Tightening System Security Under UNICOS

Nicholas P. Cardo, Sterling Software, Inc., Numerical Aerodynamic Simulation Facility, NASA Ames Research Center, M/S 258-6, Moffett Field, CA 94035-1000 USA

ABSTRACT: *Protecting a system from threats is an ever increasing and time consuming function. However, there are several changes that can be made to a system which will improve its security. This paper discusses two changes that can be made to a system for improving system security. The first change enhances the password selection criteria. The second change improves the authentication and audit trail capabilities of the* `su` *command. These two improvements, along with their rationale, are discussed.*

Security threats can come in many forms and from many sources. System administrators must take the necessary precautionary steps to protect the system and its users from attacks.

Although evaluating a system's security and securing it from all threats is a major undertaking, there are some simple changes that can be made to improve a system's security. Two such changes are improving the password selection criteria and improving the authentication and audit trail capabilities of the `su` command.

Using existing security features that are provided with UNICOS can improve system security. There are also several easily obtained security tools that can be used to enhance security auditing.

1 Improving Password Selection Criteria

The topic of passwords falls under several security concerns ranging from clear text passwords on the network to no passwords on accounts. A comon problem with passwords is that in many cases, comon dictionary words are used.

The password selection criteria determines the complexity of the password. The more complex the criteria, the more difficult it becomes to crack the password. Therefore, increasing the complexity of password selection criteria, improves the quality of the password.

For example, suppose the password to be used is the word "`password`". Without any rules, this is a simple dictionary word which can be cracked in very little time. By introducing some complexity into the criteria, the password becomes tougher to crack. The following are variations of the word "`password`" listed in order from easiest, to most difficult to crack:

```
password
passw0rd
Passw0rd
Pas$w0rd
```

By altering a few characters, the ability to crack the password becomes more difficult. Changing the spelling of words can also improve the quality of the password. For example, compress double letters to single letters to get "`Pa$w0rd`". By compressing double letters, the word chosen is no longer a dictionary word with some simple substitutions. Another technique is to drop vowels from passwords to get a password that is not a direct match for a dictionary word in the form "`Pa$wrd`." The password could be increased to 8 characters to get "`#Pa$wrd1`." The end result is a password that is based on a dictionary word, but is sufficiently altered to make cracking difficult. By adding the leading `#` and the trailing `1`, the password no longer matches a dictionary word.

Small changes to the algorithm can increase the number of combinations drastically. The number of combinations[1] of r objects selected from a set of n distinct objects is:

$$\binom{n}{r} = \frac{n!}{r!(n-r)!}$$

Using 26 lowercase letters and requiring only 6-character passwords would yield:

$$\binom{26}{6} = \frac{26!}{6!(26-6)!} = 230230$$

A password length of 8 characters shows:

$$\binom{26}{8} = \frac{26!}{8!(26-8)!} = 1562275$$

This results in an increase of 1,332,045 possible combinations. Increasing the number of possible characters to choose from by including uppercase letters, numbers, and special characters will also increase the number of possible combinations.

To get a better understanding of the importance of complex passwords, a list of commonly used password hacking techniques follows.

- Use dictionary words
- Substitute numbers for letters. Common substitutions are:

  0 for o
  1 for i or l
  5 for s
  3 for e

- Substitute special characters for letters. Common substitutions are:

  $ for s
  ! for i or l

- Use proper names
- Compress double letters to a single letter

The best passwords contain numbers, special characters, uppercase letters, lowercase letters, and are unintelligible to the human eye. However, these are difficult to remember and typically introduce another security problem, writing down passwords where they may be viewed by unauthorized personnel. A solution to this is to base the password on multiple words. An example of this would be "3atAtJoe$", which is derived from "Eat At Joe's".

1.1 Existing Password Criteria

The need for enforceable guidelines for password selection is extremely important. The /bin/passwd command supplied with UNICOS contains a few selection rules. These rules[2] are:

- Each password must have at least six characters. Only the first eight characters are significant.
- Each password must contain at least two alphabetic characters and at least one numeric or special character. In this case, "alphabetic" means mixed-case letters.
- Each password must differ from the user's login name and any reverse or circular shift of that login name. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.
- The new password must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

1.2 Modified Password Criteria

The guidelines provided with /bin/passwd are sufficient for generating good passwords, but could be improved.

Only the first eight characters of a password are used for authentication. Increasing the minimum length of a password from six to eight will improve the quality of the password.

Another improvement is to force combinations of uppercase letters, lowercase letters, numbers and special characters to help insure that commonly spelled words are not used as passwords.

With these changes, the new password criteria becomes:

- Each password must have at least eight characters. Only the first eight characters are significant.
- Each password must contain characters from at least three of the following categories:
  - uppercase characters
  - lowercase characters
  - numbers
  - special characters
- Each password must differ from the user's login name and any reverse or circular shift of that login name. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.
- The new password must differ from the old by at least three characters. For comparison purposes, an uppercase letter and its corresponding lowercase letter are equivalent.

The main differences are that the minimum length of a password is increased to eight and the number of different types of characters required is increased from two to three.

1.3 Modifying passwd

Unfortunately, as of UNICOS 9.0, there are no user exits available in passwd. This means the only way to improve the password selection algorithm is to modify the source code for passwd.

1.3.1 Minimum Password Length

Although a password can exceed eight characters in length, only the first eight characters are used by UNIX systems. While increasing the minimum length of a password increases the number of possible combinations, the 8-character limit causes an upper bound limitation to be enforced. By setting the minimum length to eight, a password cracking program can be set to try only 8-character passwords. The password field in the udb, ue_passwd, provides space for a 15-character encrypted password. At current, UNICOS still only utilizes the first eight characters of any password for authentication.

Increasing the minimum length of a password is a very simple change. The define variable MINLENGTH controls the minimum length of a password. Simply change MINLENGTH from six to eight in passwd.c.

```
#define MINLENGTH 8
```

Random passwords are not formed from dictionary words, making them very difficult to crack. However, to set the minimum length for a random password, the variable minpass in the file digrams.c must be increased from seven to eight .

```
int minpass = 8;
```

The command ranpass uses the information in digrams.c for password generation.

1.3.2 Required Characters

This change is a little more difficult but the comments within the source code help identify the section to be updated. Each criteria is separated and identified within the source code.

Locate the desired section to update, in this case the routine that enforces two alphabetic characters and one numeric or special character. Again, each of the criteria is easily identified by their comments and the source code itself.

The scanning of the characters is replaced with:

```
if (isupper(c))
   flags |= 1;
else if (islower(c))
   flags |= 2;
else if (isdigit(c))
   flags |= 4;
else
   flags |= 8;
```

This causes uppercase, lowercase, numbers, and special characters to be tracked separately.

The testing of the `flags` variable needs to be updated to permit the following values:

TABLE 1. Valid flags values

Value	Upper 1	Lower 2	Number 3	Special 4
7	1	1	1	0
11	1	1	0	1
13	1	0	1	1
14	0	1	1	1
15	1	1	1	1

These values indicate that the password has matched three out of the four categories of characters.

To increase the complexity by requiring all four types of characters, the `flags` variable must equal 15 for a valid password.

1.4 Existing Feature

UNICOS supports a password aging feature to assist in password maintenance. This feature allows the administrator to define a maximum lifetime for any user's password. At the end of the password's lifetime, the user is required to change their password.

A second part of this feature sets a minimum lifetime for a password. This is useful for preventing a user from changing their password and then immediately changing it back.

2 Becoming Another User

The `su` command allows a user to become another user, including root. Authentication is performed on the basis that the user utilizing the `su` command is in possession of that user's password.

There are two basic shortcomings of the `su` command. The first is that there is very little information logged about the activity. An audit trail would have to be performed through the accounting data, but even then the true identity of who is becoming root is unknown. The second problem area is that there is no way to restrict activity for certain accounts to specific users.

2.1 Privileged Command Processors

One way to keep from distributing privileged passwords is to use an interface utility which can only execute selected commands. One such command is `priv`. This utility has a configuration file which identifies who can execute the commands identified within the configuration file. Also contained within the configuration file is the username to execute the command.

A sample entry in the configuration file for running the command `kill` would be:

```
kill /bin/kill root nick,james,paul
```

This identifies users `nick`, `james`, and `paul` to be authorized to run the command `kill` as user `root`. This is useful for killing runaway processes or killing processes belonging to other users. An example of the command is:

```
priv kill -9 12345
```

In this example, process id `12345` will be sent a signal `9` to terminate it. The uid of the process for `kill` will be `0` so that the `kill` command is executed as `root`.

Using the `priv` command causes all of its activities to be logged as well as all of the arguments specified.

```
Jan 15 07:15:01 localhost priv: uid=nick
(1234) cwd=/home/nick cmd=/etc/priv kill
-9 12345

Jan 15 07:15:01 localhost priv: running as
uid=0, execing to binary /bin/kill
```

These two log entries identify who ran the `kill` command as well as what options were specified for the `kill` command.

It is possible to safely set up a configuration file for `priv` so that almost all operation and administrative activities can be performed without becoming `root`.

Two other alternatives are `sudo` and the Operator SHell (`osh`).

2.2 LSU

One security philosophy is to use the user's own password for authentication for accessing other accounts, including `root`. The `lsu` command is based on the premise that if a user can authenticate themselves with their password, then they can become the desired logged in user. Also, `lsu` includes the functionality of restricting users who can become another user. The configuration file contains a list of users who are permitted to become the destination user. There is also some additional audit trail information provided with the command.

One problem with the `lsu` command is that it only requires the user to know their own password. This means that if an authorized user of `lsu` who is permitted to become `root` has their password cracked, this also gives access to `root`.

2.3 Improving `su`

Giving `su` the capabilities provided with `lsu`, while still requiring the destination user's password, would provide a significant improvement to the `su` command. Unfortunately, the user exit provided in the `su` command is insufficient for performing this functionality. Therefore, a source code modification is required to build this functionality into `su`.

2.3.1 Configuration File

The configuration file consists of lines of the following format:

```
name:name1,name2,name3,name4
```

where `name` is the destination user and `name1`, `name2`, `name3`, and `name4` are users who are permitted to become `name`. If the destination username does not exist in the configuration file, then the destination user is not limited to specific users. The reverse of this is also true, if the destination user exists in the configuration file but the invoking user is not listed as an authorized user, then access is denied, even if the destination password is known.

However, a special case needs to be addressed. Suppose an authorized user desires to become root from an unauthorized account. Authentication is performed by prompting the invoking user for their username and password. So it is now known who became root, even if not from their own account.

The UNICOS function `ia_user(3C)` provides a common interface to be used as part of identification and authentication. Part of this provides the capability of restricting root to only console logins. This is enforced provided `CONSOLE` is defined for the library function `ia_user()` which is part of the module `ia.c` in the `libc/gen` source code. Since root logins are only permitted on the console, which is presumably secured based on the location of the OWS, root is protected from network attacks. Therefore, the only way to become root is to use `su`. An entry in the configuration file for root with a list of authorized users protects unauthorized users from attempting to crack the root password through the `su` command.

A valid entry in the configuration file for root would look like:

```
root:nick,paul,frank
```

With the exception of logging in directly to root at the console, the only users permitted to become `root` are `nick`, `paul`, and `frank`.

A safety precaution is built-in to prevent `su` from no longer working in the event that the configuration file is missing. Under this condition, `su` will function as though the modification has not been made and allow continued usage. This feature is important because in the event that the filesystem becomes corrupt and the configuration file is lost, administrators will still have the ability to become root and reconstruct the lost configuration file.

2.3.2 Improving the Audit Trail

One change to `su` is to improve the audit trail of users becoming privileged users. The goal of improving the audit trail capabilities is to be able to always know who became a privileged user, when, and on what terminal.

Some audit trail capabilities with `su` can be obtained from `/usr/adm/sulog`.

```
SU 01/04 20:04 + ttyp013 nick-root
```

The problem with this type of message is that the user to become root may not be in possession of the `root` password. It is impossible to identify the real individual who is becoming `root`. Since this modification includes a challenge for the users password to unauthorized users, a new log message is attained.

```
Jan  5 08:08:24 localhost su: su to root by
nick from james on tty /dev/ttyp088
```

Here, the identity of the user becoming `root` is clearly listed.

There are two additional messages to indicate failures. The first type of message indicates that the authenticating user entered an invalid password while the second message indicates successful password entry for an unauthorized authenticating user.

```
Jan 16 14:31:00 localhost su: su to root by
nick from james on tty /dev/ttyp001 failed
```

This message indicates the user `james` on tty `/dev/ttyp001` attempted to become `root` by authenticating with the password for `nick`. The "`failed`" indicates that the password entered for the account `nick` was entered incorrectly.

```
Jan 16 14:32:06 localhost su: su to root by
nick from james on tty /dev/ttyp001 is not
allowed
```

This message indicates that all passwords were successfully entered but that user `nick` is not authorized to become `root`.

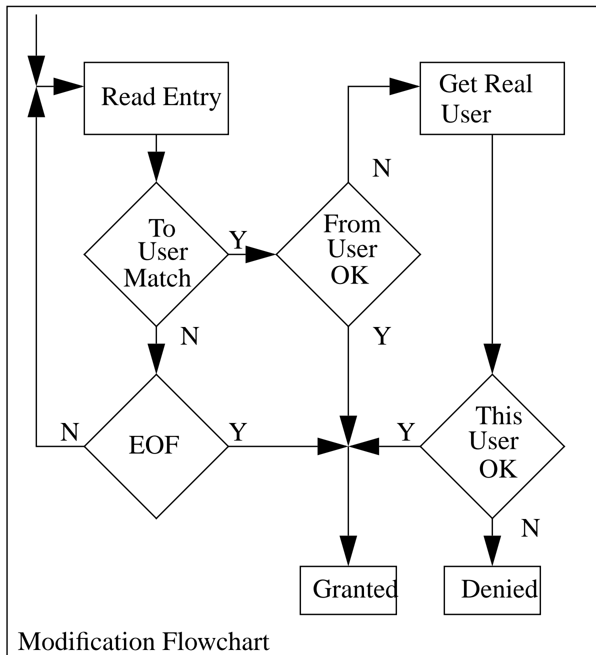
2.3.3 Adding the Functionality

The modification to su to implement this functionality is actually straightforward and simple. It consists of two components: reading the configuration file and authenticating the user.

The configuration file is processed with sequential read operations until either the destination user is found or the end of the file is reached. Further action is required only if the destination user is found.

At this point, three crucial pieces of information are known: the invoking user, the destination user, and the list of users from the configuration file. Authenticating the invoking user is a two-step process. The first step is to scan the list of users for the

invoking user's name. If there is a match, then access is granted and no further checking is necessary. However, if the user is not there, then the invoking user is prompted for their real username and password. If a proper username and password are given, then the list of authorized users is scanned for this username. If there is a match, then access is granted, otherwise access is denied.



Modification Flowchart

Caution should be exercised when processing passwords. The `getpass()` function should be utilized for prompting of passwords. This function provides a safe way of entering a password without displaying the characters entered. However, the password is now stored in a variable in plain text. The password should be immediately encrypted and then the plain text copy destroyed. Password encryption is performed with a one-way encryption algorithm. In order to validate a password, it must be encrypted and then compared with the encrypted password in the system udb.

The following code segment shows a safe way of processing a plain text password:

```
getsysudb();
ueptr = getudbnam(username);
pw = getpass("Password: ");
epw = crypt(pw,ueptr->ue_passwd);
memset(pw,0,strlen(pw));
if(!strcmp(epw,ueptr->ue_passwd))
   printf("Passwords match\n");
else
   printf("Passwords don't match\n");
```

It is important to note how the plain text password is handled. The password is obtained with `getpass()`, immediately encrypted and then the plain text password is destroyed. At this point comparisons for a correct password can be performed and appropriate actions taken.

2.3.4 *Example of Challenge*

The following is an example of an authorized user becoming root:

```
% su
Password:
#
```

For authorized users, there appears no change in the process of becoming root. This is also true for destination users not listed in the configuration file as being restricted. However, for unauthorized users, there is an additional step:

```
% su
Password:
Who are you in real life: nick
Password:
#
```

The invoking user is prompted for authentication. In this case, both the `root` password and the password for `nick` are required before access is granted.

3 Helpful Utilities

There are many utilities for performing audits and security checks that administrators can utilize. Some of these are:

`bincheck` A very sophisticated file scanning and auditing package which provides intensive reporting on changes to the file system from one snap-shot to the next. Also included is a checksum capability.

`deszip` A fast DES encryption package which includes a password validation program and a replacement for `/bin/passwd`.

`homecheck` A sanity checker for user home directories and environment files.

`noshell` A replacement shell to track access to disabled accounts. When an account is disabled, the shell field is changed to `noshell`. When someone accesses a disabled account (either login, rlogin, telnet, rsh, rcp or ftp), a message will be sent to a designated account. This message reports on the remote host from where the attempt was made, and if available, the remote user.

`raudit` A `.rhost` file auditor which reports on illegal entries, nonoperational entries, and the total number of `.rhost` entries for each user.

4 Summary

There are many ways for an administrator to improve the overall security of the system. There are also many books available to aid them with the task of setting up security procedures for a system.

This paper presents two minor modifications that can be made to additionally improve the security of a system. The first modification improves the password selection criteria for the construction of better passwords. This plays an important role in the protection of a system from external attacks. The second modification improves the identification and authentication mechanism of the `su` command. This modification keeps unauthorized users from becoming root and improves the audit trail of `su` activity.

Several security tools have also been identified to assist administrators. In addition, enabling an existing feature, known as password aging, has been discussed.

Adding security precautions to a system can be accomplished with little or no noticeable changes in system operation. There is an old saying, "An ounce of prevention is worth a pound of cure." Everyone benefits from stronger security procedures.

United States sites are welcome to visit the Numerical Aerodynamic Simulation (NAS) Facility on the World Wide Web at `http://www.nas.nasa.gov`. They can retrieve the security utilities mentioned in this paper by accessing the software archive from the NAS home page. The author can be reached at `cardo@nas.nasa.gov` for additional information.

5 References

1. *Mathematical Statistics*, John E. Freund and Ronald E. Walpole, Prentice-Hall, Englewood Cliffs, NJ, 1980.
2. *passwd(1) man page*, Cray Research Inc.