Distributed User Support on Exxon's Cray Supercomputer

Bonnie L. Hall, Exxon Upstream Technical Computing

ABSTRACT: *Exxon Upstream Technical Computing Company (EUTeC) has developed a system to allow distributed user support. The Customer Representative Interactive System for Processing IDs (CRISPI) is a local application that has increased our efficiency and enhanced our security and auditability. The application is easy to use and easy to maintain. CRISPI allows root users to minimize the amount of time spent satisfying user requests. A basic version of the application is simple to implement and may evolve into a complex system that completely manages a site's customer support needs.*

The application design is intended to be flexible. Any site could implement their own version of CRISPI using any language that CRI supports. This paper will examine the benefits of such a system and will provide the information necessary to design a CRISPI type application for a typical UNICOS system.

1 Justification

1.1 Limit Root Access

EUTeC developed this system for both efficiency and security. Every user community has many requests that require assistance from system administrators with root access. A ratio of 100 generic users to each root user is not unusual. The CRISPI application provides non-root users at distributed sites the ability to manage many of the users' requests. Requests that may be handled by a CRISPI system include:

- Group Changes
- NFS Mapping Changes
- Password Resets
- Login Account Maintenance
- Billing Account Code Maintenance
- Data Access Changes (chown, chgrp, chmod, ACLs, MLS labels)
- Creating directories and moving data
- NQS service queue and job maintenance
- Reviewing contents of various system logs
- Updating UDB privileges and resource limits
- Providing access to an MPP

With the dispersion of these tasks, the root users will have a great deal more time to concentrate their efforts on installing UNICOS updates and implementing system enhancements.

1.2 Positive User Identification

Security issues provide two good reasons to implement a CRISPI system. Positive user identification becomes difficult when a user community is spread over a large geographic area. A common way to solve this problem is to establish a customer representative contact (C-REP) at each remote site. All user service requests must be directed through the site's C-REP via e-mail or other verifiable means. The use of a C-REP for positive user identification is secure, but not efficient. A great deal of time and energy may be saved by providing the C-REPs with the tools necessary to satisfy the users' requests themselves. However, it would not be prudent to allow C-REPs uncontrolled access to root. CRISPI provides a menu based system to safely perform administrative functions on behalf of the user.

1.3 Enhanced Audit Trails

A well-developed CRISPI system will also solve problems which arise when interpreting the contents of administrative audit trails. It is important to have a log of all changes performed on the users' behalf and a record of authorizations obtained. One of the shortcomings of all UNIX systems is the lack of a single, robust audit trail for all types of changes. CRI does provide a number of good audit trails including MLS system logs, nu application logs, the su log and several others. However, these logs are all have different formats and different information. Technical expertise is required to interpret these logs and reconstruct administrator activities. CRISPI allows a customized audit trail for all administrator activities with a standard change record that makes reporting a breeze.

1.4 *Consistent Treatment of Requests*

The outcome of the more involved changes may vary when different root users satisfy customer requests. The handling of a user request for MPP access provides a good example. Several resource limits must be set and other resource limits should be increased. Often the user is added to new groups and given a directory in a new file system. The customers do not know what to expect if different administrators satisfy the MPP access requests in different ways. Frustration is bound to arise. A single script is used to perform all the changes when all MPP access requests are handled via CRISPI. No one will forget to set a limit or add a directory and all the requests are handled consistently. The customer knows exactly what to expect when a request for MPP access is granted.

2 System Components

2.1 *Contents of a CRISPI home directory*

To efficiently implement CRISPI it is necessary to maintain a uniform structure across all the CRISPI login accounts. Provide each C-REP with a home directory that contains a customized menu and a log of their activities. All the CRISPI users will use the same source code to satisfy the supported user requests. The CRISPI home directories should all reside in the same file system and the directory names should match the CRISPI login name. Each CRISPI user is given a login in a standard format such as "REPxxx" with home directory /users/REPxxx. Let xxx refer to a unique identifier for each CRISPI user. The advantage to the naming conventions becomes obvious when scripts are created to manage all CRISPI logins in one fell swoop.

2.1.1 *The Customized CRISPI Menu*

The CRISPI application is accessed through a menu interface which is the login shell. The CRISPI user interactively supplies answers when prompted for information. Not all the C-REPs are allowed the same level of access. Different main menus control which customer requests a given CRISPI login may satisfy. This is referred to as the "level" of the CRISPI account. Each CRISPI level provides access to all privileges granted to the lower levels. A site may decide that no levels are necessary and each CRISPI user may access all the functions. The following is an example of a possible three-level CRISPI implementation:

1. Level one CRISPI is given to C-REPs in remote locations. These CRISPI users provide services for a particular group of people and no others. The level one CRISPI users are able to manipulate existing login accounts or data and groups that are owned by their local site.
2. Level two CRISPI is for use by a central security administration team. Level two CRISPI users have authority to provide service to anyone in the user community. This centralized group handles adding and deleting login accounts and groups, granting MPP access, setting certain privileges or resource limits and manipulating NQS queues. A level two CRISPI user may define ownership for groups, data and login accounts to allow a level one CRISPI user to manage the entity.
3. Level three CRISPI is reserved for root users. All the functions provided by CRISPI could also be performed as root, but there are good reasons to use CRISPI instead: it retains local audit trails, provides consistent handling of requests and it takes less time. Functions that are restricted to level three CRISPI accounts include moving data between file systems, creating file links, adding new CRISPI accounts, and full use of udbgen.

Note that it is often desirable to provide sub-menus for CRISPI logins that provide many options. More than 8 options will clutter the menu and make it difficult to use. The following menu is an example of a CRISPI main menu for an account that can access many functions. Options 1 - 6 lead to various secondary menus.

```
                         Welcome to the CRISPI System
Please choose an option:

     1) NFS UID/GID Map Changes
     2) Update User Resource Limits
     3) Change Group Membership
     4) Change Data Access Permissions
     5) Manipulate Existing User IDs
     6) Review/Report Contents of Security Log
     7) Obtain Information on a User
     8) Exit the CRISPI System

Enter Number of Choice -->> ______
```

Figure 1

Option 7, "Obtain Information on a User", calls a local script. The CRISPI user may input any search criteria that may be found in an /etc/passwd entry, such as name, telephone number, or login ID. The script will locate all possible matches and allow the CRISPI user to choose a single login ID. A full screen of data is then provided for the chosen login including UID, name and phone, all groups, valid account codes, last login, and status of the account (enabled, password disabled, udb disabled, etc.). This tool is essential to CRISPI users who only access the system via the CRISPI menu.

2.1.2 Authentication Routine

Each CRISPI account must contain an authentication routine. To restrict some CRISPI logins to changes for users at a given site, it is necessary to know where each user is assigned. An effective method is to query a database on another machine. Payroll computers are kept up to date on all personnel transfers. Create a cron job to query this computer several times a day and put each user into a group that represents their organization. For instance, a group named ORGxxx may be used. The characters ORG allow the programmer to manipulate all the groups at once while xxx is a unique identifier for each group. There is an added benefit to using ORG groups - it is trivial to detect when users transfer between divisions or leave the company.

Once every user is in an ORG group to uniquely identify their organization, define a variable to store all ORG groups accessible to a given CRISPI login. For the sake of example this variable will be referred to as the ALLOW variable. The ALLOW variable and level of the CRISPI menu are the only two pieces of code that vary between various CRISPI login accounts.

2.1.3 Scripts and C programs to satisfy requests:

All CRISPI accounts access the same CRISPI source code. The menu level controls which function each user may perform.

The implementer may decide to keep different scripts to satisfy each type request, build change handling directly into the menu program or to create a hybrid scheme. Similar changes may be handled by one program while short changes may be built directly into the menu. It is important to keep the principle of least privilege in mind when deciding an implementation scheme for request handlers. Only the piece of code that requires privilege should assume privilege. All other code should run as a standard user.

It is a good idea to build questions into your code to confirm certain actions. When the option to remove an account is selected the application may prompt:

```
Are you sure you want to remove xxxxxx and
  ALL THE FILES in his home? (y/n) ==>
```

If the CRISPI user supplies any response other than Y or y, exit the code with an error message.

It is also useful to show the CRISPI user the result of a change. The application may display all groups associated with a user after a new group has been successfully added. The CRISPI user can see that the change took place correctly. Conversely, the code should exit in a manner that will get the attention of the CRISPI user in the event of an error. The error handling code should display messages that will make it possible for the implementer to locate the source of the problem.

2.1.4 The CRISPI Log

One of the most convenient features of CRISPI is the logs which are cut as requests are satisfied. One record cutting routine should be called by all the source code. This ensures uniform records in the log. The logs may be stored inside the CRISPI home directory. Scripts may be run weekly or monthly to send the contents of each CRISPI log to a report. Section 3.2.3 gives examples of CRISPI logs and reporting mechanisms.

2.2 CRISPI Utility Jobs

All source code needed to create a CRISPI home directory resides in a common area with several jobs that are used to maintain the CRISPI accounts. These utility jobs may be invoked through a high level CRISPI account or run by a root user. The CRISPI utility jobs may include the following scripts or programs:

2.2.1 CRISPI Account Installation Routine

CRISPI accounts may be installed by means of a script that takes a login name, UID, level and the ALLOW variable as input. The job can do a udbgen -v (show all fields) on a current CRISPI ID, then replace all occurrences of the user name, the UID number, and the "update" directive with a "create". The directives are then redirected into udbgen to create the account. The script will then set the ALLOW variable for the authentication routine, create the home directory, provide a header to start the log file and place the appropriate menu inside. Note that the level parameter decides which menu will be provided, and all other source is the same for each CRISPI user. The last step is to have the installer assign a password to the new account. Set "force" in the pwage field so the new CRISPI user will have to reset the password at first login. At this point the CRISPI account is complete and ready for use.

2.2.2 CRISPI Account Update Routine

The utility job for complete system updates may be modified to update a single program. One important feature is that all CRISPI accounts are updated at once. It is important to retain uniformity among accounts to keep CRISPI maintenance simple. Only the menus and the value of the ALLOW variable should differ from login to login. A single test CRISPI ID is used for testing system updates. This test ID does not follow the naming conventions used by the other accounts so it is not affected by the utility routines and may be maintained by hand as the programmer sees fit.

Due to the naming conventions the update process is simple. If all the CRISPI accounts begin with the letters REP and the homes reside in /users, a loop on /users/REP* is sufficient to collect information and install the updated system.

2.2.3 CRISPI Reporting

The reporting scheme is also kept simple by use of convention. Customer organizations supported by a given CRISPI account are identified by the ALLOW variable. All CRISPI source code invokes the same routine to record actions of the request handling programs in a log in the CRISPI home directory. The following example illustrates one possible log implementation:

Routine "cutrec" writes four lines to the log for each change made. Each of the lines begins with the customer's ORG group (see section 3.1.2). The first line is blank to make the reports easy to read. The second line records the date and the name of the CRISPI account. The third line displays a line provided by the request handler to describe the change. The fourth line allows the C-REP to enter a comment, such as a record of how the change was authorized. An invocation of cutrec from the password update request handler would look like:

```
cutrec $cust "Password updated for
customer"
```

The cust variable is the login ID serviced, the phrase is used in line three to describe the change. The user would be prompted by cutrec to enter the fourth line by cutrec invoking the `line` command. The customer's organization group is found by using the $cust variable to query /etc/group or the UDB. Consider a customer JaneJones who is a member of group ORGmathstud and a CRISPI login account named REPjohnDoe. The CRISPI Log entry for a password update would appear as:

```
ORGmathstud:
ORGmathstud:May 12, 1995 11:30:33:
            Request handled by REP-
            johnDoe
ORGmathstud:Password  updated  for
            customer JaneJones
ORGmathstud:Comment:  Jane is in my
            office with student ID
            badge
```

The reporting of the log contents may be accomplished by means of a data file to identify the name of each report and the ORG groups reported therein:

```
MathDept:ORGmathstud,ORGmath-
         teach,ORGmathTA
CompDept:ORGcompstud,ORGcomp-
         teach,ORGcompTA
```

The reporting routine loops through the data file and creates a report for each line. Each report contains information from the listed groups. Gathering the information is extremely simple. Assume the CRISPI logs are stored in each CRISPI home directory under the name of "rep.log". To create the MathDept report the script would issue:

```
egrep  -e  "^ORGmathstud:|^ORGmath-
teach:|^ORGmathTA:" \
  /users/REP*/rep.log>$rpt/MathDept
```

The entire report is created easily with one command per report. Once all the reports are created, copy the rep.log files to an archive location and initialize each report for the next reporting period by copying a header over the original file.

3 Security Considerations

The security of this application was a consideration throughout its design. Privileged access is being provided via this interface. It is essential that security guidelines are followed to design an acceptable CRISPI system that is in full control of all privileges and root usage.

3.1 The Restricted Shell

The key to the security lies in the use of a restricted shell. Each CRISPI menu is actually the shell for the CRISPI login. This means that the login cannot be used for any other purpose on the system. The CRISPI menu is displayed when the CRISPI user logs into his account. The user may only select options that are provided by the menu. If the CRISPI user breaks out of the menu, the session is ended and the user is no longer logged into the system.

It is easy to confine a user to a restricted shell. The shell field in the UDB contains the name of the CRISPI menu instead of the typical /bin/sh, /bin/ksh or /bin/csh programs. Fictitious CRISPI user REPjohnDoe has a CRISPI home directory of /users/REPjohnDoe. The menu is the shell and it is named rep.menu. The entry in the UDB would appear as:

```
:shell:/users/REPjohnDoe/rep.menu:
```

The primary risk introduced by this scheme is that the CRISPI implementer could install the wrong level DSA menu or set the ALLOW variable too lenient. Automated scans of each CRISPI menu level and ALLOW variable would detect such an error. The CRISPI implementer should only provide accounts to people who have been approved by management. Each CRISPI user should sign an agreement to use the login for intended purposes only. In the event that the implementer did provide the wrong menu or ALLOW list, the CRISPI user would be able to detect, and *should* report, the error. There is little danger that the CRISPI user could accidentally abuse privileges because the user drives all actions taken by the application.

3.2 The System Security Configuration and CRISPI Implementation

There are currently three types of MLS Unicos systems available today, PRIV_SU, PRIV_TFM and PAL. The security mechanism used will effect the CRISPI implementation.

The CRISPI login ID is going to need system privileges to satisfy requests. This is not a security exposure because the CRISPI login may only perform actions completely specified by the restricted shell. The method used to give the restricted shell system privileges will differ depending on the system MLS implementation.

3.2.1 PAL Based Security

In a system with PRIV_SU and PRIV_TFM disabled, Privilege Access Lists (PALs) completely control all system privileges. The CRISPI users will require different privileges depending on the functions performed by the restricted shell. The most restrictive solution would be to custom define a category for the CRISPI users and customize the PALs on the binaries used by the application. The least restrictive possibility would be to define the intcat and valcat fields in the UDB to secadm and allow the CRISPI applications to run as secadm at all times, with full privileges enabled. This solution does not follow the principle of least privilege and has risks if the CRISPI code is not implemented properly. Any site with PAL based security would need to consider the options carefully and custom design a solution for their site.

3.2.2 PRIV_SU Based Security

On a PRIV_SU system the root ID is all powerful. The CRISPI application must assume root power whenever special privileges are required. This is easy to do via a Set UID to root (SUID) program. A SUID program assumes the privileges of the file owner during execution. SUID is implemented via the permission mode settings on a file as explained in the chmod man pages. UNICOS differs from most UNIX systems in that a shell program will not assume the file owner privileges even when the SUID mode bit is set. A binary file must be used to issue a SUID command and assume privileges. This works out well for the administrator who is using shell programs to design a CRISPI application. A simple C program may be written that includes two system calls, setuid and execl. Use the setuid system call to assume root privileges, then issue the execl system call with a command as its parameter. Compile this simple C program and chmod the binary file to set the suid mode bit. Set the executable's owner to root. Call this program to execute the command parameter of the execl system call as root.

Whenever a script needs to execute a given command from root, the script will call the c program with the command as the parameter. If the complied C program is called `rootexec` and it resided in a directory $src, issue the command *ls -l /users/REP*/rep.log* as root by preface the entire string with the program call:

```
$src/rootexec ls -l /users/REP*/rep.log
```

This method has a built in protection mechanism. Most of the CRISPI code does not need to run as root and the programmer will only invoke suid to root programs when root privileges are required.

The primary exposure introduced by use of suid programs results from programmer error. It is important to only call commands which do not give the user access to a shell. For instance, the vi editor allows shell commands. If the vi editor were invoked suid to root than the user has to all shell commands as root. This must be avoided. Any UNIX security text would discuss this subject in greater detail. Another risk may arise from improperly defined file access. It is essential to restrict execute access on any suid to root program. More detail is given on this subject in section 2.

3.2.3 PRIV_TFM Based Security

PRIV_TFM refers to the Trusted Facility Management privilege mechanism that will be removed from the system in UNICOS 9.0. Since this mechanism is slated for removal, very little detail will be given about its uses. The easiest way to implement a CRISPI application that is compatible with this privilege mechanism is to design the application for a PRIV_SU system and add:

```
:intcat:secadm,sysfil:valcat:sec-
adm,sysfil:
```

to the UDB entry for each CRISPI account. The CRISPI application is then ready for migration to UNICOS 9.0 PRIV_SU system. NOTE: Do not use secadm category for TFM purposes on a system that is migrating to PALs, the category has different meanings under each system. PRIV_TFM systems that are migrating to PALs should implement CRISPI using PALs.

3.3 File Permissions and Owners

This section assumes a CRISPI implementation under PRIV_SU or PRIV_TFM (i.e., suid to root files will be used instead of PALs). Protecting the suid files from unauthorized execution is extremely important. It is best to set file permissions as restrictive as possible when dealing with SUID to root programs.

3.3.1 CRISPI Source Code Directory

Protect CRISPI source code from everyone except root. Keep all the source code in one directory that is set to mode 700 and owned by the CRISPI implementer or root. The request handling programs should be executable by CRISPI users only. Consider using ACLs for added protection. Keep everything associated with the application under a protected directory. Keep test files separate from production code, but make sure that the test files are also carefully protected. In an environment with multiple root users and lax change control procedures, the CRISPI implementer should install scans that ensure that all protections placed on CRISPI programs stays intact.

It is important to protect all suid to root programs individually, even with protections placed on the parent directory. The suid files must be owned by root to run as root. Create a group for exclusive CRISPI use to control who executes the suid to root files. Set the suid files to root owner and group CRISPI, then chmod to 4410. A 4410 file mode means: set the suid bit, allow user (root) read access and allow group execute. No access is allowed for all other users on the system.

3.3.2 CRISPI Home Directories

It is also essential to protect the CRISPI home directories from outside access. The CRISPI home directories may be owned by the CRISPI user and the mode set to 700. The CRISPI log file is the only file which a CRISPI user should have write permissions.

As long as the proper file permissions are in place there is no exposure created by the existence of the CRISPI application. It is up to the administrator to ensure that all CRISPI accounts and source code are properly protected and that the protections remain in place.

4 How To Get Started

Designing your own CRISPI application will provide your site with vastly improved audit trails, faster turn around time on customer requests and a mechanism to handle all request in a uniform manner. It is not necessary to invest a great deal of time in the initial CRISPI system. The key to a successful CRISPI implementation is to start small and build the application over time.

Every system administrator has customer requests that are time consuming. Start to develop a custom CRISPI application by writing scripts for common requests. The first time a request is made it will take extra time to put the solution into a script but time will be saved on subsequent requests. A CRISPI menu to provide access to the request handlers is easy to program. The initial version of CRISPI may simply provide just two options: Change a User Password and Exit CRISPI. This simple version of CRISPI would take only a couple of hours to create, install, and educate C-REPs in its use. The request handler for a password change is very simple: invoke /bin/passwd as root, run cll -r on the user and set the force bit on the pwage field in the UDB entry. An initial authentication routine (section 3.1.2) may be as simple as checking group membership.

Once contacts are established in the user community, the basic CRISPI application can be put into production. The C-REPs will handle a few basic user requests and the system administrator may use the time saved to add more request handlers for the CRISPI applications. As the application matures, efficiency will increase for both users and administrators. The users will be able to get help from a contact at their site and will not have to repeatedly track down a root user for assistance. Once the application is mature, root users will be able to invested a much larger portion of their time in system enhancements and other projects.