WHITE PAPER

beyondtrust®

Beyond Traditional Security

Successfully Securing the Open Source Enterprise

Privileged User Management in Linux Environments
by *Linux Mastermind* – Sander Van Vugt

Table of Contents

Executive Summary

UNIX BACKGROUND

To understand how the current solutions for the delegation of system administration are structured, it is important to understand the way that UNIX was originally designed and specifically the methods that were offered in this original design to manage the delegation of admin tasks. Originally, there were just two different kinds of UNIX processes: privileged processes and unprivileged processes. An unprivileged process was limited in its way of performing system functions, where privileged processes were designed to bypass all restrictions. To execute privileged processes, the root user account was used: a user account with no limitations whatsoever. From the beginning, the concept of the root account was designed to perform system administration tasts without limitation whatsoever.

SUDO

To help normal (unprivileged) users run administration tasks, several solutions were created. One of them was sudo. The purpose of sudo (short for superuser so) was simple: connect a user or a group of users to a privileged process of commands. Sudo allows normal users to run root tasks. Stated otherwise, sudo allows normal users to start a task that creates an environment in which these normal users are roo, and if for any reason there is a backdoor or bug in such a task, there is a risk that through that task, the user gets complete root access. However, before further zooming into sudo particularities, let us have a look at the supported mechanisms that are offered by the UNIX and Linux operating systems.

CAPABILITIES

Since about the year 2000, when the first 2.2 Linux kernels were released, the feature of "capabilities" was introduced in the kernel with the purpose of offering more choice than just that between privileged and unprivileged processes. A capability determines rights to specific tasks, such as changing of ownership (CAP_CHOWN), changing of permissions (CAP_CHMOD), or the capability to terminate running tasks on a server (CAP_KILL). In total, there are about 32 capabilities, and the idea of using these was to define in a more granular way what exactly a process or thread is allowed to do.

And this works fine for system processes that are started with an unprivileged user account. However, it does not work that fine for a system task that was started by the root user, because if the real user ID that was used for executing a task is zero, all capabilities are enabled. Therefore, capabilities are a partially useful attempt to limit the way administration tasks are performed, but for a modern global enterprise, they are not enough.

Capabilities are a large step forward to a more secured environment, but their effectiveness is related to the user account that runs a task. That is why in current Linux it is not that easy to use capabilities for the delegation of system administration tasks. An example of a service that does use capabilities is AppArmor of SELinux. These security mechanisms, both based on the security framework in the Linux kernel, allow administrators to set-up profiles that limit the things that programs can do in the operating system. In that profile, the application is allowed to use certain capabilities and access certain files, while at the same time it blocks access to those that are not in the application profile. Although seen as an important step forward in offering better security for Linux, these solutions do not handle the particular needs that are required for the efficient delegation of system administration tasks.

Performing Administrative Tasks

In modern Linux environment, three imporant solutions can be used to delegate system adminstration tasks to ordinary users:

- su
- sudo

- policykit

The most unsecure method that can be used to perform system administration tasks is the su command, where a user just switched identities and becomes root. The idea behind su comes from the traditional UNIX environment, where every user that is allowed to work with a computer is a trusted user that only needs to be protected against him/her. In a su environment, a user normally works with his own unprivileged user account, and when the user needs to perform unprivileged tasks, the user becomes root by executing the su command.

From that moment on, there is no limitation whatsoever to what the user can do, and nothing the user is doing is written to a log file (with the exception of the history file in the home directory of user root). In the environment where su was first created, there was nothing wrong with this approach, only limited people had access to the computer anyway, and those who had access were certainly supposed to be capable of understanding the great responsibilities that come with becoming root.

DISADVANTAGES TO USING THE SU APPROACH

- There is no way to specify which tasks exactly the user can execute: either su offers an all-or-nothing approach where the user is root, or the user is not.

- Once in the su environment, the user has full access to everything, including all log files, so if something goes wrong, it is very easy to erase the traces of what exactly went wrong.

THese two disadvantages alone make su unfit as a solution for the delegation of administrative tasks.

POLICYKIT

Policykit represents a recent attempt to offer a solution where privileged tasks can be executed by unprivileged users. There is one major disadvantage though that disqualifies Policykit as an enterprise solution for the delegation of administrative tasks: it was designed for graphical environments. That means that Policykit works great for the delegation of tasks on a graphical Linux desktop, but it was never meant to be a solution for the delegation of administrative tasks on Linux servers.

sudo: the Default Linux Solution for Admin Delegation

The best solution currently available for the delegation of root tasks is sudo.This solution, which was originally designed in the 1990's, is simple and effective: there is a sudoers configuration file in which specific users are assigned permissions to execute specific tasks as a specific user account. sudo even offers options to store the configuration at an LDAP-server, which makes it possible to use LDAP as the centralized Directory where LDAP users are connected to the tasks that are stored in LDAP as well. Also, sudo allows administrators to define lists of which users from which hosts are allowed to run specific commands as a specific user. That means that sudo not only goes beyond the mere indication of a user that is allowed to run a command as root, but it can be used to designate a specific command or group of commands to be executed as a specific user.

Contrary to common belief, sudo is more than just having people executing tasks as root. The administrator setting up the sudo environment can specify which tasks should be run under specific users using the *Runas_list*. And furthermore, sudo can define from which specific host a task can be executed. However, while all of these features are nice to have, they do not make it a totally secure solution.

The main problem with sudo is that for administrative tasks, the user needs administrative privileges. From this follows that the most important disadvantage inherent to sudo still stands: once the privileged user has started the privileged task to which sudo gives him full access, in the environment of that specific task, the unprivileged user has indeed become root. This risk has been noticed by the sudo developers, and options have been added to mitigate this specific

risk, but even if you use options like no_exec, in certain cases these options just do not provide enough protection (read the main page of the sudoers configuration file for more details).

Note: another approach to securing sudo is to combine the sudo configuration with SELinux or AppArmor policies. SELinux is available as a default mechanism on Red Hat Enterprise Linux, the most used Linux distribution in the enterprise. Creating custom configurations for SELinux however is extremely difficult, and for that reason alone, most people tend to switch it off. Therefore, to create a secure environment where administrative tasks can be delegated, the combination between sudo and SELinux does not seem usable at all.

There are still more problems that make sudo a questionable candidate for enterprise datacenters. One of these is the storing of sudo in an LDAP directory. The idea of storing the sudo rules in LDAP is great, as it allows users to take a centralized approach for delegating system administration. There is one problem though: if the user succeeds in creating a local sudo configuration that can be used first, it will completely overrule the options that are defined in LDAP. It is just too easy to overrule the centralized sudo rules that come from LDAP!

Separating Configuration from the Task Execution Environment

Assuming that a user with limited access to administrative tasks wants more, there is the major problem that the sudo configuration in sudo environments is normally stored on the same server where the privileged tasks are to be executed. That means that if a breach in security does occur, the user not only can read the complete configuration, but the user also has the option to change it and give himself more privileges.

In a secure environment, it is common to separate the security policy from the environment where the security related tasks are to be executed. In an ideal world, the user requesting access to a secured task does that by sending his request to an agent, and then this agent talks to a process on a different server, which matches the request to the policy and authorizes it. Using this multi-server approach adds another layer of security: a possible intruder does not just have to crack the security policy, but the agent-server structure as well and that can be made virtually impossible. By separating the configuration from the environment where the administrative tasks are executed, the sudo solution would become much stronger. Unfortunately, there is nothing currently indicating that sudo is being developed in that direction.

Logging

sudo makes it possible to log events: an administrator can configure sudo to log every command the user executes. If, however, the command opens an interactive environment, there is no way to log what happens in that environment. That means that if a serious security breach should occur, you may find yourself unable to trace down in the log files exactly what has happened. To be completely compliant and meet regulations, an option that allows logging of every single key that a user is typing while in the administration delegating environment is required, otherwise you will just not be able to track down what has happened.

To enable secure logging, it is mandatory to write logs to a different server. This brings the benefit that it is harder to compromise the log files in case of a security breach. The options that sudo offers to accomplish this are clearly open for improvement. The base log destination that sudo is using is syslog, or one of its modern alternatives (rsyslogd and syslog-ng). The reliability of sudo logging all depends on the underlying log mechanism that is used, and the currently available Linux log solutions have some issues too.

LINUX LOG ISSUES

First, if the traditional syslog service is used, logging happens over UDP, which means that not even the delivery of log messages at the log server is guaranteed. Fortunately, the modern syslog alternatives syslog-ng and rsyslogd offer options to use TCP as the underlying protocol, which at least ensures that the log messages arrive at the destination server. None of the current log mechanisms, however, has extended options for encryption.

That means that by default the log messages are sent unencrypted over the network, and they are stored unencrypted on disk, resulting in messages that can be intercepted. Additionally, once the log server has been compromised, the log files can just be copied off that server, allowing the intruder to completely analyze how your sudo environment has been built. This may be acceptable if you run a school, but it definitely is not if you are responsible for servers in trusted environments like banks.

Also, all of the current log solutions are open to an extremely simple hack, which is based on the fact that any user can write events to syslog. If for instance a user has shell access, executing a simple bash shell code would flood the log files with information in no time, which in almost all cases causes the syslog mechanism in use to drop all messages that are logged, even in an environment where a remote server is used for logging of events.

If a normal user runs a simple script for a few minutes, the user will be able to bring down the log services completely, a perfect way to wipe any traces that anyone has made while performing unpleasant actions on your servers. It may be clear that logging in current sudo environments is open to some serious limitations.

DETAILED POLICY DEFINITION

The last important disadvantage of using sudo is in the way that policies are written. The sudo configuration basically is a group of commands with related arguments that a user is allowed to run. Apart from that, if so allowed, the user can open a sudo shell by executing the sudo -i command, which gives the user full access to all root tasks, meaning that no policy is effective at all. What sudo needs is an intelligent environment where flexible policies can be used that are based on regular expressions.

Just using intelligent policies is not enough – in a truly secured environment, you will need the environment where the delegated administrator is working to be constantly monitored and take action if an untrusted command is executed. The ideal environment would define tasks that are allowed and also define tasks that should never be allowed when working in the secured environment, even with the option to shut down a session at the moment that a disallowed task is executed.

SCALABILITY

sudo was designed in 1994 for an operating system that was created in the late 1960's. sudo was never intended to be used in environments where hundreds of Linux and UNIX servers all need to comply with corporate regulation. As previously illustrated, sudo lacks the fundamental features to be used in large environments. The best that sudo can offer is the storage of its configuration in Active Directory or LDAP, but even if the configuration is stored in a Directory server, the architecture of sudo is still focused on the server where the administrator is doing its work.

An ideal environment for the delegation of administrative tasks allows for the integration of the configuration, as well as the users in common Directory servers, such as Microsoft's Active Directory or LDAP. To make it secure, it should not just copy the configuration over from the Directory server to the local host (where it would be open for local attacks), but it should work in an architecture where an agent communicates to a server to get authorization before a task can be executed. Only if the authorization is provided by an external entity, the delegated administration environment can be really secure.

Summarizing sudo Disadvantages

Although perfectly fit for small environments, sudo does have some serious drawbacks that make it an unfit candidate for the delegation of administrative tasks in larger environments.

- sudo uses an all-or-nothing approach and from within the sudo environment, the user effectively has become root. There are options to mitigate that risk, but they are far from perfect.

- sudo doesn't use a distributed environment, which makes it unfit for use in large corporate environments
- sudo relies on syslog, which is a fairly weak logging solution
- sudo does not offer options to define in a very detailed way what a user should, or should not, be allowed to do
- sudo was never designed to be scalable

Enterprise Ready Solutions

From the above follows, that there are no current open source solutions that offer all that is required for the secure delegation of administrative tasks. There are a few commercial solutions available though. An often-used solution that meets all requirements for the secure delegation of admin tasks is Beyond Trust's PowerBroker® Servers. In this solution, detailed policies can be created using the embedded policy language.

Also, a delegated administrative user can do his work from a secure pbshell environment, where everything the user enters is key-logged and actions can be defined to execute at the moment the user enters an untrusted command, such as disconnecting the user from a server.

PowerBroker Servers is a scalable solution, where policies are stored on another server that the one where the administrative user is doing his work. After requesting administrative access, the PowerBroker Agent requests access to the policy server and once this server has allowed access, the user can do his work. While doing this, everything the user is doing is logged using the specific logging mechanism that PowerBroker offers. By offering these features, PowerBroker has earned its merits, which is proved by the fact that many banks and other financial institutions are using it to set-up a secure environment for managing the delegation of administration tasks.

About PowerBroker Servers

ROOT PASSWORD REMAINS CONFIDENTIAL AND SECURE

PowerBroker Servers empowers IT organizations with the ability to delegate root tasks and authorization on Linux, UNIX, and Mac OS X platforms without every disclosing the highly sensitive root password. PowerBroker Servers utilizes highly flexible policy language to enable enterprises to dictate permissions for users down to the most granular level, extending to any command executable on a UNIX, Linux, or Mac OS X server. This capability along with PowerBroker Servers' audit-ready logging and comprehensive reporting combine to deliver a solution that enables businesses of every size and industry to satisfy critical internal and external compliance requirements.

beyondtrust®
Beyond Traditional Security

About BeyondTrust

With more than 25 years of global success, BeyondTrust is the pioneer of Privileged Identity Management (PIM) and vulnerability management solutions for dynamic IT environments. More than half of the companies listed on the Dow Jones Industrial Average rely on BeyondTrust to secure their enterprises. Customers include eight of the world's 10 largest banks, seven of the world's 10 largest aerospace and defense firms, and six of the 10 largest U.S. pharmaceutical companies, as well as renowned universities. The company is privately held, and headquartered in Carlsbad, California. For more information, visit **beyondtrust.com**.

CONTACT INFO

NORTH AMERICAN SALES
1.800.234.9072
sales@beyondtrust.com

EMEA SALES
Tel: + 44 (0) 8704 586224
emeainfo@beyondtrust.com

CORPORATE HEADQUARTERS
550 West C Street, Suite 1650
San Diego, CA 92101
1.800.234.9072

CONNECT WITH US
Twitter: @beyondtrust
Facebook.com/beyondtrust
Linkedin.com/company/beyondtrust
www.beyondtrust.com