AdminGroup = "L3ADMIN’S";
#
# Where
# In the development policy, we’ll pretend that there are four
```
# *NIX servers in our environment.
# Define Server list where policy will apply
UnixServers = "\{abc123, abc124, abc125, abc126\}";
# What
# In the development policy, the senior administrators can run
# any command on any of the servers listed.
# Commands = "\{*\}"; -- All commands are authorized.
#
# Define the condition where this will apply using built-in
# variables that come when a request is processed.
if ( (L3AdminGroup in groups) && (runhost in UnixServers) ) {
# Set up I/O (keystroke) log for this session
# Identify the root directory where iologs will be stored
iolog_dir="/iologs"
# Set the HHMM for log file naming
logtime=strftime("%H:%M")
# Set the log file name
iolog=iolog_dir
    + "pbul_iolog." # All logs will start with ‘pbul_iolog’
    + "admin-session." # Description for session
    + sprintf("%d-%d-%d",month,day,year) + "." # MM-DD-YYYY
    + logtime + "." # HH:MM
    + split(runhost,".")[0]; # Short name of the host
    + basename(command) + "."; # Command entered
    + ".XXXXXX"; # Unique serial number
# Set user environment for this session
runcwd = "/tmp"; # Directory to run command from
rungroup = "!g!"; # Group for user
rungroups = {"!G!"}; # All groups for user
setenv("SHELL", "!!"); # Set user shell
setenv("HOME", "!!!"); # Set user home
setenv("USER", RunUserName); # Set username
setenv("USERNAME", RunUserName); # Set Name i.e. ‘bob jones’
setenv("LOGNAME", RunUserName); # Set logname variable
setenv("PWD", runcwd); # Set directory to start in
setenv("PATH", ":/bin:/usr/bin:/sbin:/usr/sbin"); Set PATH
# Set command execution
runuser = "root"; # All commands for this policy run as ‘root’
# Notify user that session logging is on
print("WARNING: This session is being centrally recorded.");
print("Session Recording Filenname:",iolog);
# Accept the command, and proceed with execution accept;
} # End if group and host are in the list
SYSTEM ADMINISTRATOR POLICY – LEVEL 2 (MID-LEVEL)

We will use the Senior System Administrator policy from the first in this series of tech tips as a template to build the Level 2 system administrator policy. The structure will be the same, and some additional command restrictions will be added to tighten user activities using some other features available within the policy language.

In this example, we will restrict the Level 2 administrators from performing a ‘shutdown’ command on a server, or disabling a network interface. Using Advanced Control and Audit (ACA) within PBUL language, access to files, file systems, or execution of binaries can be restricted.

The Senior System Administrator policy from the first post in this series will be used as a template, and new lines will be inserted to add restrictions. Those lines will be in red within the policy for visibility.

# Level 2 System Administrator Policy
# Permits mid-level system administrators to execute nearly any command on any server. These administrators are not permitted to shutdown, or reboot a server, nor are they allowed to stop or reconfigure network interfaces.
#
# Author: Bob Policywriter
# Original Implementation: 01/01/2017
#
# Changelog:
# 01/10/2017 − Fred PolicyWriter − Added something at the request of the administrators (CHG0010000)
#
# Who
# In the development policy, mid-level system administrators are all members of the LDAP group ‘L2ADMINS’
# Define the group that this policy will apply to
L2AdminGroup = “L2ADMINS”;
#
# Where
# In the development policy, we’ll pretend that there are four *NIX servers in our environment.
# Define Server list where policy will apply
UnixServers = “{abc123, abc124, abc125, abc126}”;
#
# What
# In the development policy, the senior administrators can run any command on any of the servers listed.
# Commands = “{*}”; -- All commands are authorized.
# Define the condition where this will apply using built-in variables that come when a request is processed.
if ( (L2AdminGroup in groups) && (runhost in UnixServers) ) {
    # Prevent direct execution of specific commands
    if ( basename(command) in { "shutdown", "halt", "reboot", "ifconfig" } ) {
        # do not allow these commands to be delegated
        reject("This is a restricted command -- '");
    }
}

# Set up I/O (keystroke) log for this session
# Identify the root directory where iologs will be stored
iolog_dir="/iologs";
# Set the HHMM for log file naming
logtime=strftime("%H:%M");
# Set the log file name
iolog=iolog_dir + "pbul_iolog." # All logs will start with 'pbul_iolog'
    + "admin-session." # Description for session
    + sprintf("%d-%d-%d",month,day,year) + "." # MM-DD-YYYY
    + logtime + "." # HH:MM
    + split(runhost,".")[0]; # Short name of the host
    + basename(command) + "."; # Command entered
    + ".XXXXXX"; # Unique serial number

# Set user environment for this session
runcwd="/tmp"; # Directory to run command from
rungroup="!g!"; # Group for user
rungroups={"!G!"}; # All groups for user
setenv("SHELL", "!!"); # Set user shell
setenv("HOME", ".!~!"); # Set user home
setenv("USER", RunUserName); # Set username
setenv("USERNAME", RunUserName); # Set Name i.e. 'bob jones'
setenv("LOGNAME", RunUserName); # Set logname variable
setenv("PWD", runcwd); # Set directory to start in
setenv("PATH", "/bin:/usr/bin:/sbin:/usr/sbin"); # Set PATH

# Set command execution
runuser="root"; # All commands for this policy run as 'root'
# Restrict command execution within a session.
aca("file", "/bin/shutdown", "!exec");
aca("file", "/bin/halt", "!exec");
aca("file", "/bin/ifconfig", "!exec");
aca("file", "/bin/reboot", "!exec");
# Notify user that session logging is on
print("WARNING: This session is being centrally recorded.");
print("Session Recording Filename:",iolog);
# Accept the command, and proceed with execution
accept;

} # End if group and host are in the list

• Note that there are two sets of ACA rules in the policy. The first is in case the command was
called directly (‘pbrun shutdown’), and the second will work within a session if someone
directly accessed ‘root’.

SYSTEM ADMINISTRATOR POLICY – LEVEL 1 (JUNIOR ADMINISTRATORS)

We will use the first two policies from this series as a template to build the Level 1 system administrator
policy. The structure will be the same, and some additional command restrictions will be added to
tighten user activities using some other features available within the policy language. In addition to the
restrictions placed on the mid-level administrators which prevented shutting down servers, or stopping
network interfaces, we will add a restriction that prevents modification of any system configuration in
the /etc directory.

The mid-level system administrator policy from the second article in this series will be used as a
template, and new lines will be inserted to add restrictions. Those lines will be in red within the policy
for visibility.

# Level 1 System Administrator Policy
# Permits junior system administrators to execute nearly any
# command on any server. These administrators are not permitted
# to shutdown, or reboot a server, nor are they allowed to stop
# or reconfigure network interfaces or modify host
# configuration.
#
# Author: Bob Policywriter
# Original Implementation: 01/01/2017
#
# Changelog:
# 01/10/2017 – Fred PolicyWriter – Added something at the
# request of the administrators (CHG0010000)
#
# Who
# In the development policy, junior system administrators are
# all members of the LDAP group ‘L1ADMINS’
# Define the group that this policy will apply to
L1AdminGroup = “L1ADMINS”;
# Where
# In the development policy, we’ll pretend that there are four
# *NIX servers in our environment.
# Define Server list where policy will apply
UnixServers = "{abc123, abc124, abc125, abc126}";

# What
# In the development policy, the senior administrators can run
# any command on any of the servers listed.
# Commands = "{*}"; -- All commands are authorized.
#
# Define the condition where this will apply using built-in
# variables that come when a request is processed.
if ( (L1AdminGroup in groups) && (runhost in UnixServers) ) {

# Prevent direct execution of specific commands
# Capture the entire command the user entered
RequestCommand = sprintf("%s", runargv);
if ( ( basename(command) in { "shutdown", "halt", "reboot", "ifconfig" } ) || ( RequestCommand in { "vi /etc/**" } ) ) {
    # do not allow these commands to be delegated
    reject("This is a restricted command -- " + basename(command) + ":");
}

# Set up I/O (keystroke) log for this session
# Identify the root directory where iologs will be stored
iolog_dir="/iologs";
# Set the HHMM for log file naming
logtime = strftime("%H:%M");
# Set the log file name
iolog = iolog_dir
    + "pbul_iolog." # All logs will start with 'pbul_iolog'
    + "admin-session." # Description for session
    + sprintf("%d-%d-%d", month, day, year) + "." # MM-DD-YYYY
    + logtime + "." # HH:MM
    + split(runhost,".")[0]; # Short name of the host
    + basename(command) + ":"; # Command entered
    + ".XXXXXX"; # Unique serial number

# Set user environment for this session
runcwd = "/tmp"; # Directory to run command from
rungroup = "!g!"; # Group for user
rungroups = {"!G!"}; # All groups for user
setenv("SHELL", "!!!"); # Set user shell
setenv("HOME", "!~!"); # Set user home
setenv("USER", RunUserName); # Set username
setenv("USERNAME", RunUserName); # Set Name i.e. ‘bob jones’
setenv("LOGNAME", RunUserName); # Set logname variable
setenv("PWD", runcwd); # Set directory to start in
setenv("PATH", "/bin:/usr/bin:/sbin:/usr/sbin"); Set PATH
# Set command execution
    runuser = “root”; # All commands for this policy run as ‘root’
    # Restrict command execution within a session.
    aca("file", "/bin/shutdown", "!exec");
    aca("file", "/bin/halt", "!exec");
    aca("file", "/bin/ifconfig", "!exec");
    aca("file", "/bin/reboot", "!exec");
    aca("file", "/etc", "!write");

    # Notify user that session logging is on
    print("WARNING: This session is being centrally recorded.");
    print("Session Recording Filename:", iolog);
    # Accept the command, and proceed with execution
    accept;

) # End if group and host are in the list
Conclusion

As you can tell, the PowerBroker for Unix & Linux policy language is very powerful, and permits control over virtually all aspects of a user’s activity within a session. Building an initial policy and using it as a template for other policies is an easy way to expand policy control across an environment. The Advanced Control and Audit feature is a very powerful tool within the policy language, and permits granular control over user activities even within a privileged session.

Restricting activities for various levels of system administrators can be considered a way to enforce internal escalation within an operational environment. If the need to stop a system, or a network interface arises, making this unavailable to lower level administrators will ensure that senior administrators are engaged before availability is impacted. In some cases, only two levels will suffice, but any number of levels can be designed using policy, each with separate or compound restrictions.

It is strongly recommended that the above examples be used as a starting point for developing policies customized to suit your environment. For more information on how PowerBroker for Unix & Linux can help you simplify Unix/Linux security and compliance, download our whitepaper or contact us for a demo today.
The PowerBroker Privileged Access Management Platform

PowerBroker for Unix & Linux, is part of BeyondTrust’s solutions for Server Privilege Management and integrates with other privileged access management solutions in the PowerBroker Privileged Access Management Platform. The platform is an integrated solution to provide control and visibility over all privileged accounts and users. By uniting capabilities that many alternative providers offer as disjointed tools, the PowerBroker platform simplifies deployments, reduces costs, improves system security and closes gaps to reduce privileged risks.
About BeyondTrust

BeyondTrust® is a global security company that believes preventing data breaches requires the right visibility to enable control over internal and external risks.

We give you the visibility to confidently reduce risks and the control to take proactive, informed action against data breach threats. And because threats can come from anywhere, we built a platform that unifies the most effective technologies for addressing both internal and external risk: Privileged Account Management and Vulnerability Management. Our solutions grow with your needs, making sure you maintain control no matter where your organization goes.

BeyondTrust's security solutions are trusted by over 4,000 customers worldwide, including over half of the Fortune 100. To learn more about BeyondTrust, please visit www.beyondtrust.com.