VALab Remote Execution

The aim of this project is to enable a user to use particular routines, which we will call *modules*, without having to resort to executables or source codes located on your server. This can be done by executing the routine by remote access on your server. We provide a program which allows the user to send to the VALab server the input file(s), containing the data that a particular module requires, and to receive the output file(s), containing the results. VALab server provides to forward all the data to your server. Note that, as the remote access on your server will always pass through VALab, the user will never contact your server directly and will never know the location of a module executed on your server. The method used to exchange the files and information is electronic mail.

How to install the VALab remote execution robot on your linux server.

1. Prerequisites:

   1.1. Perl must be installed. At least version 5.8.4 it's necessary.

   1.2. It's necessary to have a mail server installed on your server. VALab server uses SendMail and we suggest to use the same mail server (www.sendmail.org). Actually, the only strictly necessary feature that the mail server should have, is the support to the `$HOME/.forward` standard (also known as *dot-forward* or *dotforward*). See Appendix 1 for more information about SendMail.

   1.3. VALab server uses also *smrsh*. *smrsh* is a restricted shell utility that provides the ability to specify, through a configuration, an explicit list of executable programs. When used in conjunction with SendMail, *smrsh* effectively limits SendMail's scope of program execution to only those programs specified in *smrsh*'s configuration. The purpose for restricting the list of programs that can be executed in this manner is to keep mail messages (either through an alias or the .forward file in a user's home directory) from being sent to arbitrary programs which are not necessarily known to be sufficiently paranoid in checking their input, and can therefore be easily subverted. See Appendices 2 and 3 for further information on *smrsh* and security issues. We strongly recommend to use *smrsh* and to place the provided *robot.pl* code in the folder */etc/smrsh/*. This precaution will eliminate any security problem related to the remote execution.

2. Extract in a temporary directory of your choice the provided archive *Valab-Remote-Execution-1.0.tar.gz*:

   ```
   tar –xzvf Valab-Remote-Execution-1.0.tar.gz
   ```

   A *Valab-Remote-Execution* directory will be created. Move into the directory:

   ```
   cd ./Valab-Remote-Execution
   ```

   If you list the content of the directory you can find the following files and directories:

   - readme – this file.
   - Mail-SendEasy-1.2.tar.gz – compressed file archive containing the Mail::SendEasy perl module.
   - robot – directory containing the main perl executable.
   - dot-forward – directory containing a .forward example file.

- prog_data – directory containing the data configuration program.

3. Install the provided perl module Mail::SendEasy.

```
(starting from the Valab-Remote-Execution directory)
tar –xzvf Mail-SendEasy-1.2.tar.gz
cd ./Mail-SendEasy-1.2
perl Makefile.PL
make
make test
make install
```

4. Copy the perl program *robot.pl* in the right directory.

```
(starting from the Valab-Remote-Execution directory)
cd ./robot
cp robot.pl /etc/smrsh/
cd /etc/smrsh/
chown root:root robot.pl
chmod +x robot.pl
```

   If you don't use the smrsh restricted shell, you can place the *robot.pl* file in any other folder you choose instead of */etc/smrsh/.*

5. Create a user called "robot".

6. Create (if it doesn't still exist) the *.forward* file in the robot home directory (*/home/robot/.forward*). Edit the *.forward* file and write the following line (with double quotes).

```
"|exec /etc/smrsh/robot.pl"
(leave a blank line)
```

   Note the blank line to be left at the end. Obviously, if you copied the *robot.pl* file in a different directory, specify that instead of */etc/smrsh/*. A copy of this file can be found in the *dot-forward* directory.

   Note: Actually, if you have the *smrsh* installed, you can just write "|exec robot.pl" as the entire path (if any), before the executable file name, will be automatically ignored and substituted with */etc/smrsh/*.

7. Create a directory where the remote execution temporary files will be placed. For example:

```
mkdir /home/robot/tmp/
```

8. Create a directory where the remote execution working directories will be placed. For example:

```
mkdir /home/robot/exec/
```

9. Create a directory where the remote execution log files will be placed. For example:

```
mkdir /home/robot/logs/
```

10. Now, you have to create the *prog.data* file, containing all the necessary information about the modules that you want to make available on your server.

    10.1. Copy in a directory (e.g. */home/robot/prog_data/* can be a good choice) the files *prog_data.pl* and *prog_data.ini.*

```
(starting from the Valab-Remote-Execution directory)
cd ./prog_data
cp prog_data* /home/robot/prog_data/
cd /home/robot/prog_data/
chmod +x prog_data.pl
```

    10.2. Edit the file *progs_data.ini*. For each program you want make available, you have to write some information (hereafter called section). The following is an example of a section:

```
module_name        = mie_robot
program_name       = mie_homogen_sphere
path               = /home/robot/progs/mie_homogen_sphere
input_files        = input1.in  ,  input2.in
output_files       = output.out
default_parameters =
error_file         = mie_homogen_sphere.err
help_file          =
description        =
```

- *module_name* is the name that the user will specify to select the execution of the module.
- *program_name* is the name of the called program. It is your internal name, the user will never see this name.
- *path* is the path where is located your executable .
- *input_file* is the list of the input files required by the program. The names are separated by commas. If the program doesn't have fixed input file names, but, for example, they will be specified by the user in the command line, simply place a * in place of names.
- *output_file* is the list of the output files required by the program. The names are separated by commas. If the program doesn't have fixed output file names, but, for example, they will be specified by the user in the command line, simply place a * in place of names.
- *default_parameters* is an optional string of default command line parameters passed to the executables. The parameters will be passed exactly as they appear after the =.
- *error_file* is the name of the optional file that contains the errors code explanation of your program. If the file is specified and an error occurs, robot will show the error code number and will search in the file for the associated text string (that usually give an extended explanation of the error). If no file is specified, in case of error, robot will show only the error code number.
- *help_file* is the name of an optional file that contains some help information that will be replied to the user in case of errors in the input parameters (command line and/or input files).
- *description* is an optional brief description of the program.

Note that all the nine fields of a section are required (not necessary in the given order). Note also that the only fields that can be empty are *default_parameters*, *error_file*, *help_file* and *description*. All the other are required. Remember that if input or output files are not required you must place a * in place of file names.

10.3. Execute the *prog_data.pl* program.

```
cd /home/robot/prog_data/
./prog_data.pl
```

10.4. Make sure that the file *prog.data* has been properly created.

```
cd /home/robot/prog_data/
ls prog.data
```

10.5. Make sure that there are no errors, that is the error file is empty.

```
cd /home/robot/prog_data/
cat ./error_file.log
```

At last, send a copy of *prog_data.ini* to valab@valab.det.unifi.it, we need some information on your programs in order to properly configure VALab server.

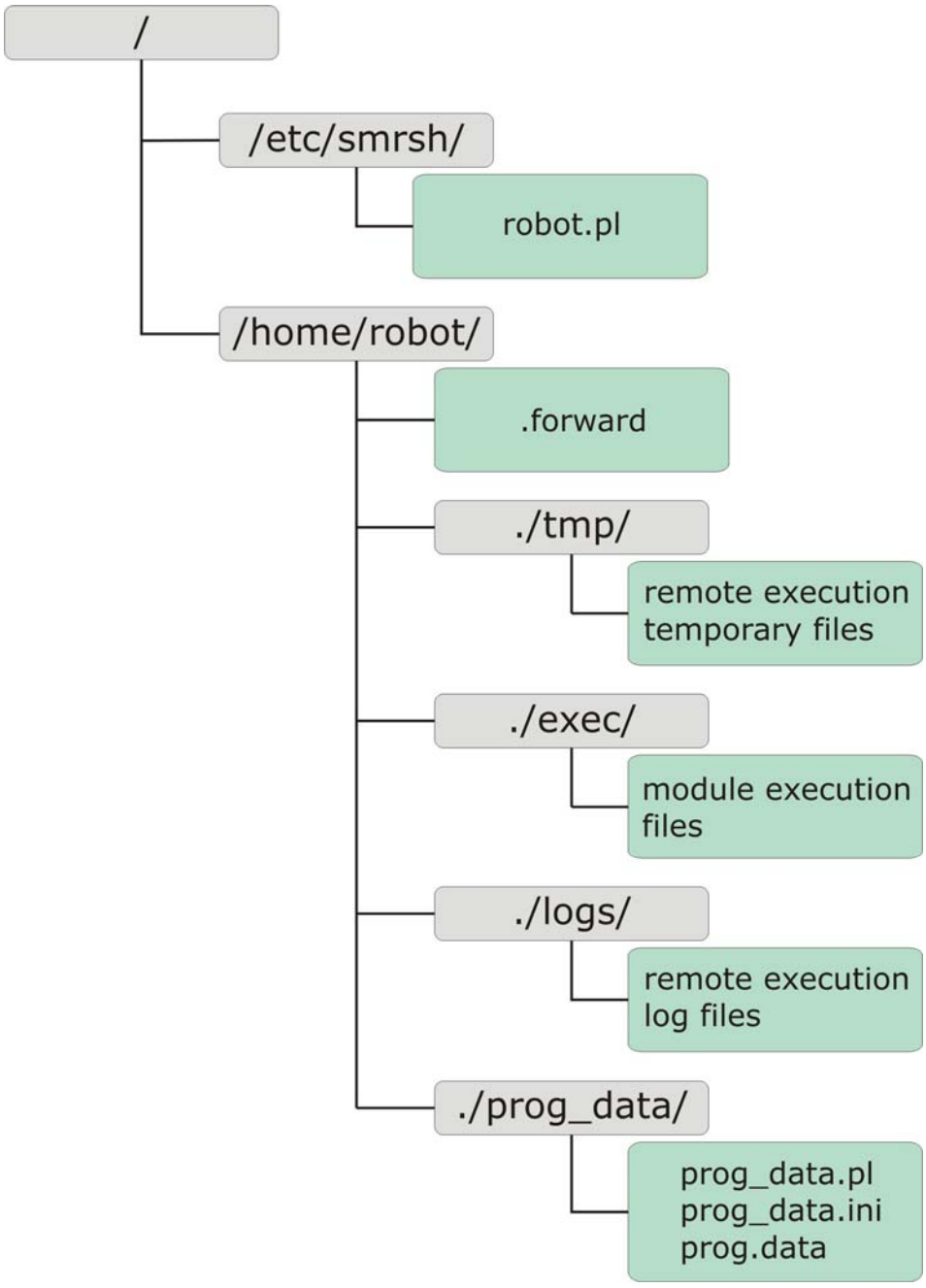
11. Finally, edit the file *robot.pl* that you have previously copied in the */etc/smrsh/* folder. Modify the following section at the beginning of the program, inserting the information on your chosen paths.

```
##############################################################
#
#               configuration
#
##############################################################
#
#         configure these parameters
#

my $log_path     = "/home/robot/logs";
my $exec_path    = "/home/robot/exec";
my $temp_path    = "/home/robot/tmp";
my $data_path    = "/home/robot/prog_data";

my $SMTP_Server   = "your_smtp_server";
my $SMTP_timeout  = 120;

#
#           end of configuration
#
##############################################################
```

If you used the directory proposed in the previous examples, you need only to place the name of your SMPT server in the place of *"your_smtp_server"* (note that the double quotes are necessary).

If you have any installing problem you can mail to valab@valab.det.unifi.it.

Location of required files and directories.



Appendix 1

Sendmail

NAME

sendmail - an electronic mail transport agent

SYNOPSIS

sendmail [*flags*] [*address ...*]
newaliases
mailq [**-v**]
hoststat
purgestat
smtpd

DESCRIPTION

Sendmail sends a message to one or more *recipients,* routing the message over whatever networks are necessary. **Sendmail** does internetwork forwarding as necessary to deliver the message to the correct place.

Sendmail is not intended as a user interface routine; other programs provide user-friendly front ends; **sendmail** is used only to deliver pre-formatted messages.

With no flags, **sendmail** reads its standard input up to an end-of-file or a line consisting only of a single dot and sends a copy of the message found there to all of the addresses listed. It determines the network(s) to use based on the syntax and contents of the addresses.

Local addresses are looked up in a file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash. Beginning with 8.10, the sender is included in any alias expansions, e.g., if `john' sends to `group', and `group' includes `john' in the expansion, then the letter will also be delivered to `john'.

Parameters

-Ac
Use submit.cf even if the operation mode does not indicate an initial mail submission.

-Am
Use sendmail.cf even if the operation mode indicates an initial mail submission.

-B*type*
Set the body type to *type*. Current legal values are 7BIT or 8BITMIME.

-ba
Go into ARPANET mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the ``From:'' and ``Sender:'' fields are examined for the name of the sender.

-bd
Run as a daemon. **Sendmail** will fork and run in background listening on socket 25 for incoming SMTP connections. This is normally run from /etc/rc.

-bD
Same as **-bd** except runs in foreground.

-bh
Print the persistent host status database.

-bH
Purge expired entries from the persistent host status database.

-bi
Initialize the alias database.

-bm
Deliver mail in the usual way (default).

-bp
Print a listing of the queue(s).

-bP
Print number of entries in the queue(s); only available with shared memory support.

-bs
Use the SMTP protocol as described in RFC821 on standard input and output. This flag implies all the operations of the **-ba** flag that are compatible with SMTP.

-bt
Run in address test mode. This mode reads addresses and shows the steps in parsing; it is used for debugging configuration tables.

-bv
Verify names only - do not try to collect or deliver a message. Verify mode is normally used for validating users or mailing lists.

-C*file*
Use alternate configuration file. **Sendmail** refuses to run as root if an alternate configuration file is specified.

-d*X*
Set debugging value to *X*.

-F*fullname*
Set the full name of the sender.

-f*name*
Sets the name of the ``from'' person (i.e., the envelope sender of the mail). This address may also be used in the From: header if that header is missing during initial submission. The envelope sender address is used as the recipient for delivery status notifications and may also appear in a Return-Path: header. **-f** should only be used by ``trusted'' users (normally *root*, *daemon*, and *network*) or if the person you are trying to become is the same as the person you are. Otherwise, an X-Authentication-Warning header will be added to the message.

-G
Relay (gateway) submission of a message, e.g., when **rmail** calls **sendmail .**

-h*N*
Set the hop count to *N*. The hop count is incremented every time the mail is processed. When it reaches a limit, the mail is returned with an error message, the victim of an aliasing loop. If not specified, ``Received:'' lines in the message are counted.

-i
Ignore dots alone on lines by themselves in incoming messages. This should be set if you are reading data from a file.

-L *tag*
Set the identifier used in syslog messages to the supplied *tag*.

-N *dsn*
Set delivery status notification conditions to *dsn*, which can be `never' for no notifications or a comma separated list of the values `failure' to be notified if delivery failed, `delay' to be notified if delivery is delayed, and `success' to be notified when the message is successfully delivered.

-n
Don't do aliasing.

-O *option=value*
Set option *option* to the specified *value*. This form uses long names. See below for more details.

-o*x value*
Set option *x* to the specified *value*. This form uses single character names only. The short names are not described in this manual page; see the *Sendmail Installation and Operation Guide* for details.

-p*protocol*
Set the name of the protocol used to receive the message. This can be a simple protocol name such as ``UUCP'' or a protocol and hostname, such as ``UUCP:ucbvax''.

-q[*time*]
Process saved messages in the queue at given intervals. If *time* is omitted, process the queue once. *Time* is given as a tagged number, with `s' being seconds, `m' being minutes (default), `h' being hours, `d' being days, and `w' being weeks. For example, `-q1h30m' or `-q90m' would both set the timeout to one hour thirty minutes. By default, **sendmail** will run in the background. This option can be used safely with **-bd**.

-qp[*time*]
Similar to **-q***time*, except that instead of periodically forking a child to process the queue, sendmail forks a single persistent child for each queue that alternates between processing the queue and sleeping. The sleep time is given as the argument; it defaults to 1 second. The process will always sleep at least 5 seconds if the queue was empty in the previous queue run.

-qf
Process saved messages in the queue once and do not fork(), but run in the foreground.

-qG name
Process jobs in queue group called *name* only.

-q[*!*]I substr
Limit processed jobs to those containing *substr* as a substring of the queue id or not when *!* is specified.

-q[*!*]R substr
Limit processed jobs to those containing *substr* as a substring of one of the recipients or not when *!* is specified.

-q[*!*]S substr
Limit processed jobs to those containing *substr* as a substring of the sender or not when *!* is specified.

-R *return*
Set the amount of the message to be returned if the message bounces. The *return* parameter can be `full' to return the entire message or `hdrs' to return only the headers. In the latter case also local bounces return only the headers.

-r*name*
An alternate and obsolete form of the **-f** flag.

-t
Read message for recipients. To:, Cc:, and Bcc: lines will be scanned for recipient addresses. The Bcc: line will be deleted before transmission.

-V *envid*
Set the original envelope id. This is propagated across SMTP to servers that support DSNs and is returned in DSN-compliant error messages.

-v
Go into verbose mode. Alias expansions will be announced, etc.

-X *logfile*
Log all traffic in and out of mailers in the indicated log file. This should only be used as a last resort for debugging mailer bugs. It will log a lot of data very quickly.

--
Stop processing command flags and use the rest of the arguments as addresses.

Options

There are also a number of processing options that may be set. Normally these will only be used by a system administrator. Options may be set either on the command line using the **-o** flag (for short names), the **-O** flag (for long names), or in the configuration file. This is a partial list limited to those options that are likely to be useful on the command line and only shows the long names; for a complete list (and details), consult the *Sendmail Installation and Operation Guide*. The options are:

AliasFile=*file*
Use alternate alias file.

HoldExpensive
On mailers that are considered ``expensive'' to connect to, don't initiate immediate connection. This requires queueing.

CheckpointInterval=*N*
Checkpoint the queue file after every *N* successful deliveries (default 10). This avoids excessive duplicate deliveries when sending to long mailing lists interrupted by system crashes.

DeliveryMode=*x*
Set the delivery mode to *x*. Delivery modes are `i' for interactive (synchronous) delivery, `b' for background (asynchronous) delivery, `q' for queue only - i.e., actual delivery is done the next time the queue is run, and `d' for deferred - the same as `q' except that database lookups for maps which have set the -D option (default for the host map) are avoided.

ErrorMode=*x*
Set error processing to mode *x*. Valid modes are `m' to mail back the error message, `w' to ``write'' back the error message (or mail it back if the sender is not logged in), `p' to print the errors on the terminal (default), `q' to throw away error messages (only exit status is returned), and `e' to do special processing for the BerkNet. If the text of the message is not mailed back by modes `m' or `w' and if the sender is local to this machine, a copy of the message is appended to the file *dead.letter* in the sender's home directory.

SaveFromLine
Save UNIX-style From lines at the front of messages.

MaxHopCount=*N*
The maximum number of times a message is allowed to ``hop'' before we decide it is in a loop.

IgnoreDots
Do not take dots on a line by themselves as a message terminator.

SendMimeErrors
Send error messages in MIME format. If not set, the DSN (Delivery Status Notification) SMTP extension is disabled.

ConnectionCacheTimeout=*timeout*
Set connection cache timeout.

ConnectionCacheSize=*N*
Set connection cache size.

LogLevel=*n*
The log level.

MeToo=*False*
Don't send to ``me'' (the sender) if I am in an alias expansion.

CheckAliases
Validate the right hand side of aliases during a newaliases(1) command.

OldStyleHeaders
: If set, this message may have old style headers. If not set, this message is guaranteed to have new style headers (i.e., commas instead of spaces between addresses). If set, an adaptive algorithm is used that will correctly determine the header format in most cases.

QueueDirectory=*queuedir*
: Select the directory in which to queue messages.

StatusFile=*file*
: Save statistics in the named file.

Timeout.queuereturn=*time*
: Set the timeout on undelivered messages in the queue to the specified time. After delivery has failed (e.g., because of a host being down) for this amount of time, failed messages will be returned to the sender. The default is five days.

UserDatabaseSpec=*userdatabase*
: If set, a user database is consulted to get forwarding information. You can consider this an adjunct to the aliasing mechanism, except that the database is intended to be distributed; aliases are local to a particular host. This may not be available if your sendmail does not have the USERDB option compiled in.

ForkEachJob
: Fork each job during queue runs. May be convenient on memory-poor machines.

SevenBitInput
: Strip incoming messages to seven bits.

EightBitMode=*mode*
: Set the handling of eight bit input to seven bit destinations to *mode*: m (mimefy) will convert to seven-bit MIME format, p (pass) will pass it as eight bits (but violates protocols), and s (strict) will bounce the message.

MinQueueAge=*timeout*
: Sets how long a job must ferment in the queue between attempts to send it.

DefaultCharSet=*charset*
: Sets the default character set used to label 8-bit data that is not otherwise labelled.

DialDelay=*sleeptime*
: If opening a connection fails, sleep for *sleeptime* seconds and try again. Useful on dial-on-demand sites.

NoRecipientAction=*action*
: Set the behaviour when there are no recipient headers (To:, Cc: or Bcc:) in the message to *action*: none leaves the message unchanged, add-to adds a To: header with the envelope recipients, add-apparently-to adds an Apparently-To: header with the envelope recipients, add-bcc adds an empty Bcc: header, and add-to-undisclosed adds a header reading `To: undisclosed-recipients:;'.

MaxDaemonChildren=*N*
: Sets the maximum number of children that an incoming SMTP daemon will allow to spawn at any time to *N*.

ConnectionRateThrottle=*N*
: Sets the maximum number of connections per second to the SMTP port to *N*.

In aliases, the first character of a name may be a vertical bar to cause interpretation of the rest of the name as a command to pipe the mail to. It may be necessary to quote the name to keep **sendmail** from suppressing the blanks from between arguments. For example, a common alias is:

```
msgs: "|/usr/bin/msgs -s"
```

Aliases may also have the syntax ``:include:*filename*'' to ask **sendmail** to read the named file for a list of recipients. For example, an alias such as:

poets: ":include:/usr/local/lib/poets.list"

would read */usr/local/lib/poets.list* for the list of addresses making up the group.

Sendmail returns an exit status describing what it did. The codes are defined in <*sysexits.h*>:

EX_OK
: Successful completion on all addresses.

EX_NOUSER
: User name not recognized.

EX_UNAVAILABLE
: Catchall meaning necessary resources were not available.

EX_SYNTAX
: Syntax error in address.

EX_SOFTWARE
: Internal software error, including bad arguments.

EX_OSERR
: Temporary operating system error, such as ``cannot fork''.

EX_NOHOST
: Host name not recognized.

EX_TEMPFAIL
: Message could not be sent immediately, but was queued.

If invoked as **newaliases**, **sendmail** will rebuild the alias database. If invoked as **mailq**, **sendmail** will print the contents of the mail queue. If invoked as **hoststat**, **sendmail** will print the persistent host status database. If invoked as **purgestat**, **sendmail** will purge expired entries from the persistent host status database. If invoked as **smtpd**, **sendmail** will act as a daemon, as if the **-bd** option were specified.

SEE ALSO

http://www.sendmail.org/

Important: Use the *man* command (*% man*) to see how a command is used on your particular computer.

Appendix 2

smrsh

NAME

smrsh - restricted shell for sendmail

SYNOPSIS

smrsh -c command

DESCRIPTION

The *smrsh* program is intended as a replacement for *sh* for use in the ``prog'' mailer in *sendmail* configuration files. It sharply limits the commands that can be run using the ``|program'' syntax of *sendmail* in order to improve the over all security of your system. Briefly, even if a ``bad guy'' can get sendmail to run a program without going through an alias or forward file, *smrsh* limits the set of programs that he or she can execute.

Briefly, *smrsh* limits programs to be in a single directory, by default /etc/smrsh, allowing the system administrator to choose the set of acceptable commands, and to the shell builtin commands ``exec'', ``exit'', and ``echo''. It also rejects any commands with the characters ``', `<', `>', `;', `$', `(', `)', `\r' (carriage return), or `\n' (newline) on the command line to prevent ``end run'' attacks. It allows ``||'' and ``&&'' to enable commands like: ``"|exec /usr/local/bin/procmail -f- /etc/procmailrcs/user || exit 75"''

Initial pathnames on programs are stripped, so forwarding to ``/usr/ucb/vacation'', ``/usr/bin/vacation'', ``/home/server/mydir/bin/vacation'', and ``vacation'' all actually forward to ``/etc/smrsh/vacation''.

System administrators should be conservative about populating the /etc/smrsh directory. Reasonable additions are *vacation*, *procmail*, and the like. No matter how brow-beaten you may be, never include any shell or shell-like program (such as *perl*) in the /etc/smrsh directory. Note that this does not restrict the use of shell or perl scripts in the sm.bin directory (using the ``#!'' syntax); it simply disallows execution of arbitrary programs.

Appendix 3

Secure Sendmail using smrsh

The **smrsh** program is intended as a replacement for /bin/sh in the program mailer definition of Sendmail. It's a restricted shell utility that provides the ability to specify, through the /etc/smrsh directory, an explicit list of executable programs available to Sendmail. To be more accurate, even if somebody with malicious intentions can get Sendmail to run a program without going through an aliases or forward file, smrsh limits the set of programs that he or she can execute. When used in conjunction with Sendmail, smrsh effectively limits Sendmail's scope of program execution to only those programs specified in smrsh's directory. If you have followed what we did above, smrsh program is already compiled and installed on your computer under /usr/sbin/smrsh.

1. The first thing we need to do is to determine the list of commands that **smrsh** should allow Sendmail to run. By default we include, but are not limited to:
    - /bin/mail *-if you have it installed on your system*
    - /usr/bin/procmail *-if you have it installed on your system*

***IMPORTANT**: You should not include interpreter programs* such as sh(1), csh(1), perl(1), uudecode(1) or sed(1) -the stream editor, in your list of acceptable commands.

2. You will next need to populate the /etc/smrsh directory with the programs that are allowable for Sendmail to execute. To prevent duplicate programs, and do a nice job, it is better to establish links to the allowable programs from /etc/smrsh rather than copy programs to this directory. To allow the mail program /bin/mail, use the following commands:

```
[root@deep] /# cd /etc/smrsh
[root@deep ]/smrsh# ln -s /bin/mail mail
```

3. To allow the procmail program /usr/bin/procmail, use the following commands:

```
[root@deep] /# cd /etc/smrsh
[root@deep ]/smrsh# ln -s /usr/bin/procmail procmail
```

4. This will allow the mail and procmail programs to be run from a user's .forward file or an aliases file which uses the program syntax.
    5. **IMPORTANT:** Procmail is required only in Mail Hub Server and not in Local Client Mail Server. If you've configured your system like a Mail Hub Server then make the link with procmail as explained above, if you've configured your system as a Local Client Server then skip the procmail step above.
6. We can now configure Sendmail to use the restricted shell. The program mailer is defined by a single line in the Sendmail configuration file, /etc/mail/sendmail.cf. You must modify this single line Mprog definition in the sendmail.cf file, by replacing the /bin/sh specification with /usr/sbin/smrsh. Edit the sendmail.cf file, **vi** /etc/mail/sendmail.cf and change the line:

Example: sendmail.cf

```
Mprog,  P=/bin/sh, F=lsDFMoqeu9, S=10/30, R=20/40, D=$z:/, T=X-Unix, A=sh -c $u
```

Which should be changed to:

```
Mprog,  P=/usr/sbin/smrsh, F=lsDFMoqeu9, S=10/30, R=20/40, D=$z:/, T=X-Unix, A=sh -c $u
```

7. Now re-start the sendmail process manually with the following command:

```
[root@deep] /# /etc/rc.d/init.d/sendmail restart
```

Instead, use the technique shown above for other /etc/mail/sendmail.cf files in your network like the *one for the nullclient local or neighbor client and servers* that use the null.mc macro configuration file to generate the /etc/mail/sendmail.cf file.