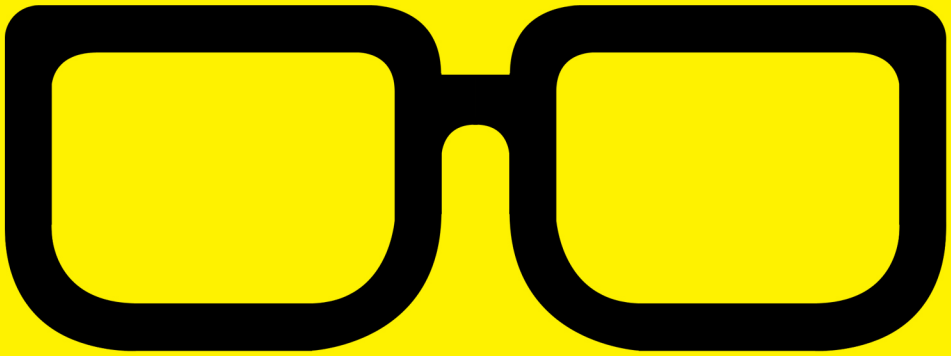
SPONSORED BY

BeyondTrust™

GEEK GUIDE



Beyond Sudo:

How to Know You Have Outgrown It (and What to Do)

LINUX™ JOURNAL

Table of Contents

GREG BLEDSOE is a Managing Consultant with Accenture in the DevOps Architecture Practice. He has more than 20 years of hard-fought experience in security and operations, having been a developer, network engineer, sysadmin, techops manager, Vice President of Operations and CISO. You can reach him at lj@bledsoehome.net or via Twitter: @geek_king.

GEEK GUIDES:
Mission-critical information for the most technical people on the planet.

About the Sponsor

BeyondTrust

BeyondTrust is a global information security software company that helps organizations prevent cyber attacks and unauthorized data access due to privilege abuse. Our solutions give you the visibility to confidently reduce risks and the control to take proactive, informed action against data breach threats. BeyondTrust's privileged access management solutions are trusted by more than 4,000 customers worldwide, including half of the Fortune 100. To learn more about BeyondTrust, please visit www.beyondtrust.com.

Beyond Sudo: How to Know You Have Outgrown It (and What to Do)

GREG BLEDSOE

Introduction

"If you build it they will come." Are freeways built to travel between existing communities, or do communities spring up around freeways? Is this a chicken-and-egg problem, or is there a complex interaction where such things shape each other?

The use of UNIX and Linux security tools raises similar questions. Do people work the way they do because of the tools they have, or do people have the tools they have

In many ways, the advance of sudo is the story of the advance of digital technology, and the story of sudo in the age of DevOps and the scale enabled by automation is this story in a microcosm.

because of the way they work? New types of tools are built when someone has an insight into how to improve the way work is currently done—and the tool then shapes the way work is done going forward. Tools are built to the work at hand from the perspective of someone who feels a gap in capability, and this is very important in deciding when a tool is "fit for purpose".

There is no question that a new tool can revolutionize the way work happens, and this is a big part of the story of the world's technological progress. Better tools can mean sizable competitive advantage, but that seldom lasts long. What works is copied, and it comes back to the application of the tool and the process around the how and why of its usage—that's what makes for lasting advantage. Value determines longevity.

The need to enforce privilege layers in our digital systems is a persistent one. In many ways, the advance of sudo is the story of the advance of digital technology, and the story of sudo in the age of DevOps and the scale enabled by automation is this story in a microcosm. Our needs have

shifted from bespoke configuration on a system-by-system basis to the need to control, configure and audit swiftly and repeatedly many ephemeral yet highly regulated systems at once. Manual solutions are no longer an option. But what makes for the right use cases for sudo? And, how do you know if you've outgrown its use? That's precisely what this Geek Guide discusses.

A Bit of Sudo History

Once upon a time, there were only large, expensive computers shared by many users with no fine-grained permission controls that ran only one function at a time. All commands ran with all rights to all the hardware, and all the processes and memory upon it. Simple mistakes could bring expensive machines like these to a halt for extended periods or could compromise the careful work of other users of the system.

This gradually began to change as the "operating system" started taking over essential functions and controlling other processes. Concepts like privilege separation began to evolve, and user accounts became the norm. Still, if you needed to execute one single command outside your level of privilege, you would escalate to the privileged user, and all those risks would come right back. This had several consequences, including the fact that if you changed the administrative account password, it had to be shared with everyone who might need it, not to mention the fact that accounting and auditing became nearly impossible.

This, in broad strokes, was the world of UNIX before 1980 when a quantum leap in fine-grained escalation control was

conceived by some gentlemen—namely Robert Coggeshall and Cliff Spencer—working at the Department of Computer Science at SUNY/Buffalo. It ran on a VAX-11/750 running 4.1BSD, and they named their software sudo, short for "superuser do". Originally its only function was to allow temporary privilege escalation to the superuser (or root) account, except using the user's own password instead of sharing the root account. The first update to this software was not released for five years, and it has iterated in faster and faster loops since, as software has tended to do as versions multiply and functions accrue.

It wasn't until 1994 that support for more UNIX platforms was added, and in 2003, LDAP integration became official. In 2005, a new parser was released, and advances in capability began to come swiftly. An ecosystem of supporting tools like "visudo" and various GUIs and GUI tools emerged, as well as competitors like "runas" on Microsoft platforms and "doas" in BSD. These newer tools have their user base, but none have grown as large as sudo, and none have developed the simplicity in command language that sudo brings. If you have used any modern *nix in the past 20 years and needed elevated privileges, you almost certainly have at least a passing familiarity with it.

Sudo Usage

The basic usage of sudo is both familiar and obscure. Hidden in the obscurity is complexity that allows capability that far exceeds common usage. Most casual users will never *really* understand that complexity because the only

line they have ever seen, used or modified is:

```
# Members of the admin group may gain root privileges
%admin ALL=(ALL) ALL
```

What they know is that the file /etc/sudoers, which they generally know to edit through visudo (so as not to foul themselves by opening the file more than once or saving it in unreadable form and thus potentially locking themselves out of the superuser account) contains parameters that define who can run what as whom and under what conditions.

In administering your own local desktop, using sudo for individual commands or using the following may be all you ever need to do:

```
sudo su - root
```

If you need to use UNIX or Linux professionally, have more than one user and the need to meet local, regional or global audit and security requirements, things can become complex quickly, because you will have to use the detailed features of sudo to define policy for multiple users, for multiple reasons, on multiple systems.

One of the keys to understanding why all this complicated difficult-to-manage complexity exists is to understand how it came to be. Sudo evolved through solving one problem at a time, and sometimes implications and effects on the amazingly functional and flexible systems we were building into UNIX weren't fully understood until after the problem already had been exploited. Pipes, for example, allow for those

amazing one-line strings of commands that make UNIX experts wizards, but if you start thinking through the various security implications of piping to and from various system devices, you'll soon go mad. If we were rebuilding a tool from scratch today, we would have all this accumulated knowledge in mind before we began and potentially simplify things considerably.

As it is, there are many subtle differences in slight divergences of sudo usage, like the difference between `sudo su`, `sudo su -`, `sudo su - root`, `sudo -i` and `sudo -s`, all of which can turn up slight but maddening variations in the resulting shells in which you find yourself. Combining these variations with different sudo defaults in the configuration file can complicate matters further.

In most basic terms, sudo will run an executable with the privileges of another user. The term *executable* is important, because it means sudo won't run any shell built-ins. (It will, however, run the shell itself.)

It also is best to specify full paths to avoid a user setting a Trojan executable in a custom `$PATH` variable and executing random things with sudo. Also, remember that any redirection is done *before* elevated commands are run, so any redirection does not have elevated privileges and can cause the command to fail. This is a good thing:

```
$ sudo date > /etc/shadow
bash: /etc/shadow: Permission denied
```

By default, you keep some of your own environment variables—and this can cause various degrees of confusion and problems if, for instance, one user's .Xauthority file is overwritten by a process

running as another user. There are some tools that try to control and/or simplify this, but they also add some complexity, as well as force you to make some trade-offs between security, ease of use and reliability. There is also a diminishing return on time invested understanding every nuance of such tools.

You always can check your sudo setup with `sudo -V`. For reference, here's a list of variables on one of my systems and how they are handled by default:

```
# sudo -V
Sudo version 1.8.16
[snip]
Environment variables to check for sanity:
        LANGUAGE
        LANG
        LC_*
Environment variables to remove:
        BASH_ENV
        ENV
        TERMCAP
        TERMPATH
        TERMINFO_DIRS
        TERMINFO
        _RLD*
        LD_*
        PATH_LOCALE
        NLSPATH
        HOSTALIASES
        RES_OPTIONS
        LOCALDOMAIN
        IFS
```

Sudo Defaults

When you want to make adjustments, you need to change sudo's defaults. You set the defaults at the top of your sudoers configuration file like so:

```
# /etc/sudoers
# This file MUST be edited with the 'visudo' command as root.
# See the man page for details on how to write a sudoers file.
Defaults        env_reset
Defaults        env_keep += "SSH_AUTH_SOCK"
```

This tells the system that you want to reset all environment variables, except for the bare-bones few, but keep the variable `SSH_AUTH_SOCK`. `env_reset` usually is on by default, but it helps to remember that by making it explicit, you'll also need to define any environment variables you want to keep explicitly. These are almost always variables that one individual wants or needs to be present on a single individual system due to system role changes, and this can cause the need to start maintaining multiple individual versions of your configurations, which can quickly and exponentially increase your management overhead, so you'll want to avoid this if at all possible.

These parameters that affect the resulting executable matter a lot when you are starting a shell. There are a lot of ways to do that, from `sudo /bin/bash` to `sudo su`, and there are many ways built in to sudo to do it. What it boils down to is that you keep progressively decreasing amounts of the originating user's environment as you move through `sudo $SHELL` → `sudo su` → `sudo -s` → `sudo su - $user` → `sudo -i`.

Situations like this, on a system-by-system basis, can derail even the most security-conscious people, because they may start to choose convenience over pain of individual system management.

Doing `sudo -i` and `sudo su - root` are functionally equivalent. The hyphen in `su - $user` means to simulate an actual login; thus, you get the same environment as the user you are becoming. As you are working with system defaults, it is important to remember that some features of sudo can conflict with your directives or help you out. For instance, `sudo -i` adds a level of safety, because this is the way specifically designed to give you the root user shell safely. Since this is the design goal, it contains some additional protections to make sure you carry nothing that might compromise your system over from the originating shell and user—for instance, if you are paranoid, you might think to run sudo itself with the full path, /usr/bin/sudo /bin/bash, to avoid the possibility of using a fake sudo, but on a system-by-system basis, do you have control over every user environment variable? Situations like this, on a system-by system-basis, can derail even the most security-conscious people, because they may start to choose convenience over pain of individual system management.

The man page for the sudoers configuration file (type `man sudoers`) has an exhaustive list of possibilities, and it's important to have some understanding of them. Some

are fairly trivial and for convenience only. For instance, the following provides feedback as you type in your password upon launching `sudo -` (asterisks instead of the usual null output), which can cause confusion if you lose track of how many characters you've entered already:

```
Defaults        p wfeedback
```

Others, like this:

```
Defaults        visiblepw
```

will cause sudo to refuse to run if the echo can't be disabled on the terminal, resulting in the password being visible. These configurations can have great significance and are priceless under the right circumstances.

Other useful default is:

```
Defaults        syslog=auth
```

By default, sudo logs to syslog, which is disabled by setting a logfile. This option will set the syslog facility to be logged to, and it can be customized. If you already are doing common logging and using a remote syslog server to aggregate logs, this can help make sudo activity more easily searchable. If you don't have this, you'll need to find and manage unique log files on every machine you have. Managing log files is simple with a few systems, but it's increasingly difficult as the number of machines scales. When attempting to determine activity, whether benign or

an attack on multiple systems, ensuring that timestamps, logging level and logging efficacy haven't been tampered with are all things that are required for sudo to be useful, but are not included *with* sudo.

Some defaults are fairly arcane and require deep knowledge of the underlying UNIX/Linux architecture, for instance:

```
Defaults        umask=0777
```

The man page for the `umask` sudo option goes like this:

```
umask    Umask to use when running the command.  Negate this
option or set it to 0777 to preserve the user's umask.  The
actual umask that is used will be the union of the user's
umask and the value of the umask option, which defaults to
0022.  This guarantees that sudo never lowers the umask when
running a command.  Note: on systems that use PAM, the default
PAM configuration may specify its own umask which will override
the value set in sudoers.
```

`umask`, in case you didn't know offhand, is the default permissions used when new files and folders are created.

Some defaults can help with management overhead and monitoring of the running systems:

```
Defaults        syslog_goodpri=notice
Defaults        syslog_badpri=alert
```

These tell the system to what syslog facility to log

To achieve the “least privilege” principle, individual commands should be authorized independently, per host, per user. This can become unwieldy at scale quickly with sudo.

unsuccessful attempts and successful usages of sudo.

A fun one to turn on is this:

```
Defaults       insults
```

Try it and see what happens.

Aliases

Standard best practice is to have users execute commands as themselves, so tools should implement this by restricting users from switching accounts. To achieve the “least privilege” principle, individual commands should be authorized independently, per host, per user. This can become unwieldy at scale quickly with sudo.

As you become more and more secure, your concerns will become more and more complex, and complexity increases the odds of mistakes and unintended consequences. Keeping everything simple must be an overarching goal to keep in mind, and with sudo, balancing simplicity and security is not easy.

A More Complex Example

Let’s look at a more complex example. Say you need to allow

an operator to shut down your server for maintenance. You want the operator to be able to shut down the system, and that's all. The first thing to do might be to type `man sudo`, but the manual pages for sudo and sudoers are sterling examples of comprehensive yet impenetrable documentation. Now is when you need to understand what actually is happening in that /etc/sudoers file.

The most basic syntax of a line in the sudoers file is this:

```
USER PLACES=(AS_USER) [NOPASSWD:] COMMAND
```

Let's break that down:

- `USER` can be any existing user(s), user ID or User_Alias. It also can be a group, specifying the group by preceding it with the special character %. Groups also can be included in user aliases.

- `PLACES` can be any combinations of hostname, domain_name, IP addresses or wild cards.

- `(AS_USER)` can be any existing user(s), user ID or Runas_Alias.

- `COMMAND` can be any existing command(s) or COMMAND_ALIASES.

- `[NOPASSWD:]` is used to specify that the following commands can be run *without* being prompted for a password. Use it with caution and advisedly. Understand the risks and have compensating controls for those risks.

You could allow the operator to do the job simply with:

```
operator ALL= /sbin/shutdown
```

But, what if you need more flexibility than that? What if you need multiple users from multiple places to be able to do a more complex combination of things?

Aliases increase both complexity and flexibility. Using them either can increase or obliterate readability and, thus, the maintainability of the file. Here are the types of aliases available:

- User_Alias
- Cmnd_Alias
- Host_Alias
- Runas_Alias

I find user aliases to be a bit redundant, as you can use regular groups:

```
User_Alias USERS = tom, dick, harry
#OR
User_Alias ADMINS = %admin
#OR EVEN
User_Alias ADMINS = +admin
#USING “+” indicates this isn’t a local group defined in
/etc/group but a network group
#There are special operators available like negation “!”
```

```
and you can combine Aliases with them
User_Alias LIMITED_USERS = NET_USERS, !WEBMASTERS, !ADMINS
```

You also can include local and network groups in your basic line syntax, so I've rarely needed to use user aliases. Command aliases, on the other hand, are a different story, but I'll get to that in a bit. For the example above, you could create a user alias for operators:

```
User_Alias OPERATORS = tom, dick, harry
```

Maybe your operators need to do a few more things, like use local printers:

```
Cmnd_Alias     PRINTING = /usr/sbin/lpc, /usr/bin/lprm
Cmnd_Alias     OPERATIONS = /bin/shutdown, /bin/kill
```

And now you have a string of aliases to use and combine (and recombine), so you can replace the operators line above like so:

```
OPERATORS localhost = PRINTING, OPERATIONS
```

Now your operators can perform select functions only when logged in locally. With this single simple example in mind, now think about how to scale this across a large organization with hundreds of servers, tens of groups and hundreds of people who all may need different privileges across each host.

Other Security Tools

With these basic examples, you should be able to see how

With sudo, enterprise control is limited, which means the tool is not designed for nor does it easily lend itself to centralized command and control with easy configuration.

you can combine simple abstractions to lock down a system to any desired degree, at least as far as execution privileges go. There are other layers of security to consider, but sudo doesn't give you file protection or tampering notification, which are also essential to a holistic security approach. For that, you need other tools. All of this layering of tools and protections does come at a cost—a cost you have to work hard to mitigate before the environment becomes unstable, unusable and unpredictable.

With sudo, enterprise control is limited, which means the tool is not designed for nor does it easily lend itself to centralized command and control with easy configuration. Because of this, commercial tools have arisen to fill those gaps. As mentioned previously, the time when sudo had a competitive advantage for modern environments is gone.

Commercial solutions now can provide a wealth of capabilities that sudo doesn't. Foremost among them are full session and event logging, centralized policy controls, and file integrity monitoring. Products such as PowerBroker for Unix & Linux by BeyondTrust provide the ability for corporations to meet even the most strenuous audits related to privileged access and compliance-related activity.

Sudo Shortcomings

Sudo was designed and built when IT organizations put a lot of effort into securing each independent and expensive server separately, per its purpose. Given the expense of compute resources, this made a lot of sense. The 50-year deflationary boom in that cost has allowed digital technology to become ubiquitous, with new capabilities layered on top of what was nearly miraculous yesterday. Spending the management time to achieve security per server is no longer viable, as in the age of DevOps, you don't manage servers anymore. In fact, when you get good at this, you hardly even manage environments. The world has moved on, and admins are moving on with it to managing "Platforms as a Service".

When you are operating at scale and speed, sudo's shortcomings become all too apparent. You must not only understand how to secure your systems using sudo, you also must understand how to automate that and do it to hundreds or thousands of servers at once. What's more, those servers are increasingly ephemeral—like virtual quantum particles, they are popping in and out of existence rapidly. Legacy management techniques no longer will work as newer and larger sets of capability are stacked on top of the old. IT is being re-invented continually, and security must be re-invented continually as well to go along with it.

A big problem is that those of us graybeards who have expertise in utilizing sudo gained it by exhaustively securing those big expensive immortal servers of yore. This is a world many young engineers will only hear of in legend. Most executives don't know this is an area in

which young engineers need training and experience. All too often, instead of doing the work to layer in security and permissions control, because people don't have the time, tools or underlying expertise to evolve it, they drop it or at least stop caring so much about it. This is a terrible mistake given the high-profile data breaches that keep dominating the headlines. You can't afford to get this wrong. CEOs and CIOs increasingly are being held accountable for failing to ask the questions about how they are accounting for advancing security even as capabilities in the rest of technology are advancing. Security must be a partner in this advance.

Next-Generation Tools

Security used to be a highly manual process. It simply cannot be any longer. The application of security must be automated, no matter the platform. For this, you need next-generation tools. You need solid configuration management, starting with tools like Puppet and Chef. You need strategies for distributing solid fine-grained application and user permissions across platforms, as you create and re-create servers and services. In this new world, you need to define less about individual users, as you may have management tools doing most of that work. Until this transition is fully implemented, a day that may never come, you always will have these permission administration concerns—and even then, you'll have to define the permissions and privileges your tools themselves need, lest they themselves be hijacked.

Admins are dealing more and more with administrating services and less with servers. The concept of a server

becomes less defined as you start using containers and unikernels. You need to update your tools and strategies for the times, and they should be able to do a lot of the work for you. To do this, there are several basic strategies. One strategy is to augment sudo, and another is to replace it with a set of tools that is designed to be centrally managed, fully automated and that can hold up to the most stringent of audits.

This is where next-generation cross-platform tools that drain some of the complexity can help. You need a new methodology for privilege management that makes it simpler at scale. While you are reconsidering legacy practices, you have an opportunity to explore new kinds of benefits:

- What if you could use one tool for several of your security use cases, like centralized policy management, automatic service discovery, vulnerability management, centralized audit logging and single sign-on between *nix, Mac and Windows operating systems?

- What if you need to automate not only the dynamic provisioning of users, but also the assignment of permissions, at scale, for various roles? This is difficult to accomplish with text-based, distributed sudoer files.

- What if you could integrate with sudo where it makes sense and replace sudo in other cases?

- What if you could apply your Active Directory policies across your *nix environments?

- What if you could augment your container isolation, provide centralized audit logging and do file-level monitoring inside them as well?

- What if you could have one go-to place for your centralized approach to holistic security?

- Does it make sense never to re-think your use of 20-year-old tools with only incremental improvements while your methodology has moved on to manage entire platforms as code?

BeyondTrust has just such a toolset that can help with all the weaknesses and complexity stemming from the legacy design of sudo and provide additional benefits as well. This is certainly worth a look as an alternative to de-prioritizing security due to lack of expertise or time to administer labor-intensive tools. Sudo is a venerable tool with a long history in privilege management, but you need to move beyond it into next-generation methodologies so security and compliance can keep up with the pace of the IT revolution.

Key Capabilities in Achieving Advanced Security and Compliance Use Cases on UNIX and Linux Platforms

Consider the following checklist when determining whether sudo is meeting your security and compliance needs. If sudo can't help address these use cases, it's time to think about a commercial UNIX/Linux privilege management solution.■

Table 1. Checklist: Does Sudo Meet Your Needs?

Capability	Function	Benefits
Auditing and Governance	Analyzes user behavior by collecting, securely storing and indexing keystroke logs, session recordings and other privileged events.	Speeds forensics and simplifies compliance by providing an unimpeachable audit trail of all user activity.
Fine-Grained Least Privilege	Elevates privileges for standard users on UNIX and Linux through fine-grained policy-based controls.	• Enables compliance through the compartmentalization of IT tasks that require privileged accounts. • Limits attack surfaces by providing just enough access to complete a task. • Eliminates admin rights from managed systems.
Dynamic Access Policy	Utilizes factors such as time, day, location and application/ asset vulnerability status to make privilege-elevation decisions.	Reduces attack surfaces by helping IT make privilege decisions based on context and risk.
Remote System and Application Control	Enables users to run specific commands and conduct remote sessions based on rules without having to log on as admin or root. When combined with integrated privileged password management, elevated applications can be launched without exposing the password.	• Enhances user productivity by simplifying processes that are complex with native tools or sudo. • Limits attack surfaces by preventing the use of the root and admin account. • Keeps systems safe by only allowing approved applications and commands to be executed.
File and Policy Integrity Monitoring	Audits and reports on changes to critical policy, system, application and data files.	• Reduces risk of tampering by ensuring that critical files have not been altered. • Protects critical files from malware and privilege misuse. • Eliminates unauthorized software installs, workarounds or gaps that could lead to exploits.
Privileged Threat Analytics	Correlates user behavior against asset vulnerability data and security intelligence from best-of-breed security solutions.	Reduces risks in user activity that can lead to data breaches.