SUS – An Object Reference Model for Distributing UNIX Super User Privileges

Peter D. Gray – University of Wollongong

ABSTRACT

This paper describes a system administration tool which allows a user to run a command as root or as some other user after authenticating. Unlike most other commands of that ilk, SUS attempts to treat the command and its arguments as references to system objects, and allows for relatively powerful matching on the attributes of those objects to determine if the user should or should not be allowed to execute the desired command. In addition, SUS has a mode to help limit the number of setuid utilities needed to provide user services via the web.

General Issues

It has long been known that the "all or nothing" traditional UNIX security model can be a serious inconvenience for many sites [1, 2]. The super user account (usually "root") has practically unlimited administrative powers while normal users are subject to all security restrictions. There are very few system administration tasks which can be performed without super user privileges.

At larger sites, it is common to have a team of system administrators of varying seniority, skill and experience, some performing fairly mundane tasks such as changing forgotten passwords.

Having a large number of people knowing the "root" password is not only inconvenient but is an obvious security hazard. In addition, permitting inexperienced or untrained staff to have access to such high levels of power and functionality poses a real risk to the integrity of computer systems. Simple typing mistakes can cause havoc.

There have been several packages written, both open source and commercial, to address this problem with various degrees of success [3, 4, 5]. To use these systems a user invokes the application (which runs with super user privileges via setuid) and passes arguments to indicate which command they wish to run and what arguments they wish to pass. The application consults a configuration file. Depending on the invoking user and the requested (target) command (and possibly other factors) the application will either allow the request and run the target command or inform the user that they do not have sufficient privileges to run the command they requested.

The criteria used in the above determination can be simple or complex depending on the tool being used. The paper by Hill [3] on the tool PRIV gives an excellent overview of the above issues and lists the capabilities of several such systems, both commercial and open source.

With the exception of PRIV, most such systems fail to examine the arguments of the target command with any degree of sophistication. PRIV offers a rich set of functionality along with some attempt to treat the arguments as references to system objects. It allows for command arguments to be restricted to objects which satisfy simple criteria.

SUS takes some of the ideas of PRIV and extends the concepts by adding the ability to treat all commands and arguments as references to objects. Such objects include users, files, groups, hosts and processes. Each such object has a set of attributes. For example, a file has an owner, a group and a type, etc. Objects on which the command will operate are examined, their attributes retrieved and compared with a selection criteria found in the configuration file.

Only if the selection criteria match the attributes of the objects is the command allowed. For example, it is possible to restrict a user to deleting regular files owned by any member of the group "admin" or only allow them to send a HUP signal to a process owned by a user whose UID is in the range 100-200.

Operation

SUS is installed as a setuid binary. When invoked to run a command it reads a single configuration file. The configuration file is preprocessed via a CPP-like macro preprocessor [9] with many predefined macros to allow for simplification of the control file syntax. For example, it is possible to restrict access to commands by time of day using SUS, even though there is no such capability in the SUS control language. The combination of conditional output in the macro preprocessor plus judicial use of the predefined macros set to the time of invocation provides the necessary functionality.

The presence of the macro preprocessor also allows for an easy mechanism for controlling the various operational aspects of SUS. By setting values of special macros in the control file, all configurable aspects of operation can be easily controlled. For example, the location of the log file can be set by simply setting the macro "LOG_FILE" to any desired location.

The configuration file syntax is similar to the popular sudo [4] product from the University of Colorado. Lines are composed of a user selector and a command selector. Each line is read and the user selector used to determine if this line applies to the current user. If so, the command selector is compared with the target command. If a match is found, the command is executed.

Objects and Attributes

SUS allows for commands (including arguments) and users to be matched as simple regular expressions. However, the real power of the system is its ability to treat command names and arguments as references to system objects. For example, in the command

```
$ sus rm a.out
```

SUS can treat the name ''a.out'' as a reference to a file object.

The object types supported by SUS are:

- USER, defined by the information returned by the getpwXXX(3) routines. Attributes which can be used for matching are the username, user id, group id, gecos field, home directory and shell. The home directory may be treated as a reference to a FILE (see below) and the primary and secondary groups may be treated as references to a GROUP object (see below).
- FILE, defined by the information returned by the stat(2) system call. Attributes for matching are the type, user id, group id, device, raw device and the file's real name. The owner may be treated as a reference to a USER object (see above) and the group to a GROUP object (see below). The parent directory may be treated as a reference to a FILE, PFILE or RFILE objects (see below).
- PFILE, identical to file except matching is performed on the referenced file's parent directory.
- RFILE, identical to FILE except matching proceeds recursively up the file system tree until a match is found or the root directory is reached.
- GROUP, defined by the information returned by the getgrXXX(3)routines. Attributes used for matching are the group name and group id.
- HOST, defined by the information returned by the gethostbyXXX(3) and getipnodebyXXX(3) routines.

```
// Make sure the target environment excludes
// dynamic linker/loader variables.
#define ENV_DELETE "LD_.*"
// Override the SHELL environment variable
#define ENV_ADD "SHELL=/bin/false"
// Allow user joe to add a user via a script
joe : /usr/local/bin/add_user.sh
// Set up a class of users whose home directory
// is in /home/sales using a CPP macro
#define sales USER(home=/home/sales/.*)
// Any user in the above group of users may change the
// ownership of any regular file to themselves
// (predefined macro SUS_USER) as long as
// the file was owned by another
// person in the sales group.
sales : chown SUS_USER FILE(type=reg, owner=sales)
// Trudy can send a TERM signal to any process whose
// group is "eng" or (because we use RPROC rather
// than PROC) has a process above it in the process tree
// whose group is "eng"
trudy : kill -TERM RPROC(group=eng)
// On the hosts in 130.130.62 subnet, andy can run anything
// he wants except the shells (ANY_COMMAND is a predefined macro)
#define SHELLS "/bin/sh | /bin/ksh"
#define secure-hosts HOST(ip=130.130.62..*)
andy @ secure-hosts : !SHELLS, ANY_COMMAND
// Bruce can run anything in the gurus directory with
// any arguments he likes
bruce : FILE(name=/share/gurus/bin/.*) ANY_ARGUMENTS
```

Listing 1: System capabilities.

Attributes for matching are the names and IP addresses (both V4 and V6).

- PROC, defined by the information available in /proc/<pid>/psinfo. Matching can be performed on the process id, the parent process id, the group id and effective group id, the session id, the user id and effective user id and the controlling tty. The user id and effective user id may be treated as references to a USER object, the group id and effective group id may be treated as references to GROUP objects.
- PPROC, identical to PROC except matching is performed on the referenced process's parent process.
- RPROC, identical to PROC except matching proceeds recursively up the process tree until a match is found or the root of the tree is located.
- REGEXP, extended regular expression matching. This is the default if no object class is specified but may be called explicitly if desired.

All of the above classes except REGEXP have an additional attribute "exists" which matches if the referenced object actually exists. This allows restricting an operation to a existing or non-existing object. For example, you may allow someone to create a new file, but not edit existing files.

Example Configuration

The configuration file syntax is basically defined as:

```
user-selector : allowed-commands
```

The preprocessor [9] is general purpose and macros may be defined to control its operation. In its default configuration it closely resembles CPP. Other styles it can support in terms of macro definition and comments are TeX, HTML and PROLOG.

The fragments of configuration file in Listing 1 demonstrate some of the capabilities of the system. C++ style comments are used for annotation.

Operational Notes

Some caveats are in order:

- all matching is done as strings.
- all string matching is performed as anchored, extended regular expressions.
- command lists are matched left to right and matching stops as soon as a match is found.
- if a command matches the entry in the file, but the match expression is negated, searching stops and the command is not allowed.

Target Command Environment

SUS allows practically complete control of the environment of the target command, including the ability to selectively remove or pass through environment variables from the environment of the invoking user which match regular expressions defined in the control file as well as adding or replacing environment variables with new values.

A current directory for the target command along with resource limits may also be set if required. Signal handlers and masks are set to default settings before the target command is invoked.

Security

SUS has been written with care to avoid any problems with buffer overflows or other potential security problems. Buffer size checks are performed on any operation where overrun could be possible. It checks that the control file is owned by "root" and is not writable by other users. Space for all data structures is dynamically allocated and there are no built-in limits in data sizes other than those set by configuration.

Logging

SUS logs all invocations for any reason, successful or not, using either syslog or straight to a file or both. All aspects of the logging operation are controlled by the configuration file. Information in the log records includes the invoking user, the current directory and the target command and arguments. Optionally, the resource usage and elapsed clock time for each target command may be logged as well.

Timestamps

Normally, SUS will force each user to authenticate by supplying their normal system password. When invoked successfully, SUS stores a timestamp for each user in a system directory, normally the root directory, and (optionally) the user's home directory. If a user has invoked SUS successfully inside a short period (configurable) then the user does not have to authenticate. Storing the timestamp inside a user's home directory allows for SUS to remember invocation across multiple hosts, as long as the home directory is shared.

Timestamps include the username, the user id, a SHA1 [7] checksum of the user's encrypted password and a SHA1 checksum of the actual timestamp itself. The root directory timestamp also includes a SHA1 checksum of a string based on the user's plain text password. This checksum field is empty in the home directory timestamp but is included in the SHA1 checksum of the home directory timestamp. This means that it is impossible to compute the checksum of the home directory timestamp without access to the root timestamp. Thus any tampering with the home directory timestamp is detectable. All checksums are checked to ensure a timestamp is valid.

The authentication step is skipped if the root timestamp is valid and current or the root timestamp is valid and the home directory timestamp is valid and current.

Any change to a user's password, username or user id cause all timestamps to become invalid.

CGI Support (Promiscuous Mode)

It is becoming increasingly necessary to allow users to perform tasks previously only offered if the user connected directly to a host (either via TTY style or X11 protocols) via a web interface. Some simple examples are the ability for a user to change their password or query their disk quota. To solve these sorts of problems CGI programmers often have to resort to setuid wrappers or scripts with obvious security ramifications.

SUS offers the capability to allow selected users to run commands as another user on the system if they can supply the target user's username and password. Effectively, user 1 can run a command as user 2 if user 1 can supply user 2's username and password.

For example, SUS can be configured to allow the web server user (say user WWW) to run a command as a normal user. The user can supply their own username and password, which is passed into SUS. SUS will authenticate the target user, then run a command (possibly a script) with appropriate arguments to perform the required operation. The command is run as the authenticated user. This mode of operation goes by the rather intimidating name of "promiscuous mode."

As an example, a site could write a script to change a user's password if run as that user. A CGI script could call the password changing script via SUS. If the user supplies their own username and password to the CGI script (and hence SUS) the password changing script is run as the authenticated user.

The overall result is that any number of scripts can be written which can run as any user who can supply their own username and password without installing any additional setuid binaries other than SUS. The overall number of setuid utilities is kept to a minimum. Exactly the same sorts of controls on who can run what exist for "promiscuous mode" as when operating normally.

Conclusion and Future Work

SUS allows for very fine grained control over what users may or may not do as root. Arguments to commands are examined to determine which system objects they refer to and the attributes of those objects may be used as criteria to allow or disallow the command.

SUS is under active development. Possible future work includes adding an ability to log a transcript of a user's TTY session with a target command and the capability to reliably prevent shell escapes from target commands such as editors.

Availability

SUS is freely distributable under the GPL and available from http://pdg.uow.edu.au/sus It is developed and tested under SOLARIS 8 but is known to compile and run under LINUX. It contains no dependencies on non-standard libraries.

References

[1] Ritchie, Dennis, "On the Security of UNIX," *UNIX Programmers Manual Volume 2*, AT&T, NJ, USA.

[2] Jordan, Carole S., "A Guide to Understanding Discretionary Access Control," *Trusted Systems (NCSC-TG-001)* September, 1987.

[3] Hill, Brian C., "Priv: Secure and Flexible Privileged Access Dissemination," *Proceedings of the Tenth Systems Administration Conference (LISA 96)*, USENIX Association, Berkeley, CA, USA.

[4] "Sudo," University of Colorado, ftp://ftp.cs.colorado.edu/pub/sudo .

[5] "PowerBroker," FSA Corporation, http://www.symark.com/ .

[5] Ramm, Karl, and Michael Grubb, "Exu – A System for Secure Delegation of Authority on an Insecure Network," *Proceedings of the Ninth Systems Administration Conference (LISA 95)*, USENIX Association, Berkeley, CA, USA.

[6] Chris Thorpe, "SSU: Extending SSH for Secure Root Administration," *Proceedings of the Twelfth Systems Administration Conference (LISA 98)*, Usenix Association, Berkeley, CA, USA.

[7] "Secure Hash Standard," *Federal Information Standards Publication (FIPS) 180-1*.

[8] "A Secure Environment for Untrusted Helper Applications," *Proceedings of the Sixth USENIX Security Symposium*, Usenix Association, Berkeley, CA, USA.

[9] "GPP – Generic Preprocessor," Ecole Polytechnique, http://math.polytechnique.fr/cmat/auroux/prog/gpp.html .