The Operator Shell: A Means of Privilege Distribution Under Unix

Michael Neuman Gary Christoph
<mcn@lanl.gov> <ggc@lanl.gov>
Computer Operations and Assurance
Computer Security Section
Los Alamos National Laboratory

ABSTRACT

The Operator Shell (Osh) is a setuid root, security enhanced, restricted shell for providing fine-grain distribution of system privileges for a wide range of usages and requirements. Osh offers a marked improvement over other Unix privilege distribution systems in its ability to specify access to both commands and files, auditing features, and familiar interface. This paper describes the design, features, security considerations, internals, and applications of the Operator Shell.

I. Introduction

In large computing environments, support personnel have more specific responsibilities than provided for by the Unix "all-or-nothing" type of privilege system. A simple solution is to give everyone who requires system privileges the root password. For some large systems at Los Alamos, over fifty people would require system privileges: Operators are responsible for shutting down the machines, syncing filesystems in emergencies, and watching status monitors; Consultants, in the course of answering user questions, typically need to view a user's directory and files without forcing him to change access permissions; Security Administrators are responsible for auditing, monitoring, and maintaining the password database; finally, a few System administrators need complete access to the system. With potentially up to fifty people holding the root password, there are glaring problems of accountability (who did what when), administration (root password distribution--how to notify fifty people quickly and easily, and worse, having to change the password whenever someone no longer requires privileges), and user restriction (e.g., ensuring consultants aren't building new kernels).

The Operator Shell (Osh) offers a solution to each of these problems. Osh is a setuid root, security enhanced, restricted shell. It allows the administrator to carefully limit the access of special commands and files to the users whose duties require their use, while at the same time automatically maintaining audit records. The configuration file for Osh contains an administrator defined access profile for each authorized user or group. This profile lists the commands which may be run and specific access rights for files and directories. In addition to this fine grain distribution of privilege, all typed commands are logged along with a notation of their success or failure, offering a comprehensive audit log.

To the user, the Operator Shell looks like a standard C shell. It supports pipes, wildcards, aliasing, redirection, and environment referencing, except there is no longer any concept of a path. The user is restricted to only the specific commands, and specific path to those commands, that the administrator chooses. To improve functionality, some otherwise dangerous commands (e.g. *more* and *vi*) have been rewritten to support the Osh model of security. Osh does not interfere with file accesses a user would normally be able to perform. Additionally, it allows the user access to files specified in the user's profile in the Osh configuration. For example, Osh can be configured to allow a user write access to a directory but to none of the files in it. Optionally, a file is also readable if the file owner places a key in his home directory, granting permission to a consultant, for example, to read his files. This gives a user specifically grantable control over his privacy.

II. Design of the Operator Shell

The Problem

The Operator Shell is designed to address the problem of special needs users. These users need certain system-level privileges, while being carefully limited to only the tasks they are required to perform. The needs of three types of users were considered in the design of the Operator Shell: consultants, operators, and security administrators.

The consulting group at LANL consists of seven cleared and uncleared lab employees. Hundreds of customers call in daily with questions about utilities, errors in their source code, and problems with logging in. As a result, consultants frequently need to look at the caller's files to determine the problem and offer solutions. Since they have no need for general system administration privileges, it was decided not to give them the root passwords, and force them to find a system administrator who could look through the files with them. This cost the consultants hours in tracking down a sysadmin who was willing to spend time watching a consultant use his terminal and phone. The time used on these types of problems was in the hours or days. They needed a way to look at any user's files, given that user's permission, without being able to read or write to anything they shouldn't. In addition, consultants are typically "power users" with little time, consequently the interface must be simple to learn and not force any unreasonable restrictions--they should be able to do everything under Osh that they could as a regular user.

The second type of special users are the operators. There are approximately twenty-five operators at LANL who are responsible for 24 hour computer support. They watch status monitors, reboot systems, sync disks, reset queues, etc. The operators perform all of their tasks using a procedure manual containing step by step instructions for each of the situations they're required to handle. Therefore, the interface to the Osh cannot change from the standard shell the manuals were written for, and the allowed command set and file permissions are well defined. The name and activities of the operators needed to be logged as well to provide an audit trail for system administrators to refer to if any problems occurred. With this log they can determine exactly what happened, when, and who responded. It was important to meet the needs of the operators since they comprised the majority of root password holders.

Finally, the needs of security administrators were considered in the design of Osh. Primarily, they need to edit password files and review security logs, with occasional review of user files (without the user's explicit permission) and in some cases, unrestricted access to the whole system. In addition to those user-level features, security administrators needed complete audit logs from Osh to detect abuse as well as the ability to carefully configure user and group access.

The driving force behind the Operator Shell is the difficulty of proper root password administration. Giving too many users the root password creates huge security risks, yet not giving away the root password to those users that need the privileges hurts productivity. Assuming a proper balance between security and productivity has been reached, other administration problems surface when one of the users holding the root password no longer requires it. Instead of changing and redistributing the password to all those who require it, an obviously difficult practice, the user retains root privileges until the next periodic root password change. The Operator Shell was designed to solve the problems of root password administration by addressing the special needs users who require limited system privileges.

Design Goals of Osh

Design goals for the Operator Shell were developed from the profiles of the special needs users as well as from a few general policy decisions. Osh is fundamentally designed to allow a drastic reduction in the number of holders of the root password without reducing productivity. Following are the goals which shaped the Operator Shell.

- Configuration should be easy
- Configuration should be flexible and specific enough to control all aspects of user access
- Security should be superior to standard root shells by offering thorough audit logs containing transaction time, command, and outcome
- Considering the nature of Osh, it should be extremely resistant to abuse and attack
- Resistance to abuse should not come at the cost of functionality
- Keep the code simple enough to be easily understood by administrators and system programers
- Interface should be well known to the user
- Support as many features of that interface as possible so Osh is virtually transparent
- Provide the user the same access he would have outside of the shell

III. Implementation

User Features

The interface to the Operator Shell looks like the standard Cshell with only a small number of interactive features unimplemented. Included are pipes, file redirection, aliasing, environment referencing, and wildcards. History, substitution, and job control may be added later. Integrated into the shell is a built-in help command which lists the defines the program was compiled with as well as available commands. Following is a sample session with the Operator Shell.

```
rho% uname -a
sn1054 r 6.1 rdm.56 CRAY Y-MP
rho% ls -l /usr/local/etc/osh
-rwsr-xr-x  1 root       519576 Dec 14 08:50 /usr/local/etc/osh*
rho% /usr/local/etc/osh
neuman michael (mcn)
Operator Shell version 1.2
rho,/u0/mcn #> help
Operator Shell (osh) Version 1.2
   by Michael Neuman <mcn@lanl.gov> 12/93


Defines:
NO COMPILE_TABLE
LOGGING
CHECK_ACCESS
OPER_OVERRIDE

Commands accessible:
help   cd     more   alias  wc     ls     printenv    telnet
rlogin rm     cp     w      cat
```

```
rho,/u0/mcn #> ls $HOME
osh       tcsh      uudecode     uue       uuencode     yes       yes.c
rho,/u0/mcn #> alias
Current list of aliases:
rho,/u0/mcn #> alias ls ls -CF
Alias[0]: ls  ->  ls -CF
rho,/u0/mcn #> ls $HOME
osh/      tcsh*     uudecode*  uue/      uuencode*  yes*      yes.c
rho,/u0/mcn #> rm /etc/rc
rm: Permission denied
rho,/u0/mcn #> ^D
```

Administrator Features

To the administrator, the Operator Shell is easy to configure and extremely flexible. It is based on a command matrix which can either be compiled into the shell or held in a file. If the commands are compiled in they are global for every executor of Osh and file access control lists aren't allowed. This is primarily useful if your configuration requires several copies of the Operator Shell protected by mandatory access control. The command matrix contains a list of all valid users and groups and the commands they are allowed to execute. The submatricies are additive, so for example, the special global submatrix "ALL" could specify basic commands like *ls, cd*, and *help* which everyone should be able to run, "consultants" could specify commands which should be accessible by everyone in the *consultants* group, and "mcn" could specify commands which should only be accessible by user *mcn*. If *mcn* was a member of the *consultants* group, he would have the privileges of all three of those submatricies. If "ALL" is not specified, a user can only run Osh if his username or groupname is specified in the command matrix. Following is an example command matrix.

```
ALL
{
help NULL
cd NULL
more NULL
alias NULL
wc /usr/ucb/wc
ls /bin/ls
printenv /usr/ucb/printenv
telnet /usr/ucb/telnet
}
consultants
{
finger /usr/ucb/finger
}
mcn
{
rm /usr/bin/rm
cp /usr/bin/cp
w /usr/ucb/w
cat /usr/bin/cat
}
```

In each submatrix (delineated by the curly brackets), the first column specified the command name which the Osh user would type, and the second column specified the path to that command. If it's a built-in command, it's path is specified as NULL to indicate that no external program will be run. The built-in commands are *more, cd, help,* and *alias*. Command handlers can also be implemented by the administrator to allow for specific access rights checking before calling the program specified in the command table. Currently, the Operator Shell has handlers implemented for *cp, rm*, and *vi* to check writeability of command line specified files, and *ldcache* (a Unicos command to administer logical device caches) to ensure that no command line options are passed to it. Handlers can be easily added by the administrator.

The Operator Shell's more basic functions can be configured by editing the file "config.h". This file includes the defines which specify where the command matrix is, where the logfile should go, and other basic configuration options as shown below.

Table 1: Configuration Options

Option	Type	Action
COMPILE_TABLE	boolean	Build the command table into Osh
TABLE_NAME	path	Path to the command table if not compiled in
LOGGING	boolean	Turn logging on
LOGFILE	path	Path to the logfile if logging is turned on
CHECK_ACCESS	boolean	Check for key in file owner's directory
ACCESS_FILE	string	Filename for the key
OPER_OVERRIDE	boolean	Allow users in OPER_GROUP to override CHECK_ACCESS
OPER_GROUP	string	Name of group allowed OPER_OVERRIDE status

Security Features

The purpose of the Operator Shell is to improve security of a system and several features have been added to that end. The most important security consideration is that the Operator Shell is a setuid root shell, and consequently so are all of it's children. This means that once a program is executed by Osh, the system's security rests solely in that program until it returns. Several significantly important utilities rely completely on the mandatory access controls of Unix. For this reason, *more* and *vi* have been rewritten to remove their reliance upon Unix protections. The standard Berkeley *more* allows the user to execute shell escapes and pipes as well as enter into *vi*. It has been replaced with an internal version of *more* using the curses library. *Vi* allows the user shell escapes and pipes as well as the ability to read and write to any file. *Vi* has been replaced with the publicly available *elvis* modified to disallow filename changes so that the user may only read and write to the file specified on the command line. Internal to Osh, a handler has been added to perform file permission checks on the command line filename.

File read permission is granted only if access to that file is not specifically denied and if one of the following is true:
• The user could normally perform the read
• CHECK_ACCESS is on and the file owner has the key in his home directory
• OPER_OVERRIDE is on and the user is in the OPER_GROUP
• Read permission to the file is specified in the user's access control list
• Read permission to the directory the file is in is specified and read access to that specific

file is not denied.

File write permission is granted if the write is not specifically denied, if the user could normally perform the write, or if write permission to the directory the destination file is in is specified and write access to that specific file is not denied. These access control lists are specified as part of the command submatrix. For example, the following submatrix would allow the user *mcn* read and write access to the entire /etc directory except write access to the message of the day and read access to the shadow password file.

```
mcn
{
rm /usr/bin/rm
cp /usr/bin/cp
w /usr/ucb/w
cat /usr/bin/cat
+w/etc
-w/etc/motd
-r/etc/shadow
}
```

In addition to these access control mechanisms, thorough audit logs are kept. All typed commands are saved in the logfile with real user ID, date and time, command typed, and an indication of success (+) or failure (-). Following is a sample from the log where mcn is a user allowed to modify the kernel, but not anything else.

```
LOGIN: mcn ran osh at Tue Feb 15 18:54:46 1994
mcn (2/15/94 18:54:47): help      +
mcn (2/15/94 18:54:51): ls /etc      +
mcn (2/15/94 18:54:56): rm /etc/fstab      -
mcn (2/15/94 18:55:03): ls /      +
mcn (2/15/94 18:55:07): rm /vmunix.old      +
mcn (2/15/94 18:55:11): ls -l /      +
mcn (2/15/94 18:55:27): mv /vmunix /vmunix.old      +
mcn (2/15/94 18:55:47): ls -l /vmunix.old      +
mcn (2/15/94 18:55:55): cd /var/log      +
mcn (2/15/94 18:55:56): ls      +
mcn (2/15/94 18:56:04): rm syslog.7      -
mcn (2/15/94 18:56:06): cd /var/adm      +
mcn (2/15/94 18:56:07): ls      +
mcn (2/15/94 18:56:23): rm lastlog      -
logout: mcn left osh at Tue Feb 15 18:56:33 1994
```

In addition to access control lists and logging, security is enhanced in other, more subtle ways. By setting both the file and the explicit path to that file, it is assured that the commands are run from their proper location. This avoids many of the more common setuid program exploitations such as IFS and PATH environment variable modification. A useful side effect of the Osh program being setuid root is the command matrix file can be set with read permission to no one. Consequently, malicious users aren't given any hints as to whom has the most power and is therefore worth the most time to hack.

Internals

There are two primary components to the Operator Shell's internal security. The first is the means of executing external programs and the second is the file access protection mechanisms. The method of executing external programs has been carefully constructed to avoid many of the security holes involved with setuid programs executing other programs. The file access protection mechanisms are designed to allow the administrator to configure the exact permissions for file accesses and allow the consultant access to users' files, all without restricting the user beyond his normal capabilities.

When a the user types a command, it's entry is looked up in the command table. If the entry exists and has an associated path, it's corresponding handler is called (if there is one) which checks for command line file writeability, then the function execute() is called which determines if the program is a shell script or an executable, then calls execv(3) with the explicit path and options. The reason for determining if the file is a shell script or not is a large number of shell scripts do not have a "#!" header and execv(3) cannot handle these files directly, so the execute() function inserts the Bourne shell as the program to be executed. If the entry is internal, the command's handler is called directly.

If the command can write to files, or if the command line contains a redirect to a file, writeability is allowed if any of the following rules are true:
• The file is specified as +w in the user's access control list
• The parent is specified as +w, and the file is NOT specified as -w
• The file would normally be writable by the user and the file is NOT specified as -w
The commands which need to check writeability must provide handlers to perform the access checking internally. This is obviously necessary for commands like *cp* which require the first argument to be checked for readability and the second for writeability.

For every command, the default security checking routine will test each argument on the command line for file readability. If the argument doesn't exist as a file, then it is assumed the argument is an option and is ignored. For this reason, handlers must be created if argument writeability must be checked.

The only functions a handler needs to perform is a test of the validity of the command line options, a notation of the command's success or failure, and then a call to execute(). Handlers can be used to check writeability of an argument, force an option to a program, ensure arguments weren't sent to a function, etc. For writing handlers, the following routines are important.

Table 2: Support Functions

Function	Arguments	Return codes
acl()	char *filename, char mode	+1 if filename is listed as a + for the given mode, -1 if filename is listed as a - for the given mode, and 0 if the filename is not listed in the acl
check_access()	int argc, char **argv	+1 if all arguments are readable (or don't exist), 0 if one argument is not readable
writeable()	char *filename	+1 if filename is writable, 0 if not writable
execute()	int argc, char **argv	Executes the command line, should not return except on error with execv(3)

The check_access() routines iterates through all command line arguments before calling any handlers. It does this according to the following pseudo-code:

```
FOR i=0 TO number of arguments
   IF argument(i) +r in acl THEN
      NEXT i  { File is readable }
   ELSE
      IF argument(i) -r in acl THEN
         invalidate command line
         return from routine
      ENDIF
   ENDIF
   {if we get this far, that means the argument isn't in the acl}
   IF argument(i) exists as a file THEN
      IF real uid is allowed to read argument(i) THEN
         NEXT i  { File is readable }
      ELSE
         IF (not CHECK_ACCESS) or  ("ACCESS_FILE" exists in argument(i)
                owner's home directory) THEN
            NEXT i  { File is readable }
         ELSE
            IF (not OPER_OVERRIDE) or (user not a member of OPER_GROUP)
              invalidate command line
              return from routine
            ENDIF
         ENDIF
     ENDIF
   ENDIF
NEXT i
```

Handlers are added by the following procedure:
1) Add the handler function prototype to the prototype section in struct.h
2) Add the command name and handler function name to the Internal[] array in struct.h
3) Modify the NUMINT define right below the Internal[] array to reflect the number of internal handlers
4) Add the handler code to handlers.c
5) Recompile

IV. Summary

Osh is best applied to any situation where specific privileges need to be given to specific users. It allows the administrator to specify access to both commands and files for individual users and groups, automatically maintains audit logs, and is easily configurable and modifiable for any situation. These features make the Operator Shell superior to root password distribution or any other privilege distribution system. At LANL, the Operator Shell has cut the number of users who require the root password on our supercomputers from more than 50 to less than 10. The security and administration benefit of this is remarkable.

Comparison with Other Systems

Similar systems to the Operator Shell are Sudo (by Jeff Nieusma) and Runas (by Christopher Carpinello).

The interface to both Sudo and Runas is via the command line. For example, a user would type “sudo vipw” which, if the access control file allows it, will run vipw as root. This is a quicker interface than Osh for short administration tasks, but it doesn’t allow the administrator to protect commands from dangerous options or specify access controls to files.

Table 3: Comparison of Osh, Sudo, and Runas

Feature	Osh	Sudo	Runas
Interface	Complete shell	Command line	Command line
Access Control	Programs and Files	Programs ONLY	Programs ONLY
Commands run as	root ONLY	root ONLY	Any user
Command protection	Protection against any options with handlers	NONE	NONE
Logging	Command and it’s success or failure	Attempts to run Sudo	Attempts to run Runas

Availability

The Operator Shell is available in two ways:

Email: send mail to Mike Neuman <mcn@lanl.gov>
FTP: Anonymous FTP at ftp.c3.lanl.gov[128.165.21.64]:/pub/mcn/osh.tar.Z

Osh is known to run under Unicos 6.1 and 7.x, SunOS 4.1.3, and Ultrix 4.2.