A Mechanism for Sharing Accounts

Matt Bishop

March, 1987
Technical Report 87.10

RIACS

Research Institute for Advanced Computer Science

A Mechanism for Sharing Accounts

Matt Bishop

Research Institute for Advanced Computer Science
NASA Ames Research Center
Moffett Field, CA 94035

ABSTRACT

A serious problem in account management is how to determine accountability when several people must share access to one account. Further, password management for such an account is extremely complex. This paper describes a mechanism to overcome these problems, and an implementation of the mechanism under UNIX.

Overview

The Numerical Aerodynamic Simulator project runs a variety of UNIX† based operating system, on its computers (a Cray 2, 2 Amdahl 5840s, 4 VAX-11/780s and 25 IRIS 3500 workstations.) Users work on and off site, using a variety of networks not all of which are under NASA's control. Off site installations can be as close as a different building on the base or as distant as ICASE (on the East Coast.) In this environment, sharing accounts is very common; it provides a very quick and easy way to enable many people to work together, and allows many people to share responsibility for a particular task. For example, the account *nasops* is used to maintain a database of users (among other functions.) So, many people need access to that account. Unfortunately, this poses some problems.

†UNIX is a Trademark of Bell Laboratories.

First is the question of accountability. If someone logs in on the account *nasops* and compromises the database, how can the offending user be traced? Password management is the second problem; how can the site administrator force 40 or so people to keep the password secret, particularly since these users need at least two passwords (one for their own account and one for the *nasops* account)? When someone changes *nasops*' password, how does he or she communicate that change to the other users in a timely manner?

A "group account" is an account meant to be shared. No-one can log into a group account, but a program called *lsu* overlays the login identity with the group identity (just as the UNIX program *su* overlays the login identity with a new user's identity.) The user must type his own password when switching to the group identity. *Lsu* checks an access file to ensure that the user can access the group account at the given time and from the given terminal, and then checks the password. If access is allowed and the user types his own password correctly, the group identity is pushed over the user's identity; if access is denied or the password is incorrect, *lsu* simply informs the user permission is denied.

Because of the sensitive nature of the program, several steps are taken to prevent compromise. While it seems redundant to require the user to type his password (didn't he type it to log in?), experience shows that people do leave their terminals unattended, so checking the password provides some assurance the user is the person running *lsu*. The password must always be typed, even when *lsu* has already determined the used has no right to *lsu* to the group account. The location of the access and log files are constructed in memory when *lsu* runs, and are erased immediately after being used. Both files must be owned by the most privileged account and must be unreadable and unwritable by anyone else; if not, the *lsu* fails and mail is sent to the *lsu* administrators. Of course, when access is denied, the user is not told why access is denied.

Another version of this program provides the same control for overlaying user accounts. The program *nsu* functions like *lsu* but requires the new account's password as well as meeting any conditions in the access file. (If the account is not listed in the access file, anyone can *nsu* to it.) In the event the access file is corrupt, *nsu* can only be used to access an account from which the access file can be fixed.

Lsu and *nsu* use the startup file of the new (group or user) identity, not that of the user running the program; this provides uniformity among users who may have wildly different environments in their private accounts. A third program, called *su*, uses the startup file of the user running the program, but uses the *nsu* access file to check permission. This provides compatibility with older versions of *su*.

Detailed Statement of the Problem

One of the nastiest problems in account management arises when a number of people must share a single account. This is actually quite common; for example, at a TOPS-20 site, several users must have access to the *wheel* account to manage the system, or at a UNIX site, there are a number of such administrative accounts that more than one user accesses on a daily basis. This poses a number of management problems.

The most obvious one is accountability. Who is using the account? If the account has problems such as being over disk quota, or being used in an insecure manner, it can be very difficult to track the individual responsible for the problems. Worse, if the account should be used at an unusual time, or to do something the users of the account do not normally do, the system administrator will have to contact each user of the account in order to determine whether or not another party has learned the password and thus been able to penetrate the system.

The second problem is the computer system's version of a key management problem. Suppose 40 people have the password to the account *source*, which is used to edit and compile system programs. As any system administrator knows, this means "at least" 40 people, because experience dictates they will share the source account password with their colleagues who want it, whether or not those colleagues are authorized to get the password. So, to prevent this, the system administrator decides to change the password every so often. Now 40 people must be notified of the new password; their work will be delayed while they are told of the new password; and undoubtedly some of them will tell their colleagues about the new password. All this gains is pain for the users as well as the administrators.

One way around these administrative problems is to create *group accounts.* Access control lists† determine whether or not a specific user can gain access to such an account. However, the accounts are not accessible by just logging in; so,

† See [DENN82], section 4.4, for a discussion of access control lists.

the user must log in as himself, and then run another program to "push" the group identity on top of his own.

This paper discusses the implementation of such a scheme under UNIX. The next section describes the environment in which the solution was developed; the section after that describes the use of the relevant program, and the section after that discusses the access control list used by the program. The final section discusses some remaining problems and offers guidance on future work.

The UNIX Environment

UNIX provides a mechanism, called *setuid*, that allows one user to execute a program with the privileges of another user. The program may be a command interpreter called a *shell* (see [BOUR84] and [JOY84]), in which case any commands issued from within that command interpreter run with the privileges of the second user. This will be the basis for the notion of "pushing" the group identity. Pushing corresponds to executing such a command interpreter; popping correspondingly means exiting the command interpreter and thereby returning to the original environment.

UNIX does not ever store cleartext passwords; rather, the passwords are encrypted using a variant of the DES, and the thirteen character result is saved in a file called the *password* file. (See [MORR79] for a detailed discussion of how and why this is done.) So, a program need not be privileged to compare a password to that of a specific user; it simply encrypts the typed password, and compares the result to the one in the password file. (This provides an elegant mechanism for preventing someone from logging into a group account; simply use a string not in the range of the encryption function as the encrypted password, and no one will be able to log in to that account. This is useful for system maintenance accounts, such as the line printer spooler's account.) In addition, the password file stores other information about each user; this information is used to set up the environment for the user.

We discussed command interpreters called shells earlier. A UNIX shell is actually more than a command interpreter; users can define variables within the shell command language, and some of these variables are special. For example, the shell variable **USER** contains the name of the user executing the shell; the variable **HOME** contains the directory in which the user began his session; the variable **SHELL** contains the name of the command interpreter; and the variable **PATH** contains a list of the places to search for each command to be executed by the shell (and the value of this variable is often referred to as the "search path"). The last variable is vital, since if two programs have the same name, the shell will decide which one to execute based on the value of the **PATH** variable. Because of their effect on the way the shell works, these shell variables are more commonly called *environment variables.*

A shell is invoked by passing the shell's name, followed by a list of arguments and preset environment variables. Shells sometimes execute the commands in a command file (called a *startup file*) when they begin executing. There are two kinds of startup files, those executed when the shell is started, and those executed when the user logs in. The shell decides if it was started when the user logged in by checking the name with which it was invoked; if that argument begins with a hyphen ("–") the shell is a login shell.

This is a rough description of points of UNIX that must be taken into consideration in the design of the group account access program. The next section describes how that program works.

The Program

The program used to push a new user identity onto the current one exists in two forms. The first, *lsu*, is a very restrictive program that is used to access group accounts. The second, *nsu*, may be used to access any account with a password.† The way these commands are used is basically the same; we shall use *lsu* for demonstration, pointing out the difference between the two when relevant.

To push the identity of the user *grpact* (for "group account"), use the command

```
lsu grpact
```

Lsu will prompt the user for his password. At this point, the user types a password. If the program is *nsu*, the password is that of *grpact*; if the program is *lsu*, the password is that of the user's account and **not** that of the account *grpact* (which may not even have a password.) After verifying the password is correct, and after checking that the user is allowed to push the identity of account *grpact* over his, *lsu* will respond by providing a shell with the privileges of the account *grpact*. To return to his original environment, the user can pop the *grpact* environment by exiting this shell. Should the password be incorrect or

†There are two other versions of this program: *su*, which exists for compatibility reasons (it uses all protection mechanisms of *nsu* but duplicates the environment of the UNIX command *su*, hence the name) and *csu*, which checks the syntax and format of the access file.

permission not be given, *lsu* responds with

```
lsu: permission denied
```

and does nothing.

If the user wants to execute only one command using the group account, he may give additional arguments to the command; all after the group account name are passed as command arguments to the shell. For example, suppose there is a group account "source" that is used to recompile system sources, and a user with access to that account wants to copy the file "aux.c" to "aux.c.old". He can do this in one of two ways (in the following, what the computer types is in **boldface** and what the user types is in normal type):

```
$ lsu source
Password: abx34yz
% cp aux.c aux.c.old
% exit
$
```

or

```
$ lsu source -c "cp aux.c aux.c.old"
Password: abx34yz
%
```

The "-c" and double quotes around the copy command are necessary, because the "source" account uses the C shell, and that requires commands which are given on the command line to be surrounded by quotes. The actual command *lsu* will execute is

```
/bin/csh -c "cp aux.c aux.c.old"
```

(refer to *csh*(1) in [UPM84] for more information.)

Normally, when a new environment is pushed onto an existing one, certain parts of the old environment are not changed. These attributes are the ones associated with logging in; remember, some environment variables are set when you log in and others whenever the shell is invoked. Sometimes a user switching to the identity of a group account will want the shell to set the environment variables associated with logging in as well as those associated with the invocation of the shell. To have *lsu* do this, give the first argument as "–". So, in the first example above, if you wanted to set the environment as though you had logged in as "source", say

```
lsu – source
```

rather than

```
lsu source
```

In the subshell started by *lsu*, some variables in the environment are changed. **USER** is set to the login name of the new identity, **SHELL** is set to the pathname of the shell being used, and **HOME** is set to the home directory of the new account. If the new account is that of the most privileged user, *root* (often called the *superuser*), the value of the environment variable **PATH** is changed to exclude all but a small number of specific system directories. This is done because many systems have directories containing programs not necessarily written and installed by system programmer; allowing *root* to execute programs in them would create interesting possibilities for Trojan horse attacks.

The use of these commands is controlled by an access file set up by the superuser *root*. Basically, this access file controls who can push what identity, when, and from where. If *lsu* is being used, the user must have permission explicitly granted by this file. If *nsu* is being used, any new account listed in the access file becomes restricted in the sense that no user can *nsu* to such an account unless the access file explicitly allows him to. However, anyone knowing the proper password can *nsu* to any account that is not restricted in this sense. In the next section, we will discuss the format of this access file.

Access File

The access file is set up by the superuser and controls who may push new identities. It is readable and writable by only the superuser. Each entry is one line, and consists of four fields separated by one or more tab characters (ASCII HT, octal 011, hex 0x09):

user *identities-he-can-push* *terminals* *time*

The first field names the user attempting the push. The second is a comma-separated list of user names the identities of which the user named in the first field may push. The third field lists constraints on the devices the user may *lsu* or *nsu* from; the last field constrains the times during which the user may push a new identity. Some sample entries are:

```
nancy   bin     !"ttyp.*"   weekdays 9am-5pm
mab     Any     console     March 1985
rlb     staff   Any         Any
gba     news    >=9600      June 21 - Sept 21
```

An entry is said to be *satisfied* if that entry allows the current user to push the desired identity from the current terminal and at the current time; that is, if the entry alone would allow access for the user to substitute the new identity (assuming the correct password were supplied.) A user may be entered in the file more than once. The file is searched in order; if any entry for that user/identity pair is satisfied, the access permission is granted. Note that if there are two enbtries for the same user/identity pair, access permission is granted if either entry is satisfied.

The ***terminals*** field may constrain the device from which *lsu* or *nsu* is run by name or by terminal speed. To constrain by name, simply type the name of the terminal; for example, if the field were

/dev/console

then the user could substitute any of the identities on that line only if he executed *lsu* or *nsu* from the system console. (If the file name is not a full path name, the prefix "/dev/" is assumed.) If the program should only be run from the console and some other terminal, say "ttyi9", the above line could read

console,ttyi9

Sometimes it is more convenient to specify terminal names by patterns, as (for example) to prevent use of *lsu* and *nsu* from pseudo-ttys, the names of which begin with "ttyp" or "ttyq", one could give the line

!"ttyp.*","ttyq.*"

where the "!" means "not" and the double quotes mean that the string they surround is to be used as a pattern. The pattern matcher used is the same as for the editor *ed*(1) (see [UPM84]) so the precise syntax differs between Berkeley UNIX and System V UNIX. Finally, there are times when systems need to constrain users based on terminal speed. For example, a site somewhat conscious of security might not want users to overlay certain identities from terminals not located within the building; for these sites, giving a relationship followed by a baud rate will allow the user to push an identity only if the speed of the terminal satisfies the relationship with the given baud rate. Thus, if the third field were

```
>=9600
```

the user would not be permitted to push any identities on that line unless he were working on a terminal which ran at at least 9600 baud. Allowed relationships are "@" and "=" (both meaning equal), "<" (less than), "<=" (less than or equal to), ">" (greater than), ">=" (greater than or equal to), and "!=", "<>", and "><" (all meaning not equal to.) Note that if permission is conditioned on speed and the program is run in the background or with the standard input, output, and error all associated with files, permission will always be denied. A complete syntax is given in Appendix I.

As examples, go back to the sample access control file at the beginning of this section. The first line says that the user "nancy" can *lsu* to the group account *bin* only when she does so from a terminal the name of which does not begin with "ttyp" (on many implementations of UNIX, this prevents her from doing so when she is logged in over a network.) The next line says that user "mab" can *lsu* to any other account on the system from the console, but from no other terminal. The third line means that the user "rlb" can *lsu* to the group account *staff* from any terminal, and the last line says that user "gba" can push the identity of the group account *news* from any terminal running at 9600 baud or greater; in practise, this means that he cannot do so when logged in over a telephone line.

The *time* field constrains when the user can push the new identity. Times may be constrained by days of the week, times of day, or days of the year. For example, if a contractor is doing kernel work, he might be supposed to access the appropriate group account only during working hours. Times are expressed as a day of the year, followed by a day of the week, followed by a time of day; any of these three parts may be omitted, but if more than one is present, all must be true for *lsu* or *nsu* to allow the pushing.

The format for each of these is quite liberal; for the day of the year, any of the following forms will be recognized:

```
July 4, 1986
July 4
July, 1986
July
7/4/86
7/4
```

The first and fifth give permission only for July 4, 1986; the second and sixth, for July 4 of any year; the third, for any day within July, 1986; and the fourth, for any day in July of any year. Days of the week may be expressed by naming the day; only as much to identify one day is needed (for example, any of "Sunday", "Sun", and "Su" will be read as "Sunday"; but "S" will be rejected as ambiguous, since it could refer to "Saturday".) Times of day may be given in any of these forms:

```
8:12:16 PM
8:12:16
8:12 PM
8:12
8 PM
8
noon
midnight
```

If an "AM" or "PM" is given, time is based on a 12-hour clock. Otherwise, it is based on a 24-hour clock. So, the first time would give permission at 8:12:16 in the evening, the second at 8:12:16 in the morning, and so forth. Note that saying "8" gives permission for the entire hour of 8 o'clock in the morning; thus, 8 and 8:00:00 are **not** the same.

Times may also be expressed as an interval. For example, to prevent someone from running *lsu* on weekends, the time field could be set to "Mon-Fri". To restrict use of *lsu* to working hours during the summer months, the field "June-Sept Mon-Fri 8am-5pm" suffices.

A complete syntax is given in Appendix II.

Refer back to the sample file at the beginning of this section. The first line means user "nancy" can *lsu* to the group account *bin* only during the weekdays from 9 AM to 5 PM; the second line says that user "mab" can *lsu* to any account on the system during March 1985. The third line indicates that user "rlb" can substitute the identity of the group account *staff* at any time, and the final line allows "gba" to *lsu* to *news* only during summers.

Fields are separated by one or more tab characters. If either of the last two fields are omitted, they are held to be satisfied. It is recommended this not be done (and this feature may disappear in the next version.) Note the terminal field must be present if the time field is present.

The program *csu* will check for, and report, syntax errors in the access file. Since it will print the syntactically incorrect lines, it can only be run by the superuser.

Lsu and *nsu* use the permission file a bit differently. With *lsu*, if a user is not explicitly listed in the file as being able to *lsu* to the new identity, permission is denied; and the default new identity is the first name in the *identities-he-can-push* field of the permission file that is satisfied. With *nsu*, if a user attempts to *nsu* to some other account, and provides a proper password, the *nsu* succeeds unless the account being pushed is named in **any** *identities-he-can-push* field of the access file. If so, the access file must allow the user running *nsu* to do the pushing. The default new identity for *nsu* is *root.*

Log File

Every time *lsu* or *nsu* is executed, an entry is made in a log file indicating the success or failure of the attempt. This log is readable and writable only by the superuser. Some sample entries:

```
SU 02/18 11:20 + ttyi9 mab-spool   |09941| - became spool (UID 2, GID 1)
SU 02/18 11:21 i ttyi9 spool-      |09942| - bad tty (line 11)
SU 02/18 11:21 i ttyi9 spool-root  |09942| - invalid password
SU 02/18 11:21 - ttyi9 spool-root  |09942| - permission denied
SU 02/18 11:21 i ttyi9 mab-nancy   |09944| - nancy not newuser (line 9)
SU 02/18 11:21 - ttyi9 mab-nancy   |09944| - permission denied
nu 02/18 11:26 + ttyi9 mab-root    |09980| - became root (UID 0, GID 0)
nu 02/18 11:42 i ttyi9 mab-root    |10088| - invalid password
nu 02/18 11:42 - ttyi9 mab-root    |10088| - permission denied
su 02/18 11:55 i ttyi9 mab-        |10236| - root not newuser (line 1)
su 02/18 11:55 i ttyi9 mab-        |10236| - bad tty (line 3)
su 02/18 11:55 - ttyi9 mab-root    |10236| - permission denied
su 02/18 11:56 i ttyi9 mab-        |10243| - root not newuser (line 1)
su 02/18 11:56 + ttyi9 mab-root    |10243| - became root (UID 0, GID 0)
cu 02/18 11:59 v ttyi9 root-       |10270| - checking syntax of XXX
cu 02/18 12:04 v ttyi9 mab-        |10317| - permission denied (not root)
```

Each line has the same format. Fields are separated by blanks, except that the last field (everything after the rightmost hyphen, in the display above) may contain blanks. The first item is a two-character code for the version of *lsu* being run; they are

code	*program*
su	su
SU	lsu
nu	nsu
cu	csu

The next two fields give the month, day, and time; they are always in the form of a two-digit number for the month followed by a slash followed by a two-digit day of the month, and a two-digit hour followed by a colon followed by a two-digit minute. The month is represented as a number from 1 (January) to 12 (December), and the day of the month follows the slash. Times are given using a 24-hour (European) clock. The fourth field is one of "+", "−", "v", or "i"; the first indicates the substitution was successful, the second that it failed, the third that a syntax check was attempted, and the fourth an informational message. (Note that one run of any of these programs may produce many informational messages, but only one message indicating success or failure.) The fifth field is the name of the terminal from which *lsu* or *nsu* were run; it is at least seven characters long (if the terminal name is longer, this field is extended appropriately). The sixth field is seventeen characters or less; it consists of the user's name followed by a hyphen followed by the name of the new identity. If there is a problem with the access file, the name of the new identity may be omitted. If for some reason no login name can be found, the user identification number is printed in parentheses. The seventh field is a stamp which can be used to associate messages from the same run with one another; it is always five digits enclosed in square brackets and separated from the sixth and eighth field by one blank. Following this field, there is a hyphen, followed by a message field.

The **message** field indicates what happened and why. If the attempted substitution was successful, the message gives the login name, user identification number, and group identification number of the new identity. If the attempted substitution failed due to a system error, such as the access file having incorrect permissions or not existing, or because the user did not know the correct password, the message indicates this. If the substitution failed because the user did not have permission to make the desired access, many messages may appear, each indicating one reason for denial of access; but all have the same stamp. Note that under certain conditions error messages may be given but the access will succeed; this mainly happens when a user listed in the *nsu* permission file tries to *nsu* to an account not listed in any newuser's field of the *nsu* permission file.

Security Considerations

Lsu and *nsu* are very sensitive programs; security was a major consideration while writing them. We used a number of techniques to increase the difficulty of "breaking" the programs.

The first weak point is the access file. Obviously, it must be owned by *root* and not writable by any other user. Less obviously, it should not even be readable by any other user (if it were, an attacker would know which users can gain access to privileged accounts, and thus that he or she should concentrate their efforts on obtaining those user's password.) So, one step within the program verifies that these conditions hold. If not, the program terminates, the access file is made unreadable, mail is sent to an administrator, and the *lsu* fails. (Making that file unreadable prevents future *lsu*'s or *nsu*'s from using it but allows a superuser to look at it later.)

There is one problem with this scheme. If the identity to be pushed is that of the superuser, and the access file is corrupt or does not exist, it would not be possible for anyone to become superuser, thereby forcing the system to be reloaded (or some such drastic action.) To eliminate this problem, when the access file is corrupt or nonexistant and the identity to be pushed has UID 0, *nsu* will ignore the access file. This way, the user need only supply the superuser password. Of course, the first thing that user should do is fix the access file problem!

In fact, the name of the access file itself is hidden, on the principle that not advertising the location of the file will discourage attackers from using other means of breaching security to alter the file. This is done by constructing the file name in memory, and clearing the file name immediately after use. Thus, the file name exists during only two systems calls: the first, to get status information about the file; and the second, to open the file.

Lsu and *nsu* also conceal the reasons for denial of access from the user. When a user runs either of these programs, and for some reason access is to be denied, the only message the user will get is:

program name: permission denied

regardless of the reason for denial. This discourages attempts to figure out what a password is, or when the action is allowed, by trial and error. Of course, the specific reason for denial is placed in the log file; this log, however, is as stringently controlled as is the access file, and if it is not readable and writable by root and no-one else, *lsu* and *nsu* will refuse to push any new identity. Moreover, the user must at all times supply a password, whether or not the password is relevant to the denial of access. For example, if user "mab" tried to *lsu* to the superuser, and he did not have permission in the access file, there is no need to request a password; even if "mab" knows the password, access will be denied. But "mab" will still have to supply the password; this is to prevent him from knowing that he was denied access due to the access file. Otherwise, if an attacker were impersonating "mab", the attacker would know to try another account because he would not be able to use *lsu* or *nsu* to access the superuser account.

Normally UNIX programs may be invoked by more than one name using a mechanism called *linking*. *Lsu* requires that the program be invoked with a known name, because the name is used to determine how to verify the access rights of the user. If *lsu* is invoked by an unknown name, it denies access.

Normally the value of the user's **PATH** environment variable is not changed (unless, of course, he instructs *lsu* or *nsu* to read a startup file which does change that.) However, if the new identity is that of the superuser, a new default value is supplied. This is done because some directories are writable by users other than the superuser; a prime example is the current working directory, known as ".", which is usually in nonprivileged user's values of **PATH** but should **never** be in that variable's value for the superuser. The danger is it makes accidentally invoking Trojan horses very likely.

Backward Compatibility

The programs *lsu* and *nsu* are based on an older program called *su*. However, *lsu* and *nsu* differ in several important respects.

First, under most UNIX systems, *su* does not do any access list checking; if the user knows the password of the user whose identity he wishes to push, the push is allowed. This prevents the use of group accounts as described in the first section, because if the group account has a password someone can log into that account. The one system which provides an access list (4.3 BSD UNIX) does so only for the superuser and requires that the user be a member of the group *wheel*. Unfortunately, the file defining the members of that group is world-readable, and so an attacker knows which accounts he should try to penetrate in order to use *su* to get superuser privileges. Worse, no constraints on time or terminal are possible, and any account for which the user knows (or can guess) the password is vulnerable.

The version of *su* provided with *lsu* and *nsu* provides the security of those two programs but the environment of the original *su*. It uses only the access file, not the group file, to determine if a user can access the superuser, unless the access file is corrupt or nonexistant; in this case it acts just like *nsu*. This is very confusing for people who are used to using the original *su* and do not know it has been replaced!

Functionally, *su* is similar to *nsu* in that it requires the password of the account the identity of which is being pushed; however, the environment variable **USER** retains its original value, and on System V-based UNIXes, **HOME** also retains its original value. This causes certain programs such as mail-reading programs to work with the original user rather than the new identity that was substituted.

Further Directions

There are two areas where security needs to be tightened; unfortunately, these are under kernel control. The first is that some UNIX systems have a bug in the paging algorithm that does not check for text pages which have been written on; in this case, it is possible to alter one page of a setuid program to execute a shell, and then rerun the setuid program. The effect is to provide a setuid shell without needing a password. The only fix is to fix the kernel.

The second is that the access file is not encrypted. Were it encrypted, concealing its location would not be so critical, and in fact this is planned for the next version of these programs.

Currently these programs run on many different systems (see **Appendix III** for a list.) *Lsu* has been used for several months for administrative accounts and its users are quite pleased with the result. Not only do they not have to remember a second password, but *lsu* provides a consistent environment for all users of any particular account. This makes working together quite simple, and allows one procedure to be drawn up, rather than a different one for each user. All in all, it appears to be a success.

Acknowledgements: Thanks to Barry Leiner for his review of, and comments on, the first draft, and Bob Van Cleef for his on the second.

References

[BOUR84] Bourne, S., "An Introduction to the UNIX Shell", in *UNIX User's Manual, Supplementary Documents, 4.2 Berkeley Software Distribution, Virtual VAX†-11 Version*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA (March 1984), as reprinted by the USENIX association.

[DENN82] Denning, Dorothy, *Cryptography and Data Security*, Addison-Wesley Publishing Company, Reading, MA © 1984.

[JOY84] Joy, William, "An Introduction to the C Shell", in *UNIX User's Manual, Supplementary Documents, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version*, Computer Science Division, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA (March 1984), as reprinted by the USENIX association.

[MORR79] Morris, Robert and Thompson, Ken, "Password Security: A Case History", *CACM* **22**(11), pp. 594 - 597.

[UPM84] –, *UNIX Programmer's Manual Reference Guide, 4.2 Berkeley Software Distribution, Virtual VAX-11 Version*, Computer Science Division, Department of Electrical Engineering snd Computer Science, University of California, Berkeley, CA (March 1984), as reprinted by the USENIX association.

Appendix I. Syntax of the Terminal Field in the Access File

This appendix describes the semantics of the terminal field in the access file. In the grammar below, words in CAPITAL ROMAN TYPE are terminals and are defined after the grammar; words in *small italics* are nonterminals and are defined within the grammar itself.

† VAX is a Trademark of Digital Equipment Corporation.

```
stat     ::= expr
expr     ::= '(' expr ')'
           | '!' expr
           | expr '&' expr
           | expr '|' expr
           | ttyname
           | ttyspeed
           | 'any'
           | 'none'
ttyname  ::= name
           | '+' name
           | '-' name
           | '*' name
           | pattern
           | '+' pattern
           | '-' pattern
           | '*' pattern
ttyspeed ::= rate
           | '+' rate
           | '-' rate
           | '*' rate
rate     ::= RELATION NUMBER
name     ::= STRING
pattern  ::= '"' STRING '"'
```

where *RELATION* is any of:

@	(equal to)	<>	(not equal to)	>	(greater than)
=	(equal to)	><	(not equal to)	>=	(greater than or equal to)
<	(less than)	!=	(not equal to)	<=	(less than or equal to)

and *NUMBER* is any non-negative integer and *STRING* is any list of characters.

In a string, the following characters must be preceded by a "\" or they will terminate the string and produce unexpected results:

,	(comma)	=	(equal sign)	>	(greater than sign)
<	(less than sign)	!	(exclamation point)	(	(left parenthesis)
)	(right parenthesis)	|	(vertical bar)	&	(ampersand)
+	(plus sign)	-	(hyphen)	*	(asterisk)
		\	(backslash)		

The symbol '+' means the characteristic applies to the standard input device only, '-' means the characteristic applies to the standard output device only, and '*' means the characteristic applies to the standard error device only.

Appendix II. Syntax of the Time Field in the Access File

This appendix describes the semantics of the time field in the access file. In the grammar below, words in CAPITAL ROMAN TYPE are terminals and are defined after the grammar; words in *small italics* are nonterminals and are defined within the grammar itself.

```
stat          ::= expr
expr          ::= '(' expr ')'
                | '!' expr
                | expr '&' expr
                | expr '|' expr
                | day_of_year day_of_week time_of_day
                | day_of_week day_of_year time_of_day
                | day_of_year time_of_day
                | day_of_year day_of_week
                | day_of_week time_of_day
                | day_of_week day_of_year
                | day_of_year
                | day_of_week
                | time_of_day
                | 'any'
                | 'none'
day_of_week::= DAY_OF_WEEK
                | DAY_OF_WEEK '-' DAY_OF_WEEK
                | day_of_week ',' day_of_week
time_of_day ::= time_of_day '-' time_of_day
                | time_of_day ',' time_of_day
                | NUMBER ':' NUMBER ':' NUMBER MERIDIAN
                | NUMBER ':' NUMBER MERIDIAN
                | NUMBER MERIDIAN
                | NUMBER ':' NUMBER ':' NUMBER
                | NUMBER ':' NUMBER
                | NUMBER
                | SPECIAL_TIME
day_of_year ::= MONTH NUMBER ',' NUMBER
                | MONTH NUMBER
                | MONTH ',' NUMBER
                | MONTH
                | NUMBER '/' NUMBER '/' NUMBER
                | NUMBER '/' NUMBER
```

where *NUMBER* is a positive integer (if not valid, an error message is printed), *MERIDIAN* is either "am" or "pm", *SPECIAL_TIME* is either "noon" or "midnight", *DAY_OF_WEEK* is any of:

Sunday	Tuesday	Thursday	Saturday
Monday	Wednesday	Friday	weekdays

and *MONTH* is any of:

January	April	July	October
February	May	August	November
March	June	September	December

With any of these keywords, only enough of the name to identify a unique name must be given, and case is ignored. (Note that 'a' is not a unique abbreviation for "am", because it is also the first letter in "april" and "august".)

Appendix III. Compilation and Installation

When you get the *lsu* package, type

```
sh Setup
```

This program asks you to enter your system type as defined below:

enter	*if your system is*
BSD4_2	Berkeley Software Distribution Release 4.2
BSD4_3	Berkeley Software Distribution Release 4.3
DYNIX2_0	Sequent Balance Dynix Release 2.0
NPSN3	NAS Processing System Network Build 3
ROS3_3	Ridge Operating System Release 3.3
SGI2_3	Silicon Graphics IRIS Graphics Library 2 - Workstation 2.3
SGI3_4	Silicon Graphics IRIS Graphics Library 3 - Workstation 3.4
SGI3_5	Silicon Graphics IRIS Graphics Library 3 - Workstation 3.5
SUN4_2	Sun Microsystems UNIX Release 4.2
SYSV	AT&T System V
UNICOS	Cray UNIX Operating System
UTS	Amdahl UTS

(You can supply the type as a command-line argument, too.) This will link the command file to make the programs for the system to "Makefile". Once this is done, replace the lines

```
LSUPERM="./.lsu"
SUPERM="./.su"
LOG="./.log"
DESTDIR="/usr/local/bin/"
```

with the full path names of the access files for *lsu*, both *nsu* and *su*, the log file, and the directory where the executables are to be placed, respectively. Then type

```
make
```

to compile the programs and

```
make install
```

to install them.

If your system is not one of the ones named above, look in the directory "Make" for a file that will work, or build one using them as a model. The command files all are named "Make.*sys*", where *sys* is the system type entered to *Setup* (see above.) Do this for your system, too, and add the new abbreviation to *Setup*. Then look in the directory "Ed" – these files edit part of "lsu.H" to change the names of the log file and the permission files. The output will look something like this (the lines that begin with "buf[" will be different; on a System V-based version of UNIX, the *ed* commands will have some extra backslashes. This is due to the difference between BSD and System V pattern matchers.)

```
/^\/\* begin SUPERM \*\/\\$/,/^\/\* end SUPERM \*\/\\$/c
/* begin SUPERM */\
buf[0] = '.';\
buf[1] = '/';\
buf[2] = '.';\
buf[3] = 's';\
buf[4] = 'u';\
buf[5] = '\0';\
/* end SUPERM */\
/^\/\* begin LSUPERM \*\/\\$/,/^\/\* end LSUPERM \*\/\\$/c
/* begin LSUPERM */\
buf[0] = '.';\
buf[1] = '/';\
buf[2] = '.';\
buf[3] = 'l';\
buf[4] = 's';\
buf[5] = 'u';\
buf[6] = '\0';\
/* end LSUPERM */\
/^\/\* begin LOG \*\/\\$/,/^\/\* end LOG \*\/\\$/c
/* begin LOG */\
buf[0] = '.';\
buf[1] = '/';\
buf[2] = '.';\
buf[3] = 'l';\
buf[4] = 'o';\
buf[5] = 'g';\
buf[6] = '\0';\
/* end LOG */\
1,$w
q
```

This is used as input to *ed* to set up the names of the log file and the permission files without putting the strings into a form that can be read by looking for ASCII strings in the executable.) Figure out which one works. Finally, look in "sysdep.h" and add the appropriate definitions.

Messages indicating log and access file problems are sent to the user "lsumaint". Either create a mail alias by this name to reroute the messages to the appropriate people, or change the value of the constant **LSUMAINT** in "sysdep.h".

Appendix IV. Manual Pages

The following pages are the manual pages for *lsu*, *nsu*, and *su*, version 3.0.

NAME

su, lsu, nsu – substitute user id temporarily

SYNOPSIS

lsu [–] [*userid* | ––] [*command_as_shell_argument*]

nsu [–] [*userid* | ––] [*command_as_shell_argument*]

su [–] [*userid* | ––] [*command_as_shell_argument*]

csu [*file* ...]

DESCRIPTION

Lsu, *nsu*, and *su* allow a user to become another user without logging off. These programs, collectively called "su programs", will execute a shell with real and effective UIDs and GIDs set to those of the specified user. The new shell will be the program named in the shell field of the specified user's password file entry, or *sh*(1) if none. The new user ID stays in force until the shell exits.

Any additional arguments given on the command line are passed to the program invoked as the shell. Notice that with *sh* and *csh (1),* this means commands must be preceded by the option **–c** and must be quoted (see the respective manual pages or the examples below.) In this case, the default user ID may be used by replacing *userid* with ––. This must be done if *userid* is to be defaulted, since otherwise *lsu* will think the first word of the command is the login name of a user.

The user's environment variables are changed as follows: *lsu* and *nsu* set **USER** to the login name corresponding to *userid* (the two are usually the same); all su programs set **HOME** to the home directory of *userid*; **SHELL**, to the full path name of the shell being executed; and if the *userid* has a UID of 0, **PATH** will be set to a specific set of directories. (If there is no **PATH** environment variable and *userid* has a UID of 0, a **PATH** environment variable will be added.) However, argument 0 of the shell being executed is set to the name of the su program being executed except when the first argument is –, in which case the environment is changed further to what would be expected if the user had logged in as *userid*. This is done by invoking the shell with the first character of argument 0 being '–' (that is, as *-lsu*, *–nsu*, or *–su*); this convention causes shells to read their startup files.

To use *nsu* or *su*, the user must know the password of the *userid* to which he wishes to change. The system administrator may allow only certain users to use these programs to change to a given *userid*; in such a case no other user may use these programs to change to that *userid* whether or not he knows *userid*'s password. In these cases the system administrator may also limit the devices from which these two programs may be run and the times during which it may be run. With all users, the default *userid* is *root*.

To use *lsu*, the user need not know the password of the *userid* to which he wishes to *lsu*. However, the system administrator must explicitly grant the user permission to use this program. The system administrator may constrain the use of this program by limiting the devices from which it may be run, the times during which it may be run, and/or the *userid*s that may be substituted. The system administrator also controls the *userid* that will be assumed should no *userid* be supplied on the command line.

If the current user has a UID of 0, access control files are not checked and no passwords need be supplied.

Csu takes as arguments one or more access files, and checks the syntax of the entries in these files. If no arguments are supplied, *csu* reads from its standard input. *Csu* may only be run by the superuser.

EXAMPLES

To become user **spool** while retaining your nonlocal environment, type

```
lsu spool
```

To become user **spool** but change the environment to that which **spool** would have had it logged in, type

```
lsu – spool
```

To execute *command* with the login environment and permissions of user **spool**, type

```
lsu – spool –c "command"
```

Note the arguments following **spool** are passed directly to **spool**'s shell. To execute *command* as the default user ID, type

```
lsu –– –c "command"
```

In all cases, *su* or *nsu* may be used rather than *lsu*.

WARNINGS

The shell supplied by *lsu* and *nsu* runs the startup files of the user you are *lsu*'ing or *nsu*'ing to. This is in the spirit of what these programs should do. If you really want the old behavior, use *su*; it runs the startup files of the user you are *su*'ing to, and is provided for backwards compatibility with an older version of *su*(1).

The default search path varies from system to system. If **PATH** is defined, the directories which will be in the search path when a user *lsu*s, *nsu*s or *su*s to superuser also vary from machine to machine. In an ideal world, the resulting search paths would always be the same, regardless of the machine on which this program were run, but in our environment, this is not likely soon.

FILES

/etc/passwd	password file	various access files	log file

SEE ALSO

csh(1), login(1), sh(1)

AUTHOR

Matt Bishop (*mab@riacs.edu*)

VERSION

This describes version 3.0 of *lsu*, *nsu*, and *su* .