The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

Priv: Secure and Flexible Privileged Access Dissemination

Brian C. Hill
University of California, Davis

Priv: Secure and Flexible Privileged Access Dissemination

Brian C. Hill – University of California, Davis

ABSTRACT

Large UNIX installations have become increasingly demanding of ever growing staffs of system administrators. Since very few system management tasks can be performed without access to root, large sites quickly face a ‘‘too many cooks’’ problem. The problem emerges clearly out of the numerous existing, yet only partial solutions to the problem.

Examples of broad access needs are numerous: Help desk staff need to be able to examine user’s files to assist over the phone or change passwords. The ability to *su*(1) to only non-root users is also be useful. System operators need to manage print queues, kill jobs and reboot systems. Neither group, however, should necessarily need access to to pids or files owned by root, making blanket access to *kill*(1) and *cat*(1), for example, potentially problematic. Users in a research lab need to mount cdroms. In walk-in consulting areas, securing access to commands via the privileged user’s own passwd prevents unattended terminals from being sabotaged. In our environment at UC Davis, we have all of these groups of users and none represent core system administration staff, the only true root users. Several packages and other mechanisms address this issue, but even most of the more expensive commercial packages fail to allow privileged system access that is both secure and flexible.

Access Control Issues

Root access control and auditing/logging fall under the category host-based security. Root access control determines the conditions necessary for a user to successfully invoke a privileged command. The purpose of access control is not only to prevent or limit malicious activity, but to mitigate detrimental accidental activity as well. For example, if *kill*(1) is inadequately protected, a user might accidentally mistype the pid argument to *kill*(1) and kill an important system daemon. The usefulness of auditing and logging begins after a user has successfully launched a command.

Access restriction means are numerous. Simple methods include restricting program invocation by user, groups of users, host, and user password. More elaborate means include restrictions on time of day, point of access (tty, remote host), additional challenge passwords, arguments (system objects and their owners, numbers of arguments, options). Advanced methods usually involve system call tracing or intercepting. The methods available have largely been defined by the various existing packages, but a summary of methods can also be found in the National Computer Security Center’s (NCSC) ‘‘A Guide to Understanding Discretionary Access Control in Trusted Systems’’ [NCSC1]. Last updated in 1987, this document is admittedly dated, but useful as reference.

The user and group restrictions are the most basic access restrictions. They are supported by all packages and methods (referred to hereafter as just packages).

The host restriction is more of a configuration convenience, allowing centralized configuration to be used across several hosts.

The password restriction is an interesting concept. It requires a user to enter his/her password to execute a privileged command. The advantage is that users do not need to be given passwords to other logins, most importantly to the root login. There are fewer passwords for a user to remember (just his/her own, generally) and it is easier to revoke privileges for user without having to change a password and distribute the new password to all the users that do need it. The unfortunate consequence is that the more users there are that have this kind of access, the more passwords that will float around that can be used to gain privileged access, since passwords can potentially be sniffed as they are typed over an insecure network. Password challenges (discussed later) and single-use passwords provide an important way to mitigate this problem.

The time restriction limits access by time of day or week, etc.. The difficulty with time restrictions is the extent to which time syntax can vary.

Point of access restrictions, which limit access by a user’s location, include tty and remote host restrictions. A tty restriction can be used limit access to a given command to only the console, requiring a user to be on-site to gain privileged access. Restricting tty access can also limit batch-mode operations, when that is deemed to be undesirable, by disallowing input and or output to be redirected from or to a plain file.

Unfortunately, programs like 'expect' [Libes] will foil such restrictions by allowing batch input to be supplied under the guise of a regular tty.

Restricting access based on remote host can be done one of two ways. The most portable way is with the information from from /etc/utmp, which can be used in a straightforward fashion, but this information is often inaccurate and, on older systems, writable by the user. This may become more viable in the future as more version of UNIX make this information non-world-writable and more difficult to alter in general. The other, much more solid method is to use the socket information behind the input file descriptor, but the kernel data structures needed to derive the remote host vary widely between operating systems. Lsof [UPurdue] reportedly contains code to accomplish this.

The time and point of access restrictions be can used to limit access to employees after business hours or away from the office [FSA2].

Password challenge mechanisms take a variety of forms. Some are as simple as requiring a second user passwd contained in password database, possibly encrypted, separate from the UNIX passwords. Others simply require additional information, including social security numbers, date of birth and mothers' maiden names. Better mechanisms include those that require the password of another user (safe-deposit box style). The best mechanisms implement single-use passwords, like DES Gold [Enigma]. Some privileged access packages support these external single-user password packages.

Explicit setting of user id to other than root, changing of the umask and setting environment variables are also ways to improve security and flexibility. A program that only needs to be run as a miscellaneous system user, should not be run as root. The print job cancelling command *cancel*(1) can also be run as user 'lp' on most systems. A command that appears in different places on two different operating systems can be configured as one command in the privileged command database, by setting the PATH only to include the two locations where the command is located. Many commands in /usr/etc on older UNIX systems and /usr/sbin on newer systems fit this profile. Setting the umask to explicitly prevent creation of world writable files as the result of a *cp*(1) command is also useful.

The restriction mechanism least utilized by packages focus on the command line arguments, including number, option clusters (also known as switches), relative argument position and, most importantly, arguments that represent system objects. Restricting the number of arguments is often useful when command or an activity cannot be tailored to totally eliminate the potential for damage. This has the effect of slowing down the accidental damage a user might be able to do, by requiring repeated interaction to accomplish a large task, like chown'ing or chmod'ing large lists of files. The options restriction can be used to eliminate the case where a user might supply a '-R' to chown. A simple string comparison is not sufficient as options can usually be mixed with other options, forcing all permutations of available options for a command to be specified. Many commands will simply ignore unknown options, increasing the number of permutations that would have to be specified. More specifically, restrictions can be placed on system objects presented on the command line, like login ids, files, pids, and print jobs. Systems that implement restrictions on system objects are often referred to as 'capability-based' systems [NCSC1]. This is the most powerful and flexible of all restrictions. A user can be granted access to *kill*(1) all pids not owned by root or other important system users, like daemons. Preventing a user from specifying arguments in certain places on the command line can also be controlled with a restriction that causes other restrictions only to affect certain arguments. The typical use of this would be with commands like *mv*(1) and *cp*(1), where the targets (which always appear at the *end* of the command line) are more vulnerable than the sources. A *cp*(1) command could be configured to allow copying *from* files owned by any user, including root, but only *to* any file and directories *not* owned by root. Any ownership(s) of destinations can be specified.

A template is a way to specify mandatory parts of a command, which accomplish a couple of odds and ends. They allow commands with long argument lists that don't vary much, like adduser programs, to be shortened to just a few arguments, simplifying procedures for privileged users. A template also a allows a command to be configured to allow arguments in a specific order. If you want to restrict an adduser program to not allow "root" to be added or changed, you could specify simply that no "root" strings appear in on the commands line, but, for example, if the gecos field needed to contain the string "root", this would not be practical. Argument position restrictions would not work because many adduser programs, like Solaris 2.5's *useradd*(1), allow option and corresponding arguments to be specified in any order. In this situation, a template can be created that nails the options to a particular location in the command line, requiring the user to only supply the arguments to those options, thereby allowing restriction to only be placed only on certain arguments.

Executable checksums on binaries are another safeguard in a privilege dissemination mechanism that reduce the likelihood of Trojan Horses, but the information is difficult to maintain. Anytime a binary is patched by the operating system vendor or a locally maintained binary is updated to a later version, checksum information in a checksum database must be changed.

Configuration languages/syntax also vary between packages. Some use simple tables and others use fully BNF (Backus-Naur Form) languages. Some languages use a syntax that falls 'in between', similar to a *printcap*(4) file.

Though each type of restriction has its weaknesses, taken together, they provide some very flexible, yet secure means of disseminating privileged access. Priv attempts to just that.

Auditing

Logging of privileged activity takes many forms, from recording general details about the access, to full session capturing. The simplest information includes the user's login id, tty, current working directory, command plus arguments and time of execution. In some cases, just syslog(2/3) us used, but other packages manage their own log files. More elaborate schemes encrypt logged information and send it to a central log host. Some packages also offer full session capture capability.

While many commands can be configured securely to give access to a wide variety of users without much risk, some commands inherently remain at least partially vulnerable. Non-writing commands, like ls, though potentially intrusive, pose relatively little threat if configured properly. Logs of these commands might be of little interest. Commands like *cat*(1) and *kill*(1), however, present a slightly increased threat that a system manager would probably want to monitor more carefully, but still with a simple log. In some cases, for example, it might be desirable to give experienced customer service agents full *su*(1) access to group of regular user accounts (but not root or daemon, for example), allowing more flexible privileged movement, and therefore better customer service. What a system manager depends on at that point, is a full session capture. Auditing obviously becomes a tricky task, when a privileged user has enough access to be able to erase the logs. Sending audit information to a remote log host, which itself must presumably have to have more secure access than the others, largely alleviates this problem. Encrypting logs is yet an additional measure that makes alterations nearly impossible. In the event that a log were kept locally on a system, a user would be forced to remove an entire log to cover up a an illicit act, raising a red flag to system managers. Another way to handle sensitive commands is to have log information appear on the system console or other designated ttys and/or have mail sent to system managers with the log information just before command execution.

More general guidelines for auditing are available in the NCSC's "A Guide to Understanding Audit in Trusted Systems" [NCSC2].

The Packages

The simplest solution could be one using UNIX groups, where setuid copies of some key unix commands, like *cat*(1), *kill*(1), *lprm*(1), and *ls*(1) could be placed in a directory created for privileged commands with execute permission set only for certain groups of users. This solution lacks log keeping and additional protections like per-use passwords. Additionally, in an environment where users would only need certain privileges on certain systems, different group files would need to be maintained on different systems, increasing management tedium.

Some vendors offer home grown solutions that lack some of the same flexibility as the third party tools do, but most importantly, they lack cross-platform compatibility. Access Control Lists (ACL's), for example, offer a very precise mechanism for granting access to specific files to specific users. Among the limitations of ACL's is that some platforms don't have them and those that do, including SunOS 5.X, DYNIX/ptx and HPUX, each have slightly different implementations of ACL's; this makes it difficult to devise a single, installation-wide strategy in a heterogeneous installation. Also, ACL's are limited to files, they cannot be used to protect process ids in a similar manner. Lastly, removing users and managing global permissions is cumbersome as operations have to be performed on a per-file basis.

Though the focus here is on UNIX tools, mainframe and mainframe-like operating systems have long had native privilege dissemination mechanisms more elaborate than UNIX has ever had natively. IBM's "Resource Access Control Facility (RACF)" [IBM] on MVS is such an example, allowing read/write permissions to be granted to system objects by owner. VMS has had a privilege level mechanism since its inception that allow over 15 kinds of access for privileged use.

SuSub [TLogic] is the simplest of the commercial products discussed here. SuSub provides only simple user restriction, password protection and simple logging of commands.

Sudo [UColo] is a popular public domain that allows user, host and password restrictions. It support simple logging. Sudo's password restriction allows for a five minute (default) validation period. If another command is entered within five minutes, no password is required and the 'ticket' is updated. Sudo sends mail to a designated user whenever a attempt to invoke a command with Sudo fails. Sudo's main advantage is that it is simple to configure and supports many operating systems.

PowerBroker [FSA1] is a fairly comprehensive package that offers almost every feature mentioned except robust control over command line arguments, a significant drawback. While it does offer some command line argument control, it lacks control over command line arguments that might represent system objects.

Other solutions like Exu [RammGrubb] seem to focus more on the secure network aspect of implementing secure privilege dissemination, rather than the dissemination details, and also more on centralized system management. Exu uses Kerberos.

Ideally, flexible access restrictions should be provided at the kernel level, at least as a drop-in module, like SeOS [Memco1]. This very complete product offers, essentially, every features mentioned. These types of products are even more expensive, however, then other commercial products, like Power Broker. They can also lead to increased system overhead as all suspect system calls by all users on a system are intercepted by a kernel 'user exit' routine. SeOS also does not yet support all major UNIX operating systems.

What all of these solutions lack, however, except SeOS, is the unity of all of their features in one package and, most importantly, robust control over command line arguments that might represent system objects, like files, process ids and print jobs, in a way that is consistent. A user that had kill privileges could kill anything including sending a TERM signal to init causing a system run level change to single user. A user that could cat files, could examine anything on the system. The packages and other mechanisms mentioned address the basics and some more elaborate aspects of privileged access, but even the more expensive commercial packages fail to allow privileged system access that is both secure *and* flexible.

Priv offers a highly flexible approach to disseminating access to privileged commands that is secure and easy to manage in a large installation, on a variety of UNIX platforms. It ties together most of the features of other packages and adds control over command line arguments. The basic strategy is such:

1. A privileged user runs a command with priv:

```
$ priv ls ~smith/
```

2. Priv looks up the command in a policy description file or NIS map and retrieves a restriction list.
3. Priv checks a series of conditions and/or requirements in the restriction list that must be satisfied in order to execute a command. This might involve tests on user-supplied arguments, password prompting, resetting of uid or uid, user and host checks, and more.
4. A syslog entry is made and a system console log is made.
5. A final command is assembled and executed.

A command description file contains per-command descriptions of restrictions and conditions that must be passed before a command can be executed. This file uses a format similar to a printcap file and can be accessed as a file or via NIS. Priv supports all of the following features on a per-command basis:

- Inclusive or exclusive restrictions – Conditions containing lists of items can be inclusive or exclusive.
- Restriction by user – Commands can be restricted by user, group or netgroup of users.
- Restriction by host – Commands can be restricted by host or netgroup of hosts.
- Restriction by operating system type – Commands can be restricted by operating system of current system. This makes it easier to tailor one command across platforms.
- Password protection & double password protection – Passwords can be required to execute commands. Additionally, passwords for additional neighboring privileged users, like coworkers in neighboring cubicles, can be required to add safe-deposit box-style security to command execution. Support for several vendor shadow passwd implementations is also included. This is useful for a consulting environment where one consultant would need the acknowledgment of another to execute a sensitive command like 'passwd' or 'chown'.
- Syslog(2/3) logging options – Syslog facility and level can be tailored for each command.
- Session capture – Capture session dialogue for later review.
- Setting uid and gid to other than root – Commands are executed as user root and group 0 by default, but this can be changed to any user and/or group specified.
- Setting umask – Umask can be set to specific value.
- Setting environment variables – Environment variables can be set, like PATH. This also enables single command configuration across platforms.
- Argument range – Conditions that affect arguments can be limited to certain arguments or one or more ranges of arguments specified by position numbers.
- Configuration conjunctions – "or" and "and" can be used to join multiple restrictions under one command. See below.
- Built in kill, su, cd and pwd commands – Built in commands increase execution speed and add execution behavior consistency when vendor versions of commands, like su, behave differently from vendor to vendor. Priv will also simulate cd and pwd.
- Final command rewriting template – This allows parts of a command to already be specified, taking additional arguments supplied by the users only as indicated by macros in the command rewriting template.

The most powerful aspect of priv is its robust ability to restrict arguments to commands. This is where all other well known packages fall short. Arguments can be treated as system objects – pids, files and login names.

- File ownership – Arguments that are files owned by a certain user or users in a netgroup can be restricted by limiting access or excluding access to files they owned. For files that

don't exist (i.e., the destination of a cp command), the ownership of the directory is used.
- Files – Arguments can be taken as filenames. The inodes and devices of file arguments and files listed in this restriction are compared. If this were done as string comparison, symlinks and other tricks could be used.
- Processes ownership – same as file ownership, but for pids.
- Strings – Arguments can be subject to inclusive or exclusive string, substring and regular expression comparisons.
- Login names – Arguments are resolved to uids to handle cases where uids are specified instead of login names or in cases where more than one login name maps to the same uid.

The following are some examples of the commands from a configuration file that use many of the conditions and requirements listed:

```
kill     :user=!joe,@systems: \
         :str=!.-9,!#.-kill: \
         :argv=/bin/kill,$*: \
         :or: \
         :user=joe: \
         :pid=!root,!bin,!daemon: \
         :str=!.-9,!#.-kill: \
         :argv=/bin/kill,$1:
```

Anyone from the systems netgroup, except user 'joe', may kill any job, but NOT with signal 9 (as '9' or upper or lower case '-kill'). 'joe' may only kill jobs not owned 'root', 'bin', and 'daemon', he may only kill one job at a time and he may NOT specify '-9' (or '-kill') on the commands line.

```
cp       :user=@consultants: \
         :argr=-1: \
         :fowner=!root: \
         :argv=/bin/cp,$*:
```

Consultants may cp any file, but may not cp file *to* any destination owned by root. The 'argr=-1' sets the argument range for later restrictions, causing the fowner parameter to only affect the last argument in the user supplied list.

```
chown    :user=@consultants: \
         :opts=!R: \
         :str=!root,!*root.\.*: \
         :os=!SunOS 5: \
         :setenv=PATH\=/usr/ucb\: \
         :/usr/bin: \
         :argv=chown,$1:
```

Consultants may chown any file to any user except root or any root.* combination (as allowed by most chown incarnations). The 'str'ing test performs a *strcmp*(3) test on the first item in the list and a regular expression match on the second item. The chown command can be found in the supplied PATH. The '-R' option cannot be supplied (as -Rf -fR or -R) and chown cannot be run on a SunOS 5.X system.

```
su       :passwd: \
         :and: \
         :user=@systems: \
         :argv=su,$*: \
         :or: \
         :user=@consultants: \
         :login=!root,!daemon: \
         :argc=+1: \
         :argv=*su,.-,$*:
```

Require the user's passwd to su. Allow users in the systems group to su anywhere. Consultants may only su to non-root accounts. The login restriction above will not allow any argument to be specified that maps back to the uid that root owns or the uid that daemon owns. The argc restriction requires that at least one argument be specified, also preventing the user from su'ing to root by default.

Except when capture mode is required by a condition, priv does not fork, it simply execs assembled commands. Later versions will allow logging of exits codes from exec'd commands.

Priv is written in about 5000 lines of C code and compiles into roughly a 65K binary on most systems with shared libraries. Priv is written to use the largest intersection possible of system calls and library routines from system to system. It is as POSIX.1 compliant as possible. Priv goes to reasonable lengths to improve speed, for example, by eliminating unnecessary copying and comparing of strings. Priv currently builds on at least the following platforms:
- AIX 3.2-4.3
- DYNIX/ptx 4.0
- HPUX 9.0-10.0
- IRIX 4.0.5-5.3
- Linux 1.0 (various implementations)
- OSF1 2.0-3.2D (Digital UNIX)
- SunOS 4.0-5.5 (Solaris 1.0-2.5)
- Ultrix 4.0-4.4

What Priv Needs

The restriction language implementation is fully extensible, making it easy to add new conditions and restrictions to the program very easily.

Priv should offer a few other features:
- Checksum checking on executables to avoid Trojan Horse problems
- A system call trapping mechanism using UNIX *ptrace*(2) to allow secure use of vi and other programs with shell escape commands
- Restriction on time of day execution
- Support for additional challenges, like single-use password generators, etc.
- Support for additional vendor-unique shadow passwords sources
- Support for additional restriction databases and formats
- Default templates for handling general cases for faster configuration

- A graphical user interface
- Log file encryption
- Possibly enhanced language
- Control over additional system objects like print jobs
- Allow mail notification upon command execution

Other Security Issues

The mere discussion of security often heats into discussion of any weakness that package has, whether or not those weakness should be legitimately addressed by that package. The related issues require action and those that aren't deserve at least awareness. Security issues involving setuid programs and potential weaknesses in configuration languages must be addressed. After all, a tool specifically designed to enhance security should be the last to detract from it.

The nature of Priv's language makes it possible for sensitive commands to be configured insecurely, but this is an almost unavoidable side effect of such a flexible language. Other than the minimal checks Priv offers to avoid totally insecure configuration, it is difficult to determine how Priv, or any package like it, could even largely prevent such problems. To start, the simplest case could be examined, where a command is configured with only a user restriction. This means that Priv could require at least one restriction, and this might be reasonable, but some might argue that this isn't enough. If priv were to require any two restrictions, say user and host, it would no longer be flexible enough to tailor the wide range privileged access that it is capable of. Suffice to say, a system manager at the helm of a Priv configuration file, or any other such tool, should be well versed in security issues as s/he bears an inalienable burden of careful configuration.

Setuid programs have long taken a beating. Most of the weaknesses in setuid programs stem from logistical oversights and weak programming practices. Setuid weaknesses are pointed out, in general, in a paper presented by Karl Ramm and Michael Grubb at LISA IX [RammGrubb] and by NCSC document on access control mentioned earlier [NCSC1].

The most common logistical errors involve the file ownership and modes, especially with configuration files and temporary files. While some trusted programs have been known not to check any file permissions at all, like *rlogind*(1) on many systems, the most common problem is the race condition. A program running as a trusted user, usually root, often checks file permissions for proper ownership and mode and *then* opens a file. Intruders write programs that cycle quickly to create a symbolic link pointing to /.rhosts (or some other vulnerable file) between the time the file permissions are being checked and the time the opening and writing begins. This is more common with temporary files that such tools would need to open in /tmp, which is a world writable directory. The race condition can be mitigated by opening a file first and, and then checking permissions and ownership on the open descriptor, but many tools still do not seem to do this. The more obvious logistical issues can be addressed largely by increased conscientiousness in program design. More information about setuid programs and methods to test them can be found in ''Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring'' [UCDCS1]. The other programming weaknesses, however, are related to oversights that are less intuitive.

Inexplicit management of signals, signal handlers, inherited environment variables and maximum string lengths in programs has also created many back doors in setuid programs [Dilger1]. Despite the increasing visibility of these problems, setuid programs are still used. PowerBroker, Priv, Sudo, and SuSub all depend on the setuid bit. The increased awareness of secure setuid programming can probably be said to have made setuid programming more viable than it once was thought to have become.

Priv avoids race conditions and explicitly ignores all available signals that it does not use, resetting them just before executing the user-specified command. Environment variables are not affected directly, but Priv allows environment variables to be controlled via the command configuration restriction 'env'. All cases where buffer overruns might be possible have been carefully avoided. Lastly, and possibly most importantly, Priv assumes the user's id for as much of its processing as possible.

In addition to the aforementioned areas of concern, it is important to limit the focus of the discussion by reminding the reader what the discussion is not about, especially regarding network security issues.

Obviously, priv is not a passwd changing program like passwd+ [Bishop], a secure passwd program like DES Gold [Enigma], an enhanced *login*(1) replacement like FSA's PowerLogin [FSA4], a system security integrity auditor like COPS or Tiger or a firewall. The vendors of PowerBroker, SeOS and SuSub are specialists in security packages and do offer a wide variety of such packages. Some of these are bundled with their access restriction packages discussed here, but *not* out of necessity to accompany a privileged access tool.

Most importantly, priv does not concern itself with network security. There is no encryption of traffic or attempt to authenticate any network connections. Priv only *uses* network services, like NIS, and does pretend to implement them. Concerns over network security are better left to packages like Kerberos. Priv's goal in the matter would be merely to support many database sources/engines, including more secure engines like NIS [Sun] and eventually NIS+ [Sun].

Ease Of Management

The easiest packages to manage generally allow centralized configuration for many hosts and include GUI management tool. Packages that do not allow host names to be included in the configuration, including the unix group method, SuSub and ACL's. PowerBroker, Priv and SeOS all allow configuration across hosts. SuSub, PowerBroker and SeOS provide a GUI management tool.

Performance

Performance tends not be an issue with most of these packages. The time to verify authorization and record audit information is generally quite short, usually under one tenth of one second. The only notable exception here is SeOS. Since SeOS will intercept certain system calls made by all users, whether or not a user has any authorizations for privileged action and whether or not a user attempts a privileged action, system performance will degrade slightly. Rough data provided Memco [Memco2] suggest anywhere from a 1% to a 15% drop in system performance, with the average drop being about 4%. This might be an issue for some system managers.

Price

Costs of these solutions vary widely with the features they support. The no-cost alternatives include, in order of robustness, the UNIX groups method, vendor bundled mechanisms, the partial solution of ACL's, Sudo, and Priv. The cost of commercial packages can only be roughly summarized due to the discount incentives for quantity, educational use, platform power, etc.. In order to provide a comparison, basic commercial pricing schedules for one system and thirty systems are listed. Again, discount incentives may radically change prices. Pricing listed here is subject to change in general and is intended only for comparison. In cases were platform size affected price, the least expensive platform price was used.

Prices quoted as of August 10, 1996

	1 copies	30 copies	
SuSub	$295	$6870	[TLogic]
PowerBroker	$500	$9000	[FSA3]
SeOS	$8,000	$80,000	[Memco3]

Conclusion

Most of these products do not go far enough and those that do are potentially pricey, depending upon the number of copies needed. The commercial product SuSub isn't even as flexible as Sudo, a limited, but seasoned public domain tool. PowerBroker is fairly comprehensive package, but lacks control over system objects. SeOS is a slick and reasonably complete product, but it is literally two orders of magnitude more expensive than the other commercial products.

Priv addresses nearly every aspect of privileged access dissemination in way that allows a high degree flexibility and security across many platforms in a large installation. Commands to accomplish very specific tasks at specific levels of access can be tailored quickly and safely. Priv combines most of features of free and commercial products into one mechanism, which is itself free. Priv is also written in such a way that it can be easily extended. The Information Technology department at UC Davis has been using priv in several units for nearly four years, as well as at a couple of private companies, a true indication of the soundness of Priv.

Acknowledgments

The late Sam McCall, Systems Programmer, U.C. Davis. Sam mostly inspired this package. Sam was always one of my best UNIX resources. This version of Priv is dedicated to him.

Dan Dorough, Systems Programmer, U.C. Davis. Dan is still the sharpest, most solid UNIX and VMS programmer I know (if not the most old fashioned!). I doubt that I could have learned anywhere else what I learned from him about programming.

Michael Dilger, Programmer, Sun Microsystems. Mike gave me numerous pointers to help me tighten the security of the code in Priv considerably as documented above.

Dr. Chris Wee and Todd Heberlein of the Computer Security Research Laboratory and Greg Benson of Software Verification Laboratory, both of the U.C. Davis Computer Science Department. They answered all kinds of my questions about writing technical papers.

Kathryn Hemness, System Administrator, U.C. Davis. She keeps converting my shabby text man page to a *real* man page format.

Bryan Stansell, Global Networking and Computing, and Kanbiz Aghaiepour, TCSI, who worked in the trenches with me at U. C. Davis when we were students.

Dave Zavatson and Dana Drennan, System Administrators, U.C. Davis. Miscellaneous contributions.

Everyone else in Net Land that gave pointers, tips, code and other advice.

Software Availability

Retrieve the software from ftp://ftp.ucdavis.edu/pub/unix/priv.tar.gz, though the directory location may change in the future.

Author Information

Brian C. Hill finally got his B.S. from the University of California at Davis in 1993. He has worked part time for U.C. Davis in various systems programming and consulting roles for seven years, currently with the Graduate School of Management and the U.C.D. GIS Center, and as an independent contractor for over five years. In addition to programming Priv, his specialties include aggressive distributed systems administration, sendmail and DNS configuration. He can be reached at bchill@ucdavis.edu.

References

[Bishop] P] Passwd+, Dr. Matt Bishop, University of California, Davis.

[NCSC1] Carole S. Jordan, ''A Guide to Understanding Discretionary Access Control'', in Trusted Systems (NCSC-TG-003), September 30, 1987, National Computer Security Center, pp. 7-11.

[NCSC2] A Guide to Understanding Audit in Trusted Systems (NCSC-TG-001) September 30, 1987, National Computer Security Center, pp. various.

[Dilger1] Michael Dilger, personal communication, University of California, Davis, Computer Science Department, dilger@cs.ucdavis.edu,

[FSA1] Logging and Auditing root Actions with PowerBroker, FSA Corporation http://www.fsa.ca/pbtech.htm

[FSA2] Power Broker Audit White Paper, FSA Corporation http://www.fsa.ca/pbaudit.htm

[FSA3] Paul Scripko, personal communication, FSA Corporation pscripko@fsa.ca

[FSA4] PowerLogin, FSA Corporation http://www.fsa.ca/powerlogin.htm

[Memco1] SeOS, Memco Software http://www.memco.com/products/Seos.htm

[Memco2] Laura Gillespie, personal communication, Memco Software, (800) 862-2602 x401, laura@memco.com.

[Memco3] Cecilia McClaine, personal communication, Memco Software, (408) 559-8520 x125.

[TLogic] SuSub, Technologic, Inc. http://www.tlogic.com/ susub.htm

[Enigma] DES Gold, Enigma Logic http://www.safeword.com/

[Libes] Expect, Don Libes

[UColo] Sudo, University of Colorado http://www.cs.colorado.edu/˜millert/sudo

[RammGrubb] Karl Ramm, Michael Grubb, ''Exu – A System for Secure Delegation of Authority on an Insecure Network'', Proceedings of the Ninth Large Installation Systems Administration Conference (LISA IX), September, 1995. http://web.mit.edu/afs/athena.mit.edu/user/k/c/kcr/www/exu.html

[UPurdue] Lsof, Vic Abell, Purdue University ftp://vic.cc.purdue.edu/pub/tools/unix/lsof

[UCDCS1] C. Ko, G. Fink, K. Levitt , ''Automated Detection of Vulnerabilities in Privileged Programs by Execution Monitoring''. Proc. 10th Annual Computer Security Applications Conference Orlando, FL, 5-9 Dec. 1994, pp. 134-144.

[IBM] Resource Access Control Facility (RACF), IBM http://www1.ibmlink.ibm.com/HTML/SPEC/g2214102.html

[Sun] NIS+, Sun Microsystems Incorporated.