Pp. 1-12 of the *Proceedings of LISA '02: Sixteenth Systems Administration Conference* (Berkeley, CA: USENIX Association, 2002).

Work-Augmented Laziness with the Los Task Request System

Thomas Stepleton – Swarthmore College Computer Society

ABSTRACT

Quotidian system administration is often characterized by the fulfillment of common user requests, especially on sites that serve a variety of needs. User creation, group management, and mail alias maintenance are just three examples of the many repetitive tasks that can crowd the sysadmin's day. Matters worsen when users neglect to provide necessary information for the job. They can grow bleakest, however, at volunteer-run or otherwise loosely-coordinated sites, where sysadmins often collectively hope for someone else to attend to the task.

The Los Task Request System addresses all three problems. It mitigates user vagueness with web forms generated from XML parameter specification files. It skirts sysadmin sloth by requiring one simple review and approval step to set changes into motion. It then saves time by automatically executing commands tailored from user input. Amidst this convenience, cryptographic signatures on Los directives ensure that only administrators can alter the system. Overall, Los aims to make life easier for users and sysadmins by standardizing and streamlining the submission, review, and execution of requests for common system tasks.

Introduction

For over a decade, the volunteer student system administrators of the Swarthmore College Computer Society (SCCS) have provided shell, mail, and web services to hundreds of College-affiliated users. However, a problem arose during the 2001-2002 school year: nobody was volunteering to take care of common system administration requests. The sysadmins had an excuse: most were seniors that year and were confronted with the double whammy of the formidable Swarthmore workload and figuring out what to do after college. Still, the requests kept piling up.

Immediately, the SCCS chose to hire new sysadmins from the freshman and sophomore classes. At the same time, however, an idea began to take form. Instead of having users mail the admins with only vague ideas of what they need to say to get things done, what if a web form could guide them in supplying the necessary information? Then, what if the sysadmins could just direct the data to some handy scripts and have everything taken care of automatically? The notion of turning the e-mail client into a system administration tool was compelling, and through an impossible feat of time management, development of the Los Task Request System began.

From the onset, it became clear that Los would have to satisfy some challenging requirements:

1. It would have to be general enough to handle many different types of system administration tasks.
2. It would have to reduce the time required for common system administration tasks beneath the threshold of the harried student volunteer.
3. It would have to collect and present necessary system configuration information to the user in order to be user-friendly (e.g., no rote memorization of group names).
4. It would have to be secure by design. Only sysadmins should be able to make changes to the system, and integrating new tasks into Los should never compromise its security.

Happily, after months of programming, Los appears to fulfill all of these requirements. Points 1 and 3 were handled by diligent coding of no particular novelty; the approach to points 2 and 4, on the other hand, is Los's most compelling feature.

Los can be characterized as a "semi-automatic" system administration tool. Some system administration tools directly empower the user to make important changes to the system. These "fully automatic" tools are carefully written to resist malicious behavior on behalf of the user; however, since they must have elevated privileges, there's always a slight risk of an exploit. Los is designed so that only the sysadmins can activate the privileged part of the system. A review of the user's input, or "task request," by a responsible human, while relatively brief and unchallenging, is mandatory.

The best way to understand how the Los system works is to follow it as it handles a single task request. This "bird's eye view" will reveal that there are many steps involved in the process. However, it is important to remember that users and sysadmins themselves only see a small and manageable fraction of them for any given request.

The process starts on the Web, where the user makes a selection from a catalog of available automated tasks (Figure 1). This catalog is generated from a collection of *task description files*, which are XML files that contain all the information Los needs to solicit and apply task information from a user.

Using information from the description file for the user's chosen task, the Los web interface retrieves information from the user with a "wizard"-style series of input forms (Figure 2). The description file can invoke sophisticated filters that check the validity of input and solicit corrections (Figure 3).

When the user finishes entering data, Los checks the e-mail address they specified by sending them a verification message. The user visits a web address from the message, and Los sends their input on to the sysadmins. The user is finished and now waits for the task request to be fulfilled.

A few moments later, a sysadmin sees the task request as an XML document attached to an e-mail. Surveying the user's input, the admin decides it is valid and uses a small utility to forward the data to the Los task execution module. The utility cryptographically signs the request with the admin's GNU Privacy

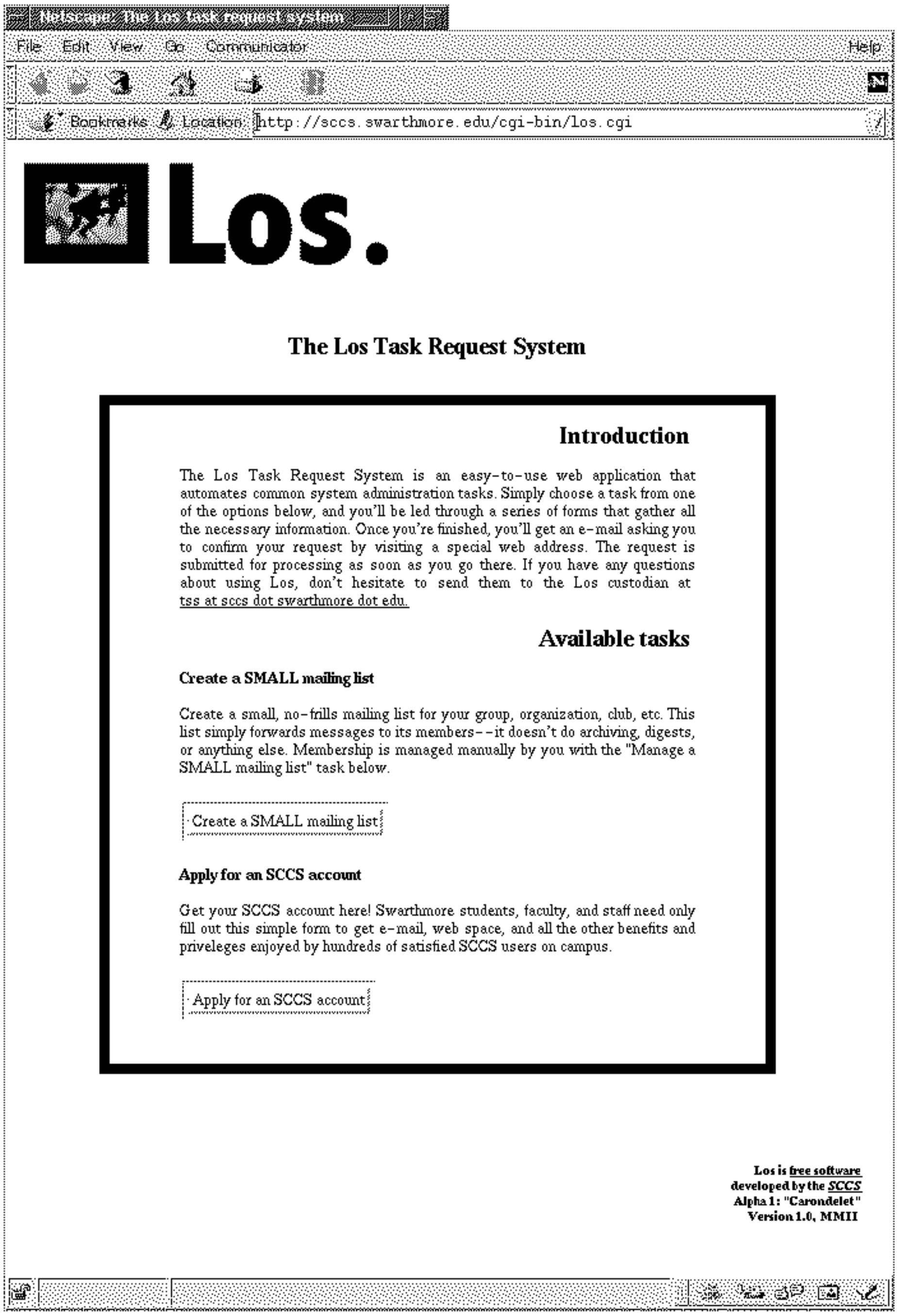

Figure 1: The Los task list. The user begins here.

Guard (GPG) [1] key before sending it. (In the future, the utility will be unnecessary; the sysadmin will simply forward a signed copy of the task request e-mail to the execution module directly.) The sysadmin is finished and waits for an e-mail confirming the execution of the task.

The Los task execution module, having validated the signature on the task request, loads the appropriate task description file and determines what commands it needs to run. It gleans arguments to the specified commands from the task request data, and each command is executed. Finally, Los mails the commands' output to the sysadmins for review.

By now, the SCCS has successfully adapted a number of system administration tasks to this automated paradigm. Users are able to request new accounts, create and manage mail aliases and mailing lists, and allocate and control access to shared student organization webspace through six custom-made Los tasks.

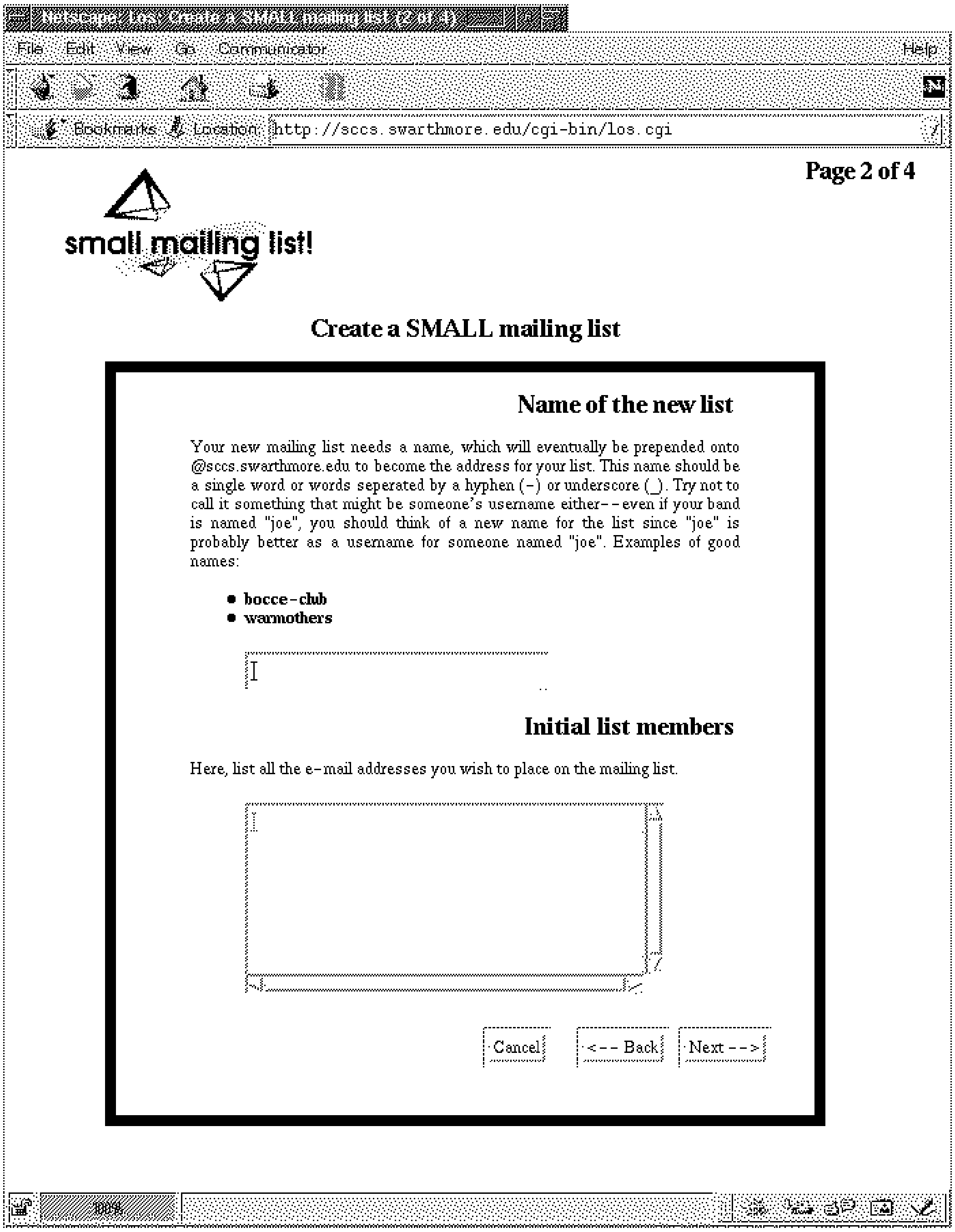

Figure 2: Using a "wizard"-style interface, the user enters data for the task.

Related work

The goals of Los are not especially novel. Several systems that automate or at least accelerate common system administration tasks already exist. These seem to fall into two categories: the fully automatic user-centric systems that require no sysadmin intervention, and systems which are intended to be seen only by the administrators. A few seem to cater to both, depending on their configuration.

A good first place to look for both kinds of software is the Internet hosting business, where the users are owners of particular websites or other Internet resources and the administrator must oversee the servers that host them. Because site owners want to provide the same services on their sites that organizations with dedicated servers can provide, it is often necessary for system administrators to directly configure mail transport agents, FTP daemons, and other systems to their needs. Some solutions to this problem streamline the sysadmin's job: one example software package is ispbs [2], which provides a convenient web interface for administration of many such sites. There are several systems that also have user-centric

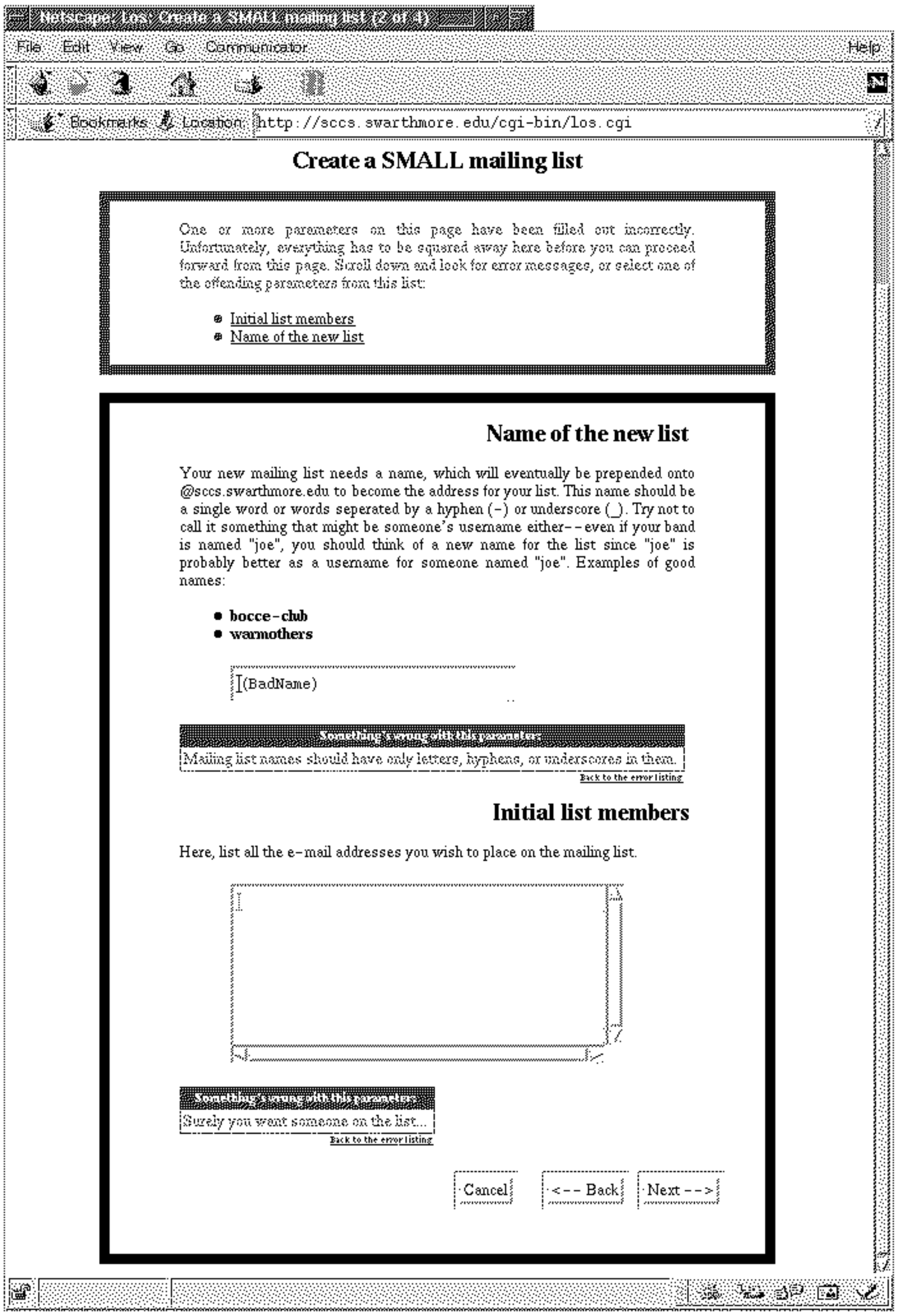

Figure 3: Task description files can invoke filters that check the validity of user input.

capabilities, however, including Account Systems Manager [3] (ASM), and ISPMan [4] which provide web-based configuration interfaces to the user as well. To make system changes, all three eventually require some sort of automated privileged mechanism: ispbs uses a script that is automatically executed as root by cron, ASM executes changes immediately by always running as root, and ISPMan places user requests into an LDAP database which is queried periodically by an execution system.

We find similar software outside of the Internet hosting realm as well. System administration tools that take in data from sysadmins and automatically apply it are well known and include such software as Webmin [5] and Linuxconf [6] Both of these systems provide standardized, extensible means of gathering data from the system administrator and executing the requested changes. Linuxconf can even acquire input through different interfaces, including a web based interface, a native GUI frontend, and a text console interface. These systems also require privileges to work: like ASM, Webmin has a dedicated webserver which runs as root, while Linuxconf in common configurations uses a SUID server program executed by the xinetd Internet super server. Another systems of this sort is the Pelendur account management system [7].

Both Webmin and Linuxconf also have user accessible fully automatic capabilities. While Linuxconf does so by using a built-in per-user privilege system, Webmin employs a separate system called Usermin [8], which also features a dedicated webserver running as root. Other fully automatic systems include Accountworks [9] and Mailman [10]. These systems, which manage user accounts on a corporate network and larger mailing lists respectively, are not as general as those described above. In the case of Mailman, its narrow application focus permits it to be easily isolated from the rest of the system, thus mitigating a great deal of the security risk involved in user-activated system alteration.

The components that make up Los are also not especially new. Any user of an e-mail to FTP interface, the Majordomo [11] mailing list system, or various e-mail based problem tracking systems is familiar with sending commands by e-mail. An add-on to the RT problem tracking system [12] even checks cryptographic signatures on e-mail directives [13]. The XML encoding of user data bears a resemblance to existing XML based RPC mechanisms like SOAP [14]. Finally, the extensible web-based user input system is similar to (but rather more limited than) configurable web-based database frontends like FileMaker [15].

Los Components in Depth

The bird's eye view detailed in the introduction reveals three major stages in the life of a Los task request: creation, review, and execution. Los takes advantage of these divisions with a design that employs a separate mechanism for each stage. While the mechanism for task request review is actually the judgment of a discriminating system administrator, the other two steps are automated by two Perl programs of considerable complexity: the web interface and task execution module mentioned previously. The web interface is a CGI program that resides in any web accessible directory that permits execution of CGI scripts. The task execution module usually resides in a library directory that contains other files necessary for both programs. The task description files, which contain detailed specifications for the data required for a task and the commands for applying it, supply both components with the specific information they need to do their job.

Both programs are designed for version 5.005 and greater of the Perl interpreter running on relatively POSIX-compliant systems. However, they also employ several different Perl modules from the Comprehensive Perl Archive Network (CPAN) [16], which may further restrict their use to the more familiar and modern Unices. A copy the GNU Privacy Guard encryption software must be installed for the Los task execution module to function. At the SCCS, Los was developed and runs on version 2.2 of Debian GNU/Linux. This section will further detail the design, use, and implementation of these important Los components.

Task Request Creation with the Web Interface

The business of getting task request information from the user is conducted by a single large CGI program that creates a series of ''wizard''-like web forms for the user. While one might initially suspect that this consists mainly of presenting questions and HTML form inputs to the user, the job is much more complex for all but the simplest of data collection tasks. Much of the complexity of the Los task creation script is designed to handle the following issues:

1. **Dynamic generation of choices** . In order to keep things simple for the users, it is often necessary for the system to generate a list of possible choices on the fly rather than require the user to remember them and type them explicitly into an input box. A good example of this is a form that allows users to alter membership in a user group that has write access to a web page or other resource. It's much easier for the group members to identify their group name from a listing rather than spell it out on their own; later on, furthermore, the script must be able to list the members of the selected group for alteration.
2. **Dynamic checking of input**. Since no task request is executed without a sysadmin's approval, it's not absolutely necessary for user input to be validated at every step. However, having the system perform checks on its own can relieve the sysadmin of having to incrementally correct the user's choices again and

again. It can also allow the sysadmin to focus on more subtle errors in the input instead of typos. The issues involved in dynamic input checking are similar to those surrounding dynamic choice generation.

3. **State maintenance, revision, and security**. Since the input script gathers information with a series of forms, it is necessary for it to preserve all the information the user has already supplied as it asks for more information with new forms. In the current system, this information is stored on the client side with "hidden" HTML form input elements. However, this requires tamper checking to make certain the user hasn't maliciously altered any of the data stored on their end. In general, judicious management of input is necessary to ensure that data is kept intact through multiple back and forth transactions between client and server.

As mentioned earlier, the Los input script first greets the user with a catalog of available tasks. This simple task is accomplished with a cursory scan of all the task description files for title and summary information and is not especially complicated. More thorough examination of the task description files happens when the user selects a task and begins supplying data. Because the input script maintains all state on the client side, the following steps generally take place on each new page load:

1. The script organizes information it loads from the current task description file.
2. It determines which page of the wizard-style input forms the user has just completed.
3. If the user is advancing to the next page, it checks their new input against the specifications in the task description file. If there is the problem with the input, it prepares to show the last page again with error messages; otherwise, it readies the next screen. If, on the other hand, the user chose to go back to a previous page, the script prepares to go backwards without checking input.
4. The script now displays the new page for the user, a process consisting of generating the HTML form elements that belong on the page and storing all of the data the user has entered so far. Data entered on other pages are cached in hidden HTML form inputs – the rest, if the user has been here before, is stored in the actual form elements shown on the page. The style of the page itself is determined by a collection of templates that can be configured by the administrator.

Each page generated by the input script is described by one of several parameters sections of the task description file. The parameters section consists of multiple parameter entries, which each describe a particular piece of information needed for the task. In addition to providing a short description of the parameter for the user, these entries also specify the proper type of HTML form widget for acquiring the information and a list of tests that validate the user's input. Some parameters, like the user's e-mail address, are required for all tasks; currently, if a task description file omits them, the input script will automatically insert default stand-in parameter entries into its own in-memory representation of the task.

Figure 4 contains a sample parameter entry from a Los task description file. The selector tag invokes a routine in a "standard library" of HTML form elements to generate the simple text input widget needed for this parameter. The following format tags are either Perl-compatible regular expressions (PCREs) or library calls like the selector tag, identified with pcre and filter tags respectively. For specialized applications, admins may create their own collections of selectors and filters if they choose.

Selectors and filters are nothing more than Perl routines, and relatively little effort is made to shield

```
<parameter name="uname" title="Preferred username">
  <description>
    Please choose a new username here. Be sure to specify one that is at
    least three letters long.
  </description>
  <selector name="Los::Selectors::Input" args="size=8,maxlength=8"/>
  <format>
    <pcre>/.../</pcre>
    <description>This username is too short</description>
  </format>
  <format>
    <filter name="Los::Filters::Tolower"/>
    </description></description>
  </format>
  <format inverse="true">
    <filter name="Los::Filters::IsUser"/>
    <description>This username is already taken</description>
  </format>
</parameter>
```

Figure 4: An simplified example of a parameter entry from a Los task description file.

them from the internals of Los. This means that they can actually alter the data submitted by users, a desirable feature in some cases. Because the example parameter entry in Figure 4 is soliciting a username from a new user, it uses a filter called Los::Filters::Tolower to convert the input to lowercase characters. This is necessary for the next filter, which checks whether the username already exists on the system. Some elaborate filters take even more advantage of this freedom: for example, a system of password filters for protecting access to certain Los scripts checks and updates a password database as it processes user input.

Many arguments to selectors and filters can be interpolated, thereby incorporating user input into their operation. This is the basis of the dynamic generation of choice mentioned earlier. Below, an example selector tag generates a textarea for editing group membership:

```
<selector
  name="Los::Selectors::GroupTextarea"
  args="rows=10,cols=10,wrap=off,
  group=~pagename~"/>
```

The name of the group to be edited is stored in the variable pagename, which is named between two tilde characters in the group argument to this selector. Presumably pagename was supplied by the user in an earlier form page; in the script this example was taken from, the user chooses *pagename* from a menu of student organization webpages.

Eventually the script runs out of form pages to show the user; at this point the user has entered all the information mandated by the task description file. The script does one final check of all the user input by checking every format item of every parameter. If all is well, the script creates a file storing the information as a formal Los task request, giving it a unique ID in the process. It e-mails a confirmation URL containing the ID to the user and keeps the file on hand until the URL is visited. Once this occurs, the task request is sent at last to the system administrators for review and approval.

A great deal of effort has gone into making the Los input script flexible enough to handle many different types of information gathering applications. In recent months, the standard selector and filter libraries have grown in their abilities to draw information from files, user and group databases, and other sources required by the Los tasks designed for the SCCS. Nevertheless, it is certain that there will be applications at other sites where these routines will be inadequate. Even the input script itself is limited to proceeding linearly through the lists of parameters in the task description files – although it can modify its questions based on prior user input, it cannot adopt radically different branches of questioning on any basis. Thankfully, because Los's modular design permits other interfaces to create task requests to do what the web interface does, tasks with complex data gathering needs can be handled by custom applications and still work with the rest of the Los system.

Task Request Execution by Mail

Figure 5 shows a complete Los task request, which the system administrators receive as e-mail attachments from the input script. If the request is satisfactory, the sysadmin executes the task request by signing it with their GPG key and sending it on to the task execution script over e-mail. This relatively simple gesture belies the considerable complexity involved in making certain that the task request is trustworthy and avoiding the perils that come with an e-mail activated system with elevated privileges.

Because the formatting of e-mail messages differs between mailers and because the Los executor is designed to diminish security risks by analyzing all aspects of an input e-mail, a special utility program is

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE los_transaction SYSTEM "los_transaction.dtd">

<los_transaction date="Sat, 06 Jul 2002 17:54:49 EST"
                 ip="24.205.87.222"
                 id="1025996089-489825418">
  <taskinfo title="Create a BIG mailing list"
            version="1.0"
            creator="tss"
            taskfile="maillist_big_new.xml"
            md5sum="d64d1c85c8c7f3700826d4eda32d512f" />
  <parameters>
    <parameter name="EMAIL">tss@sccs.swarthmore.edu</parameter>
    <parameter name="password_check">1a@S3d$F</parameter>
    <parameter name="pledge">null</parameter>
    <parameter name="password">1a@S3d$F</parameter>
    <parameter name="FULLNAME">Tom Stepleton</parameter>
    <parameter name="DESCRIPTION">null</parameter>
    <parameter name="listname">test-list</parameter>
  </parameters>
</los_transaction>
```

Figure 5: A sample Los task request – this one requests a new mailing list named test-list.

currently required to generate e-mails for the executor. The sysadmin pipes the task request into the utility, which encodes it in Base64 to preserve its formatting, solicits the admin's GPG passphrase, signs the data with the admin's GPG key, and sends it on to the executor.

The Los executor is currently a SUID root Perl script. This should rouse concern in cautious system administrators, as SUID scripts are widely reputed to be dangerous [17]. However, the Perl interpreter on Unix systems has a special facility for safer execution of SUID scripts: suidperl, which, by automatically imposing a technique known as *taint checking*, requires the programmer to properly shield the actions of the SUID script from externally controlled influences like input and environment [18]. There are further security precautions taken by suidperl to thwart the subversion methods commonly directed at SUID scripts, and modern Unix systems often have mechanisms that prevent the race condition attacks that made all SUID scripts unsafe in past years. Still, some admins may be unwilling to adopt the extra risk that accompanies a new program with root privileges and may conclude that Los is not appropriate for their sites.

When a signed task request is sent to the Los executor by mail, it is actually sent to a dedicated user whose e-mail is redirected by Procmail [19] or an analogous mechanism into the Los execution script. The script itself is owned by root and is otherwise exclusively executable by members of a dedicated group, of which the Los executor user is the only member. Security conscious sysadmins may elect to impose additional constraints of their own on the mechanism that directs e-mail into the Los execution script, such as an independent verification of the cryptographic signature on the task request or a blanket rejection of all messages except those from a few select hosts. Thus, though e-mail may seem like a particularly unprotected mode of transit for the task requests, judicious application of common e-mail utilities like Procmail make it possible to carefully inspect and filter them first.

Once underway, the Los executor first checks the GPG signature on the message. To do this, as it does with any external program call that doesn't require root privileges, the executor spawns a subprocess that adopts the UID of the dedicated Los user, performs the work itself, and reports back to the parent through UNIX pipes. For this particular step, a double function is served, as the Los user also owns the GPG keyring against which the task request signature is validated. For the signature to be approved, the signer's public key must be a trusted key (i.e., it must be locally signed with the Los user's key, requiring the admin to su to the losuser and import and sign their key manually) and must be explicitly mentioned in a file containing a list of keys whose owners have authorization to approve task requests. Unless all of these conditions are met, the Los executor will abort and report the failure to the sysadmins.

The script moves on to carefully decode and parse the Base64 encoded task request, maintaining a healthy paranoia about unexpected input. This done, the script examines the taskinfo tag from the task request to determine which task description file was used to generate it. The title, version, and creator attributes must correspond exactly with the version and creator information specified in the task description file, otherwise the script assumes that two different versions of the file are in use (a situation that might arise if the input script and executor are on different hosts) and aborts. Optionally, the executor can compare the MD5 checksum of its copy of the task file with that of the one used by the input script. This is not enabled by default, however, as some admins may choose to have different task description files for task request creation and execution.

With all its suspicions allayed, the Los executor can turn its attentions at last toward executing the task. There is still one contingency to anticipate, however. It is possible that the same task request might be sent to the executor twice by two sysadmins acting independently. For certain tasks, this could be harmful to the system. The executor prevents this by attempting to deposit the contents of the task request into a file named with the task request's ID in a designated log directory. If the file already exists or if the script is unable to get an exclusive lock on an empty file, the executor presumes that the task has already been executed and aborts. This protective feature also doubles as a convenient logging mechanism.

The Los executor finally spawns a subprocess to execute the task. The commands for task execution appear in a commands block at the end of the task description file, which also specifies the username under which the commands should be run. Immediately the subprocess drops as many privileges as it can and executes each command one by one. The Los executor has a relatively flexible means of interpolating variable names in command arguments. However, it is not as flexible as the shell and is not intended to be. Rather than reply on a sophisticated command line interpreter built into the script, it is expected that administrators will simply pass variables into shell scripts that do most of the work themselves.

Once the subprocess is finished, its output and the output of all the commands it invoked are sent back to the sysadmins. The well-traveled and highly-automated life of the task request is over, and the user is (hopefully) satisfied.

Los In Use at the SCCS

The Swarthmore College Computer Society has prepared a number of tasks for automation by Los. These tasks represent a certain critical intersection between those that are most frequently requested by our users and those that are the most bothersome to take care of. These include the creation of new users and mailing lists, the management of mail aliases, and the creation and management of student organization webpages. These are interesting problems as they require both the input and execution sides of Los to involve themselves deeply in the analysis and modification of different aspects of system configuration.

The Los task description files at the SCCS have been written to thoroughly screen user input for correctness. In cases like mail alias creation, this requires the input script to check whether the new alias name isn't already being used by users, existing mail aliases, or Mailman mailing lists on our system. Organization web page management requires the culling of group membership information as well as password-restricted access. The standard selector and filter libraries handle these jobs capably, though as new needs for Los arise, there will doubtless be a need for more library functionality.

For most of the SCCS tasks, the Los executor simply invokes an external script with the data collected from the user. Often these scripts must modify configuration files like the mail aliases database, a task which requires care even when performed by a human administrator. To make this modification task simpler, Los comes with a utility that allows the scripts to perform the modifications in a ''record-oriented'' manner: within a section of the file delineated by special comments, the utility adds, alters, and removes comment-delimited records of text provided by the scripts.

This utility exhibits a high degree of caution and will fail if the record section and record delimiters are not all well-formed. Similarly, whenever possible, the scripts employ the system file modification tools supplied with our Linux distribution, including useradd and gpasswd for modification of the user and group databases respectively. The SCCS believes that standardized tools are the key to safe automated modification of important system configuration files, and we abide by this in our executor scripts as often as possible.

Creating Los task description files really is programming, and the time it takes depends on how thoroughly the sysadmin wishes to check user input and how difficult it is to safely automate the execution of task requests. The selector and filter lines in parameter entries are comparable to function calls and tend to each take several different arguments. It would not be difficult to make an interface for task description file creation that uses a web browser and dialog boxes to simplify the task; indeed, this might greatly speed the process, as much of the development effort goes into manually creating the XML and remembering the arguments to selectors and filters.

For the moment, setting up Los for a new task can take some time, on the order of several hours for us at the SCCS. Furthermore, if the task requires a custom selector or filter, some rather involved Perl programming may be required, as the interface the Los input script uses to invoke the selectors and filters is complex. This may change in the future, but current efforts are focusing on making the standard libraries more versatile and complete.

For the time being, expeditious programming of Los task description files requires planning beforehand. The admin can work backwards, starting with figuring out how tasks can be executed automatically and then determining exactly how to obtain the information needed for task execution through Los. Once choices about parameter inputs have been made, the admin can start thinking about what checks they wish to apply to the user's input. At last, with all of these things established, the admin can code up the task description file parameter entry by parameter entry. Thankfully, once the task description file has been completed, installing it into the catalog of Los tasks is as simple as dropping it into the same directory as all the other Los tasks. The task appears in the listing, ready to use, the next time the main Los catalog page is loaded.

Los was completed at the SCCS in the final months of the 2001-2002 school year. As such, most students at Swarthmore were too busy to request the tasks Los has been configured to handle. After the end of the semester and up to the time of writing (mid summer), requests have been understandably sporadic. Los has indeed capably handled these requests and has dramatically improved sysadmin response time, usually finishing within a couple hours of the request submission the business that could take up to a week depending on the demands of our courses or the distractions of summer.

However, it has yet to face the normal SCCS request workload or the heavy period that comes at the beginning of the school year. The SCCS fully expects Los to greatly improve our service to the college community under these stresses, and by the time of the 2002 LISA conference we intend to quantitatively demonstrate this improvement.

One thing we are capable of measuring now is the performance of the Los system on the SCCS servers. Currently we're running both the Los input script and the executor on our main login server, a 400 MHz Pentium II-based machine with 380 MB of memory. Because Los is frequently opening files, generating NIS or LDAP queries, or doing whatever it needs to do to to get the information for its selectors and filters, it is not a champion of speed.

The Los task catalog on the SCCS takes just under four seconds to load on the Swarthmore network, with the time mostly occupied by the superficial scan of the six different task description files we use. For some of the more complicated tasks, it can take about the same time to proceed from one wizard screen to the next. Even when the input script is just creating HTML without doing any input checking or complicated widget generation, the overhead of loading and parsing the task description file and otherwise getting things ready can take about a second.

Naturally, the speed of the execution script depends on the particulars of the task being executed. Though the SCCS task description files exhibit considerable complexity when it comes to checking the user's input, the fact is that once the input is collected, there isn't much work to do for our tasks. Typically a few files will be modified and group membership will be altered, and then the task is finished. Our non-scientific gauge of how long a task request takes to execute, which involves approving a task and then enthusiastically mashing the TAB key in the Pine mailer's message index, indicates that most of our tasks take between five and ten seconds to be executed.

Future Directions in Los

After months of development, Los has grown into a system that meets all the goals the SCCS set out for it. It provides a straightforward interface for common system tasks, eliminating the usual e-mail dialog needed to determine exactly what the user wants. It provides an antidote to the ''someone else will do it'' syndrome of the busy volunteer sysadmin by drastically reducing the time it takes to attend to these tasks. However, Los is surely not right yet for everyone. This section lists some possible improvements to Los or similar semi-automatic system administration systems.

More thorough task delegation. For the SCCS, Los abbreviates the time it takes to attend to system tasks enough that it's not necessary to formally assign task requests to system administrators to ensure that someone attends to them. Indeed, this is probably not a good strategy for us, as it's hard to predict when a particular admin has time to attend to the system rather than coursework. However, it might make sense at some sites for task requests to be automatically delegated to members of the administration team. Modifying Los to send task requests to particular administrators would be fairly easy – making a system that carefully manages who was assigned what would be harder.

Accountability through cryptography. At the moment, Los doesn't record who authorized the execution of a task request. Since a cryptographic signature is required for this to happen, a great deal more could be done to indicate incontrovertibly who authorized the execution of a task. This also would demand relatively little modification of Los, though it does demand a secure, external means of logging task requests and signatures.

Use of XML based RPC standards. As hinted in the references section, a Los task request is little more than a remote procedure call. Los happens to use XML for task requests and task description files mostly due to the great amount of support for XML in Perl and elsewhere, and thus relatively little attention was given to modeling Los's data transaction formats after established XML-based standards. However, it may be more beneficial from an integration and versatility standpoint to use one of the standard XML-based RPC message formats such as SOAP [14] or XML-RPC [20]. Los task requests could then conceivably be used with other systems besides the Los executor.

Integration with Linuxconf. As mentioned previously, Linuxconf is a powerful collection of tools designed to automate and provide a straightforward interface for common system administration tasks. Unlike Los, however, Linuxconf is designed for the system administrator and focuses on the kinds of system parameters the user shouldn't necessarily have to deal with (firewall setup, printer configuration, etc.).

At sites with a lot of personal Linux workstations, however, users may legitimately wish to alter these system parameters of desktop machines while administrators might prefer not to give them root access. One solution might be to use Los to collect configuration requests from the user and then to invoke the powerful Linuxconf modules with the Los executor to make the changes.

Linuxconf already does much of what Los does with respect to collecting and applying data, so it may instead make sense to adapt Linuxconf to the semi-automatic approach to task request approval.

Easier review of task requests. Right now the review of Los task requests requires the sysadmin to visually parse XML to determine whether the user's input is appropriate. This is not especially difficult, but it could be streamlined by a program that interpreted Los task requests, combined them with the information in their corresponding task description files, and generated more legible representations of the user's data. Standardized technologies like XSLT [21] could make this a rather straightforward task,

Easier authorization of task requests. One extremely desirable improvement to Los is the elimination of the clumsy approval script. It would be better if the executor were capable of taking signed e-mail forwards from any mail client, determining whether the signature was valid, and executing the task request. This is difficult, however, as it requires careful analysis of the e-mail, and of course the consequences of a misjudgment could be dire. Further complicating matters, some mailers have different behaviors when it comes to signing e-mails with attachments, let alone signing forwards of e-mails with attachments. Still, the benefits of being able to simply forward a task request to the Los executor make this an eminently worthwhile goal.

Novel input methods. The web interface to Los is a fairly satisfactory means of acquiring input from the user. Pains have been taken to make the default template for Los's HTML output attractive in Lynx and other text-based browsers. Still, it might be useful to be able to submit Los task requests from handheld computers, kiosks, embedded specialty systems, or other devices where web browsers are not practical. Unfortunately, this may require a restructuring of Los, as the selector and filter lines in the task request files are tied fairly exclusively to the Web-only standard libraries.

Integration with problem tracking systems. Though not necessary for the SCCS, some sites might benefit from managing Los task requests with a problem-tracking system. Users could check on the status of their task requests, and sysadmins could tell at a glance which tasks were awaiting attention. A history of executed tasks would also be available for later perusal.

Limited experimentation on integrating Los with problem tracking software has already taken place. Because Los uses e-mail as its transaction transport mechanism, the e-mail based GNATS system [22] was an easy choice. It was not difficult to alter the CGI input script and the task request approval script to create and interpret GNATS problem report e-mails. However, there are opportunities for tighter integration. Just as you can now edit a problem report by specifying its category and ID number on the command line (as in edit-pr mycategory 532, a sysadmin should be able to approve a task request in the same fashion with a script that automatically updates the status of the problem report that contains it. This should not be a challenging task.

One issue that remains to be resolved is the encoding of task requests in GNATS problem reports. When the input script sends a task request to the sysadmins, it places it in a Base64 encoded MIME attachment to avoid corruption of the data. Most Los applications don't need this precaution, but the risks of quoted-printable mail encoding, CR to CRLF conversion, and other e-mail mutations make it a prudent one. A default GNATS installation does not deal well with MIME-encoded e-mails, though this issue is being addressed. For the time being, though, another encapsulation mechanism may be necessary.

Use on multiple systems. At the moment, thanks to the stopgap task approval script, Los task requests can only be directed toward a single Los executor, and thus a single computer system. Certainly the script called by the executor could use a tool like Igor [23] to trigger changes on many systems at once. What about a situation where different kinds of task request must be executed on different machines?

When the task approval script is eliminated, the sysadmin will be able to simply direct task requests toward the proper computer by altering the To: filed in their e-mail client. However, because executing a task request on the wrong system could have negative consequences, it might be appropriate to also place extra information in Los task requests that prevent them from being executed on the wrong system. This would not be an especially difficult modification.

Getting Los

In order to encourage widespread use and enthusiastic development, Los has been released under the most recent version of the BSD license. Los can be downloaded from the Free Software Foundation's Savannah development repository at http://savannah.gnu.org/projects/los/ .

Acknowledgments

Deserving recognition are the Swarthmore College Computer Society and the Swarthmore Linux User's Group mailing list members for their resources, support, and encouragement. Advice and instruction from the denizens of #perl on irc.openprojects.net as well as from Frank J. Tobin, the author of the GnuPG::Interface Perl module, are gratefully acknowledged.

Author Information

After getting his start administering the Linux mail and web server at tiny Thomas Jefferson School in St. Louis, Tom Stepleton went on to serve as a sysadmin at the Swarthmore College Computer Society for all four of his undergraduate years. Currently, Tom is a first year doctoral student at the Carnegie Mellon University Robotics Institute, where some systems can physically evade administration. While Tom tends to crank out open source software packages semiannually, Los is his largest to date.

References

[1] GnuPG Team, et al., "GNU Privacy Guard," http:// www.gnupg.org/ .

[2] Host Plus, et al., "ispbs," http://ispbs.hostplus.net/ .

[3] Neikous Software, et al., "Account Systems Manager," http://asm.neikous.com/ .

[4] Ghaffar, Atif, et al., "ISPMan," http://www.ispman.org/ .

[5] Cameron, Jamie, et al., "Webmin," http://www.webmin.com/ .

[6] Solucorp, et al., "Linuxconf," http://www.solucorp.qc.ca/linuxconf/ .

[7] Curtin, Matt, Sandy Farrar, and Tami King, "Pelendur: Steward of the Sysadmin," *Proceedings of the Fourteenth Usenix System Administration Conference*, Dec. 2000.

[8] Cameron, Jamie, et al., "Webmin," http://www.webmin.com/index6.html .

[9] Arnold, Bob, "Users Create Accounts on SQL, Notes, NT, and UNIX," *Proceedings of the Twelfth Usenix Systems Administration Conference*, Dec. 1998.

[10] Viega, John, Barry Warsaw, and Ken Manheimer, "Mailman: The GNU Mailing List Manager."

Proceedings of the Twelfth Usenix Systems Administration Conference, Dec. 1998.

[11] Chapman, D. Brent, "Majordomo: How I Manage 17 Mailing Lists Without Answering "-request" Mail." *Systems Administration (LISA VI) Conference (LISA '98)*, Oct. 1992.

[12] Vincent, Jesse, et al., "RT: Request Tracker," http://www.fsck.com/projects/rt/ .

[13] Vincent, Jesse, "enhanced-mailgate," http://www.fsck.com/pub/rt/contrib/2.0/rt-addons/enhanced-mailgate.README .

[14] W3C, "W3C Recommendation: Simple Object Access Protocol (SOAP) 1.1," http://www.w3.org/TR/SOAP/ .

[15] FileMaker, Inc., "FileMaker: Products: FileMaker Pro 6," http://www.filemaker.com/products/fm-home.html .

[16] *The Comprehensive Perl Archive Network*, http://www.cpan.org/ .

[17] Akin, Thomas, "Dangers of SUID Shell Scripts." *Sys Admin Magazine*, June 2001.

[18] Birznieks, Gunther, "CGI/Perl Taint Mode FAQ," http://gunther.web66.com/FAQS/taintmode.html .

[19] van den Berg, Stephen R., Philip Guenther, et al., "Procmail," http://www.procmail.org/ .

[20] UserLand Software, et al., "XML-RPC," http://www.xml-rpc.org/ .

[21] W3C, "W3C Recommendation: XSL Transformations (XSLT) Version 1.0," http://www.w3.org/TR/xslt/ .

[22] Free Software Foundation, et al., "GNATS," http://www.gnu.org/software/gnats/ .

[23] Pierce, Clinton, "The Igor System Administration Tool," *Proceedings of the Tenth Usenix System Administration Conference*, Sept. 1996.