LPRng – An Enhanced Printer Spooler System

Patrick Powell – San Diego State University, San Diego, CA
Justin Mason – Iona Technologies, Ireland

ABSTRACT

The LPRng software is an enhanced, extended, and portable version of the Berkeley LPR software. While providing the same general functionality, the implementation is completely new and provides support for the following features: lightweight (no databases needed) lpr, lpc, and lprm programs; dynamic redirection of print queues; automatic job holding; highly verbose diagnostics; multiple printers serving a single queue; client programs do not need to run SUID root; greatly enhanced security checks; and a greatly improved permission and authorization mechanism.

Introduction

Print spooler software is one of the most common and heavily used system application programs. While printing may appear to be simple on the surface, in practice it is complicated by the following problems. Each model of printer has a peculiar set of interface and format requirements; this means that the printer software must be highly configurable at the device interface level. Next, multiple users may want to share the same printer; this leads to the need for a spooling system with the associated problems of priority and fair use. Printers are notorious for failing at the most inopportune times; the spooling software needs to report failures and to reconfigure or repair the system in a simple manner. Finally, the software should be portable so that the same software can be used on different systems; in a network based system this introduces the problems of security and authentication.

The LPRng Printer Spooling [Pow95] software is a descendant of the 4.3 BSD Line Printer Spooler Software (LPR), [Cam94] but has totally redesigned and reimplemented. The evolution started in 1986 at the University of Waterloo, where the original 4.3 software was modified to support a variety of new printers. Due to restrictions with the original AT&T and Berkeley software license these modifications could not be distributed. The problems encountered during this process led to the development of the PLP (Public Line Printer) software [Pow95a] and PLP Version 3.0 (PLP3.0) was released in 1988. The PLP software architecture was based on the the original LPR code, but with highly verbose diagnostics and a much more elaborate set of administration functions.

From 1988 to 1994 various sites and administrators modified and extended the PLP3.0 software. The `plp@iona.ie` mailing list was formed to distribute and coordinate these changes, and in 1994 a major programming effort by Justin Mason <jmason@iona.ie> restructured the PLP3.0 code, integrated the majority of extensions, and PLP4.0 was released in 1995.

The complexity and problems with the PLP software have been discussed in the `plp@iona.ie` mailing list as well as various USENIX newsgroups. Given the current network security issues, client/server based applications, and growing administration problems, it was clear that the PLP4.0 software would need extensive revisions. The need for a new version of print spooling software discussed, and there was general agreement on the following design goals.

First, run time diagnostics and detailed error reporting were essential and should be the highest priority. When problems occur users and administrators must quickly diagnose the causes, and obtaining information is essential. Next, the user interface to the printing facilities should change as little as possible. This would allow a gradual evolution from LPR and PLP to the new software with as least surprises to the users as possible. However, the administrative interface could change, and many improvements and changes were suggested. It was essential that the new software be compatible at the network interface level with other implementations of the LPR spooling software. While in 1990 the RFC1179 – Line Printer Daemon Protocol [McL90] documenting the network protocol to be used to transfer print jobs and status information between line printer spooling programs was published, many of the existing implementations do not conform to RFC1179 or have made extensions to the RFC. The existing LPR and PLP software uses a set of *filter* programs to interface to various printers. A major concern of administrators was that these *vintage* filter programs should be usable with the new software. Finally, the long list of security, administration, and networking problems should be eliminated if at all possible.

These considerations led to the design and development of the LPRng software. While it is a totally new design and implementation of spooling software, it uses routines and support code from the Free Software Foundation GNU Project, and is distributed under the GNU Copyleft License. [GNU91] The LPRng software was intentionally designed to use as few non-portable or non-standard Operating System facilities as possible, or to use them in a highly controlled and portable manner. The use of the GNU utilities such as *autoconf* and *Gnumake* allow operating system dependent versions of various support routines to be selected at compile time in an automatic manner.

The following sections discuss the overall architecture of the LPRng software, and then deal with the major components. The emphasis of this discussion are the added functionality or differences of LPRng. The LPRng configuration information, extensions to the printcap database, and changes to the `lpr` and other client programs is discussed. The operation of the job spool queues and the new algorithm used for job printing is then covered, together with a description of the filter interface mechanism. Security and associated problems with SETUID ROOT programs is briefly discussed, and the summary at the end lists some outstanding issues.

LPRng Software Architecture

The LPRng software architecture is shown in Figure 1. While LPRng is similar in structure to the Berkeley LPR software, it differs in many important details. The dashed lines indicated TCP/IP based communication between two programs; solid lines represent access to files or directories. Boxes with dotted outlines represent databases that may be accessed by all programs, either as files or by using network facilities. The user programs such as the print spooler `lpr`, status reporter `lpq`, job remover `lprm`, and control program `lpc` are now client programs which connect to one or more `lpd` server processes using TCP/IP. After validation and authentication the servers carry out requested activities on files and or provide status information. The *configuration* and *printcap* databases provide the information needed by both server and client programs. While clients do not need access to the printcap database, in many cases a *runt* database is useful for providing printer configuration information.

As in the LPR software, the `lpd` server manages one or more spool queues where print jobs are stored. These are are implemented as directories in a file system. A print job consists of a control file, which contains user information and printing options, and data files which contain the actual information to be printed. A spool queue can be a *bounce* or *forwarding* queue, which temporarily stores print jobs before they are transferred to another queue, or a *print* queue which has an associated printer.

Operation of a spool queues is controlled by information in the spool queue printcap entry and the printer control file in the spool directory; individual print job may also have a job control file as well.

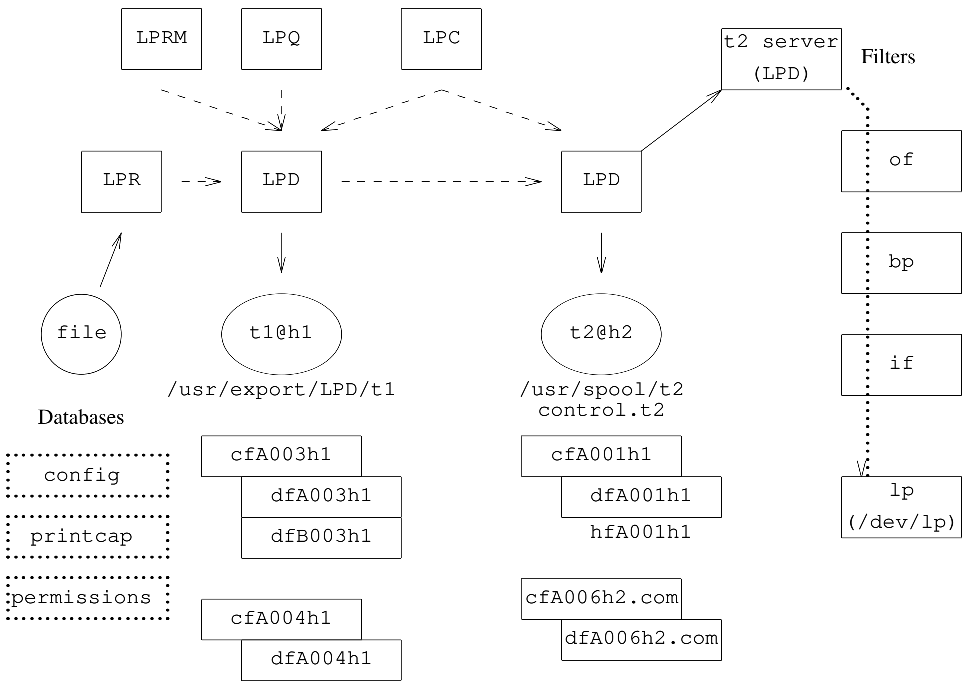

Figure 1: LPRng Spooling Software Architecture

Jobs are submitted to the lpd server by the lpr program which transfers the job over a TCP/IP connection. The lpd server then forwards the job to another server or print it. The lpq program requests and prints job status information, and the lprm program removes jobs from the spool queue. The LPRng software uses a *permissions* database and the printcap information to determine if a user is authorized to use a facility; authorization can be based on originating host, user name, and a variety of other attributes.

After a job is placed in a print queue, lpd creates a server process to manage the printing operations. This server process then creates the necessary filter processes which interface to the printer hardware. The data files are passed through the filters to the actual printer.

Configuration Information

Configuration information is used by both the LPRng clients and the lpd server. The configuration information controls the network behavior of the programs, and provides a set of default for commonly specified system information. Compile time defaults can be overridden by values read from a configuration file, whose format is shown in Figure 2.

In all LPRng database files leading whitespace, blank lines, and lines whose first non-whitespace character is a # are treated as comments and ignored; a \ as the last character of a non-comment line will logically continue this line to the next line, replacing the \ with one or more spaces.

Each line of the configuration file has a configuration variable and its value. The client_configuration_file and server_configuration_file values are used only at startup and initialization, and specify the configuration files for the LPRng client and lpd server programs. Each of the configuration files is read in sequence and variable values are updated as the files are read.

Much of the configuration information provides site dependent information or allows configuration for testing. The default_printer and default_host variables set the default printer and host to be used by client software; the %h and %H strings are replaced with the short or fully qualified domain name of the host on which the software is running. The default_banner_printer sets the default banner printing program to be used by the lpd server; the lockfile and logfile are used by the lpd server to prevent multiple servers from running and to record lpd logging information.

The lpd_port variable specifies the TCP/IP port on which the lpd server listens for client requests. In production versions this is usually 515 (the printer alias in the network service database); by setting it to some other port a test version can be run in parallel with production software.

```
# ENG LPRng Test Configuration
# compile time only:
#client_configuration_file /etc/lpd.conf:/etc/lpd_client.conf
#server_configuration_file /etc/lpd.conf

default_printer          t1
default_host             %H
default_banner_printer   /usr/local/bin/lpbanner
lockfile                 /usr/spool/LPD/lpd.lock
logfile                  /usr/adm/lpd.log
#lpd_port                printer
lpd_port                 4000
originate_port           721 731
user                     daemon
group                    daemon
#printcap_path           /etc/printcap:/usr/etc/printcap
printcap_path            /tmp/LPD/printcap.%H
#printcap_path           |/tmp/LPD/pcserver
#printer_perms_path      /tmp/LPD/printer_perms.%H
#printer_perms_path      /etc/printperm:/usr/etc/printperm
printer_perms_path       /tmp/LPD/printer_perms.%H
#print_perms_path        |/tmp/LPD/permserver
use_info_cache           yes
# include facility
include                  /tmp/LPD/common.conf
```

Figure 2: Configuration Database Format

The originate_port value specifies a range of TCP/IP port numbers for originating connections. RFC1179 specifies that these connections should originate from port 721 to 731 inclusive; in most UNIX environments these are *privileged* ports and cannot be used unless the program's effective UID is ROOT (0). On a UNIX system, if the client software is not SETUID ROOT, then only the ROOT user can successfully bind to a privileged port. See Security Considerations for details on problems this may expose. The user and group entries specify the effective user and group IDs to be used by the lpd server. For this to be effective, the lpd server must be SUID root or be started by a root process; see Security Considerations for details.

The printcap_path and printer_perms specify where database information will be found; this information may need to be read repeatedly by the lpd server. The use_info_cache option allows the server to read the information once at startup and then use a cached copy of this information, as does the inet.d server. If lpd receives a SIGHUP signal it rereads the database information. Finally, it is possible to use the include facility to read additional configuration files. This facility may be removed in later releases of the LPRng software.

Printcap Information

Entries in the printcap database define spool queues and their configuration available to the LPRng software. Figures 3a and 3b show a set of client and server printcap database entries. Leading whitespace, blank lines, and lines whose first character is '#' are ignored. For compatibility with the historical LPR printcap format, \ at the end of a line appends the next line to the current line.

```
# Client Printcap Database
# printer p1@'local host'
p1
# remote printer
p2
  |full|double|rotate
  |twosided|XDR Line Printer
  :lp=p2@host
# remote printer alternative
p3:rp=p3:rm=host
# connect to port 2000
p4:2000%host
# all entry (lpq -a)
all:all=p1,p2,p3
```

Figure 3a: Client Printcap Examples

A printcap entry consists of a primary name followed by an optional set of aliases, followed by an optional set of variable tag names and values. The primary name is the name by which the printer is referred to in error messages and status information. The | separator starts an alias entry and the : separator starts an variable entry; entries extend to the end of line or the next separator character; leading and trailing in each entry whitespace is ignored.

The LPRng client programs need only the lpd server host name and target printer on the server. This can be specified on the command line using the '-Pprinter' or '-Pprinter@host' option; if no default is specified in the configuration information the local host is the default server host. In Figure 3a, the simple printcap entry p1 means printer p1 on the default host; entry p2 has has two aliases, the last of which is really a comment and will be used when displaying status information.

```
# Server/Client Printcap Database
# file: /etc/printcap
# clients see p1 as remote pr
# server use sd tag to get
#  /usr/spool/LPD/p1/printcap
p1
  :cm=Test Printer 1
  :sd=/usr/spool/LPD/p1
  :lp=p1@host
# second printer,
p2
  :sd=/usr/spool/LPD/p2
  :tc=common
# common information
common:
  :lf=log
  :rw
  :of=/tmp/LPD/psof
  :if=/tmp/LPD/psif

# Printer specific information
# used by server,
# file: /usr/spool/LPD/p1/printcap
p1
  # override previous value
  :lp=/dev/ttya
  :lf=log
  :rw
  :of=/tmp/LPD/psof
  :if=/tmp/LPD/psif
# debug
#   :db=9,remote=10
# autohold
#   :ah
```

Figure 3b: Server/Client Printcap Examples

The lp (line printer) tag specifies the printer device or host. The form lp=printer@host is printer on host; the form lp=printer@host%2000 indicates the lpd server is available on port 2000. This last form is useful when running multiple versions of spooler software and when connecting to speical network based printers. A file pathname such as lp=/dev/ttya specifies a printer device to be used by the server; the form `lp=host%2000` indicates port 2000 on host is network based printing device.

More printcap information is needed for the `lpd` server, as is shown in Figure 3b. Spool queues have printcap entries with a `sd` (spool directory) tag. The `tc` tag (recursively) appends a printcap entry to the end of the referencing entry.

The `lpd` server checks to see if a `printcap` file is in the spool directory, and will read the printcap information from this file, overridding existing information. This allows a single *master* printcap database to be used by both clients and servers; the clients ignore the `sd` tags and the server gets printer specific information from the printcap file in the spool directory.

A major administration problem is the distribution of printcap information. One solution is to use a network database such as Sun Microsystems NIS, HESIOD, Sybase, etc. Rather than build in a specific database access method the LPRng software uses the concept of *database filters* to access the information. In Figure 2, the configuration `printcap_path` value `|/tmp/LPD/dbserver` specifies using a filter program to get printcap information.

The filter program is started by the client or server process and a string containing the name of the desired printcap entry is sent to the filter's `stdin` port; the returned printcap information is read from the filter's `stdout` port. By convention, a `all` request returns either all the available printcap entries, or an `all` printcap entry whose `all` tag contains a list of available printers.

The Sun NIS database can be access by using a simple shell script and the `ypmatch` program; HESIOD, DBII, Sybase, and other databases can be supported in the same manner.

Job Submission

The `lpr` client program submits jobs to the `lpd` server by simply using a TCP/IP connection and sending the files to the server. The only information the client needs is the printer and hostname, and can run as a user application.

If the printer output is piped to the `lpr` client, then RFC1179 allows the output to be directly copied from the client to the server by using the `lpr -k` (for seKure) option. While LPRng supports this option, many other LPR server implementations are defective or do not support this capability. This is useful when creating large jobs, or there is are security related problems with creating a temporary file on the client host.

The LPRng clients can run as ordinary user processes; eliminates any problems with unauthorized access to files, as the client has no permission except those of the user.

However, for the `lpr` client to be compatible with *vintage* LPR spooling software (i.e.- SUN

Attribute	Match	Connect	Job Spool	Job Print	LPQ	LPRM	LPC
SERVICE	S	'X'	'R'	'P'	'Q'	'M'	'C'
USER	S		JUSR	JUSR		CUSR	CUSR
HOST	S	RH	JH	JH	RH	JH	JH
IP	IP	RIP	JIP	JIP	RIP	JIP	JIP
PORT	N	PORT	PORT		PORT	PORT	PORT
REMOTEUSER	S		CUSR		CUSR	CUSR	CUSR
REMOTEHOST	S	RH	RH	JH	RH	RH	RH
REMOTEIP	IP	RIP	RIP	JIP	RIP	RIP	RIP
PRINTER	S		PR	PR	PR	CPR	CPR
SAMEHOST			SH			SH	
SAMEUSER			SU			SU	

KEY:

CUSR	user name in connection	
JUSR	user name in control file	
RH	connecting host name	
RIP	connecting host IP	
PORT	connection origination port	
JH	host name in control file	
JIP	IP address of JH	
SA	Same Host	JIP == RIP
SU	Same user	JUSER == CUSER

Match: S = string with glob wild card, IP = IPaddress[/netmask],
N = low[-high] number range; NOT negates the test status

Figure 4: Permission Attributes

Microsystems), it must originate a connection from a *privileged* port. For this reason, when run as a SETUID ROOT program, after making a connection to the the server, the `lpr` client uses *setuid*(2) to drop the root permissions, and operates as an ordinary user program.

Several of the *vintage* `lpr` options such as the `'-s'` (use symbolic links) and `'-r'` options (remove on printing) are not supported; the symbolic link option has no effect as files are transferred directly to the server, and the remove option has caused more than one user to accidentaly delete the files that he wanted printed!

Permissions and Authorization Checking

One of the requirements of any printer spooling system is to deny access to unauthorized users and to record accounting information for authorized users. The LPRng software uses a rather elaborate permissions and authorization mechanism, similar to the ones used by computer network firewalls.

Since all spooling operations are carried out by the `lpd` server, it is the only process that needs to perform permissions checks. Permissions are checked when a connection is made to the server, and before the server performs and action or provides information requested by the various client programs. In addition, the server checks job permissions before it prints a job as well as when the job is submitted. This allows NFS based printer spooler software, which copies control and data files directly to a spool directory, to be used with the LPRng software. See the Security Considerations section for a discussion of problems related to allowing this type of activity.

Each request for service has a set of attributes and values; a list of these attributes is shown in Figure 4. Figure 5 shows a sample permissions database. Each line in the database consists of a match result and a list of attribute names and match patterns. Permissions checking is done by scanning the database in order, checking each line for a match. If all the entries on line match, then the result is the match result for the line or the current default. Note that each entry can have several alternate patterns; these patterns are tried in order until a match is found.

The `default_permission` configuration variable specifies an initial (default) permission line; additional permission databases are specified by the `printer_perms_path` configuration variable. When checking permissions for a spool queue with printcap entry, the `xu` printcap tag provides an additional set of databases to be searched. If no match is found after searching all specified databases then the last specified default permission will be used.

Attributes are treated as string, integer, or IP address values. The string patterns are based on the simple `glob` patterns of the Bourne and C shells, and use case insensitive matching with only the `*` metacharacter. For example, the pattern `A*b` will match `Ab`, and `AthisB`. IP address patterns are an address (ADDR) followed by an optional netmask (NM) which defaults to 255.255.255.255; the match succeeds if (using C language notation) `(IP^ADDR)&NM` is zero. For example, the pattern `130.191.163.0 / 255.255.255.0` matches all of the addresses in the `130.191.163.0` subnet range. Number patters are a low to (optional) high integer range.

The special pattern *char*= `pattern` matches the *char* line in the job control file against pattern. For example, `C=A*,B*,C*` will check the *C* (class) information line for a string starting with A, B, or C. The special pattern *NULL* matches missing or no information; for example the permissions entry

```
# Permissions Database
# Reject connections not in our subnet
REJECT SERVICE=X NOT IP=130.191.0.0/255.255.0.0
# Allow root on trusted hosts to have control access
ACCEPT SERVICE=C HOST=hop.sdsu.edu,skip.sdsu.edu \
    PORT=721-731 USER=root
REJECT SERVICE=C
# do not allow forwarded jobs from anybody but dickory
ALLOW SERVICE=R NOT SAMEHOST HOST=dickory.sdsu.edu
REJECT SERVICE=R NOT SAMEHOST
# Allow PC lab to spool to laserwriter
ACCEPT SERVICE=R,P,Q PRINTER=lw4 HOST=*.eng.sdsu.edu
# Let them remove jobs if from the same host
ACCEPT SERVICE=M PRINTER=lw4 HOST=*.eng.sdsu.edu SH
REJECT HOST=*eng.sdsu.edu
# if no match in other database then you fail
DEFAULT REJECT
```

Figure 5: Sample Permissions Database

`ALLOW SERVICE=R,P USER=NULL,*` allows anonymous job spooling and printing.

Spool Queues and Job Files

The main activity of the `lpd` server is centered on managing print jobs in the spool queues. A print job consists of a control file, containing user and other information, and data files containing the information to be printed. The control file format is specified by `RFC1179`; a sample job control file is shown in Figure 6. Control file names have the format `cfXnnnHOST`, where X is a letter, `nnn` is a 3 digit job number, and `HOST` is a host identifier. Data files names have the format `dfXnnnHOST`, where X is a letter, and `nnn` and `HOST` are identical to the corresponding control file.

```
Htaco.sdsu.edu
Ppapowell
J(stdin)
CA
Lpapowell
Qt1
fdfA917taco.sdsu.edu
N(stdin)
UdfA917taco.sdsu.edu
```

Figure 6: Job Control File

Control file lines starting with an upper case letter provide information and those starting with lower case letters specify a format and a data file to be printed with the format. For example, the `P` (person) and `H` (host) lines give the originating user and host name; the `I` (indent) and `L` (banner name) are used when printing the job.

The LPRng software extends the basic RFC1179 control file entries by adding `Z` (output filters options) and `Q` (original queue). The value of these options are passed to the filters that format and print the data files. For example, Figure 3a shows an example of a printcap entry (p2) with several aliases. The `lpr` command `lpr -Q -Pdouble -Zheavy_paper` will create a control file with the `Qdouble` and `Zheavy_paper` entries and sends it to the p2 printer. The output printing can use the `Q` and `Z` entries to select various paper and format options.

LPD Server Operations

The `lpd` server creates *queue server* process for each spool queue, and then waits for connections from clients. Each time a request arrives the server will create a new process to handle the requests. The `max_servers_active` configuration variable can be used to limit the number of active servers. The queue server process uses the printcap entry information and a set of control files in the spool directory to control its activities and report its actions (Figure 1). In the discussion below, *printer* is stands for the primary printer name; all files are in the spool directory unless otherwise indicated.

The Server lock file (*printer*) is used to ensure that only one server process is active at a time. The spool control file (`control.`*printer*) has the format shown in Figure 7a, and controls one or more of the spool queue related activities. Entries in this file override defaults and values in the printcap database. Note: the information shown in this file may not be present at all times.

The control file `spooling_disabled` and `printing_ disabled` entries disable spooling to the queue and printing from the queue respectively. The `redirect` entry causes the server to transfer all spool jobs to the specified remote printer. When `autohold` is enabled, the server will not process a jobs until it is released by a request from the `lpc` program.

```
printing_disabled 0
spooling_disabled 1
debug 10,remote=5,log=/tmp/log
redirect  p3@mentor
autohold  off
class     A,B
```

Figure 7a: Spool Control File

The `class` entry restricts the printable jobs to the specified class. This facility allows special forms to be mounted on a printer and only jobs which need them to be printed. The special pattern *char*`=patterns` restricts printing to jobs with a control file line starting with *char* which matches pattern. For example, `P=accounting` could be used to restrict printing to jobs from the `accounting` user.

The `debug` entry is a diagnostic and testing aid. The set of options are used used by the server to enable or disable specific testing functions. For example, `10,remote=5,log= /tmp/log` specifies a general debugging level of 10, setting the remote flag to 5, and logging to the `/tmp/log` file.

The `lpc` (line printer control) program is used to request the `lpd` server to change the spool control file values and take other actions, such as starting or stopping server processes. The `lpc` program can also request (brutal) spool server process termination, and (gentle) restarting of spooling activities.

The spool server process scans the spool queue, ordering jobs to be serviced in a first-in, first-out order within priority classes. Class A is the lowest (default) priority, and Z is the highest. When a job is selected for for servicing, the spool server forks a subserver process to carry out the actual work..

The reason for using a subserver process for per job servicing is based on experiences with a variety of UNIX implementations. Some of these implementations have *memory leaks* or *file* *descriptor leaks* associated with various database and networking routines; each time a process uses these routines they open a new file descriptor or allocate some temporary storage. Unfortunately, these descriptors are never closed the descriptors or reclaim the storage. These defective functions are *firewalled* in a subserver process, which only exists while a particular job is processed. Note that the same problems exist in the `lpd` server, which also takes care to isolate these actions in a subserver process.

When a job is selected for service, the subserver process creates a job hold file to record information; job `cfA001mentor` will have hold file `hfA001mentor`. The hold file has the format shown in Figure 7b.

```
active      2743
hold        1
priority    0x873486
remove      1
redirect    p4@mentor
error       Printer timed out
```

Figure 7b: Job Hold File

The `active` entry records the process ID of the subserver process, and indicates that the job is active. A non-zero hold entry indicates that the job is being held by administrative actions; a hold value of 0 allows a job to be printed. The `lpc` *hold* and *release* commands can be used to hold and release jobs.

The `priority` field specifies an additional level of job priority; jobs with non-zero priority fields are serviced before jobs with 0 fields; the `lpc` *topq* command updates the priority value.

The `redirect` entry supplements the spool queue `redirect` information. This entry allows individual jobs to be moved to another spool queue. The `lpc` *move* command updates the redirect value.

The `remove` and `error` entries are used to solve a problem with defective or misconfigured printing software. After a job is serviced its files are removed from the spool directory. However, sometimes due to accident or intent, the files cannot be deleted, resulting in the job being endlessly printed and preventing normal operations. When a job is serviced, the job hold file is created and written in the spool directory; if the hold file cannot be modified the job is not serviced. After the job has been serviced the `remove` field is set to a non-zero value; this prevents the job from being reprinted, and the `error` field records any error conditions that might inhibit retrying servicing the job. This information is displayed by the `lpq` (line printer queue) program. After the job files have been successful removed, the server then removes the job hold file.

A *bounce* queue is used to temporarily hold jobs until they can be forwarded to a remote printer. This is useful when sending jobs to a network printer. The LPRng software `lpr` and `lpd` programs use the same algorithm to check file permissions and accessibility when sending jobs to a remote printer.

Printing Algorithm

On the surface, dealing with the printer hardware should be quite simple: the printer device is opened, the job data files are sent to the device, and the printing device is then closed. The actual algorithm used by the `lpd` server for printing a job is rather complex, in order to deal with the following problems.

1. Each printer usually has specific requirements for connection and initialization, not to mention the actual transmission of data.
2. If the connection to the printer is a serial line, *stty*(1) (or a similar function) must set the speed, format, and other characteristics. When a serial line is closed and reopened the line characteristics may be reset to some default value, requiring the line to be held open throughout the printing process.

3 The effects of the failure printing a job job should be localized to that job.

4 Different types of output such as raster plots, PostScript files, text files, etc., may require different handling when printing. This can be very device specific.

5 Multiple users may use the same printer; jobs need to be carefully separated, banner pages provided, and other administrative functions performed.

6 Administrators have a strong desire to record the printer usage so that users can be billed appropriately.

In order to handle printer specific problems, each printer has a set of filters or support programs which provide support for specific operations. For example the `of` filter will print banners, page separators, and other high level queue control functions. Files whose print format is the (lower case) character `?` will be printed using a `?f` filter; the programs corresponding to each format are found in the printcap file.

The algorithm used by LPRng is shown in Figure 8. It is similar to the original Berkeley algorithm, but not identical. Names such as `'of'` refer to entries in the printcap database and `OF` is a filter process created from the `'of'` information; `OF = filter('of') -> LP` means create the OF filter from the `of` information in the printcap file, and send it output to the `LP` filter or device.

```
LP = open( 'lp' );  // open device
OF = IF = LP;    // set defaults
if( 'of' ) OF = filter( 'of' ) -> LP;
                    // make OF filter
if( accounting at start 'as')
   do accounting;
if( leader on open 'ld' ) `ld` -> OF;
                    // send leader
if( FF on open 'fo' ) `fo` -> OF;
                    // send FF

// check to see if banner required
do_banner =
   (always banner 'ab'
      || (!suppress banner 'sb'
         && control file 'L' ));
if( ! header last 'hl' && do_banner ){
   BP = OF; bnr = null;
   if( banner start 'bs' ) bnr = 'bs'
   else if( banner program 'bp' ) bnr = 'bp'
   if( bnr ){
      BP = filter( bnr ) -> OF;
   }
   short banner info -> BP;
   if( BP != OF ) close( BP );
}
// suspend the OF filter
if( OF != LP ) suspend OF filter;

for each data file df in job do
   // send FF between files of job
   if( !first job && ! suppress FF 'sf' ){
      if( OF != LP ) wake up OF filter;
      'ff' -> OF;
      if( OF != LP ) suspend OF filter;
   }
   // get filter for job
   ?F = LP; // default - no filter
   format = jobformat;
   if( jobformat == 'f' or
      jobformat = 'l' ){
         format = 'f';
   }
   filter = format filter from printcap;
   if( filter ){
      ?F = filter( filter ) -> LP;
   }
   // send data file to printer
   // through filter
   data file -> ?F;
   // kill filter
   if( ?F != LP ) close( ?F )
endfor

// finish printing
if( OF != LP ) wake up OF filter;
if( header last 'hl' && do_banner ){
   if( ! no FF separator 'sf' )
      'ff' -> OF;
   BP = OF; bnr = null;
   if( banner end program 'be' ) bnr = 'be'
   else if( banner program 'bp' ) bnr = 'bp'
   if( bnr ){
      BP = filter( bnr ) -> OF;
   }
   short banner info -> BP;
   if( BP != OF ) close( BP );
}

if( ff on close 'fq' ) 'ff' -> OF;
if( trailer on close 'tr' ) tr -> OF;
if( accounting at end 'ae') do accounting;
if( OF != LP ) close( OF );
close( LP );
```

Figure 8: Printing algorithm used by LPRng

While the algorithm used by LPRng resembles the original Berkeley LPR algorithm, it has subtle differences. Before the job is printed, it is checked for the formats it uses. If there is no filter available for a data file, the job is not printed and only an error message is generated.

The printing device is opened and closed for each print job. This eliminates problems of printer failure when various network and other printers fail such that they will not work correctly until reset by a network reconnection or a device open.

The `as` and `ae` printcap entries specify a filter or format to be used to record accounting information at the beginning or end of a job respectively. For example, for a 230 byte long job spooled to printer `p1` by `john` on `pc1` the entry `as=start $P $u $H $b` will write `start p1 john pc1 230` to the accounting file. The entry `as=/usr/local/psaccnt start` will run the `psaccnt` program and wait for it to terminate. Similar action is taken at the end of a job using the `ae` printcap entry.

Each site usually has different needs for banner printing. LPRng has removed fancy bannner printing from the `lpd` server to a separate program. The `bp` (banner printer) program generates a banner for a job; users can modify the banner without modifying the LPRng software. Banners can be printed at the beginning and end of jobs.

LPRng can use *vintage* filters available for LPR and other spooling systems with a minimum of changes. The section on Filters discusses how they are accommodated.

LPRng supports multiple printers serving a single print queue. The master print queue has a `sv=server1,server2,...` (servers) printcap entry listing the server printer names; server printers have a corresponding `ss=master` (serves) printcap entry. The master spool queue server process creates a subserver process for each slave printer; the subserver processes print all jobs in the server spool queue and then terminate. As each of the subserver processes terminates, the master selects a job from the master spool queue and then creates a new subserver process. This subserver will copy the job to the server spool queue and then process the job. Note that print jobs can be directly spooled to slave spool queues, allowing users to send jobs to a server printer as well as to the master spool queue.

Filters

The LPRng software makes heavy use of filter processes for printing and other operations. A filter specification has the form

| `[ROOT] [-$]` *path optionsP*

Printcap printer filter entries usually drop the `'|'` filter indication. Normally, filters run with EUID and RUID *daemon*; the ROOT keyword runs EUID ROOT. See Security Considerations for details.

The `path` entry specifies the absolute pathname of an executable file and the `options` are a set of options to invoke the filter with. In addition to the user specified options, the LPRng software will append the configuration variable `filter_options` unless suppressed by the `-$` flag.

The options are scanned for variable substitutions indicated by `$` characters. If *key* has a non-zero length string value `X`, then `$`*key* expands to `-`*key*`X`, `$-`*key* expands to `X`, and `$0`*key* to `-`*key* `X`, i.e., a space separating the key and value. For a printer filter, if the data file format is binary `$c` expands `-c`. The substitution formats allow the user to create interfaces to *vintage* printer filters with a minimum of effort; see Figure 9 for an example. As a further aid, The printcap `bkf` (backwards filter) flag appends a list of options which are compatible with most *vintage* printer filters.

In addition to the command line options filters have the `PRINTCAP`, `CONTROL_FILE`, and `DATA_FILE` environment variables set to the printcap information, control file contents, and data file name being printed. This allows filters to use information in the control file or printcap entries with a minimum amount of effort.

By convention filters read input from `stdin`, write to `stdout`, and write errors to `stderr`. The error output is usually directed to the error logging file for the printer. Print filters have their current directory set to the printer spool directory.

Security Considerations

Security considerations were a major factor in the design of the LPRng software. Many of the problems center on the following issues.

1. Users trying to use the printer spooler software to exploit bugs in the operating system and gain root access.
2. Users trying to use the printer spooler software to gain unauthorized access to other users files,
3. Users trying to gain illegal access to printing facilities.
4. Users trying to avoid accounting procedures.
5. Denial of service attacks.

The first issue to be dealt with is the problem of ROOT permissions. All of the client LPRng programs can run as ordinary users; this eliminates a large number of attacks on system security by trying to exploit various defects in the system based on SUID root programs. The LPD server is the only program that absolutely needs to run with real UID (RUID) ROOT as it uses a *privileged* TCP/IP port to listen for incoming connections, and in most UNIX systems *bind*(2) requires EUID ROOT permissions to bind to a privileged port. (It is not recommended that a non-privileged port be used as a trojan horse user program can bind to it and impersonate the LPRng software.) According to RFC1179 a connection to a server must originate from a (privileged) port in the range 721-731.

Given this need for ROOT permissions, the LPRng code goes to extreme lengths to ensure that only the *bind*(2) calls are made with EUID root, and that all other operations are done either as daemon (server) or as user (clients). It is strongly

Filter specification:
```
  path arg1 arg2 $P $w $l $x $y
    $K $L $c $i \
    $Z $C $J $R \
    $0n $0h $F $-a
```
Expanded Specification
```
path arg1 arg2  \
   -PPrinter -wpw -lpl -xpx -ypy \
   -Kcontrolfilename -LLogname -iIndent \
   -ZZoptions -CClass -JJobinfo -RRaccountname \
   -n Person -h Host -Fformat af

Note: pw, pw, etc. are from printcap entries,
  Printer, Logname, etc. are from control file lines,
  other information generated by server.
```
Figure 9: Filter Specification and Expansion

recommended that the `lpd` program not be SUID root, but should started up by the system initialization *rc*(4) scripts or a root user.

It is recommended that all client programs be run as user (non privileged) jobs. Only files accessible to the user will be read or transferred to the server. If a user wants to access a printer that requires privileged ports, it is a simple matter to create a *bounce* queue on a server that will forward a job to the remote system.

The `checkpc` (check printcap) program scans the printcap and permissions databases, spool queues, and checks permissions of files and directories. If run by ROOT with the `-f` (fix) flag set, it will try to change ownerships, create files and/or directories, and remove junk or old job files from spool queues. This program also has some portability tests built into it, and can be used to check that the target system can safely run the LPRng software.

Most efforts to circumvent accounting and permissions checks are based on forging or impersonation of another user or network host. The current version of the LPRng software depends on the various system configuration and database utilities to provide user authentication and system authentication. This is clearly inadequate, and a future release of LPRng will support encryption based authentication; the KERBEROS and the PGP systems are under active study for possible use. The method will be base on using the *filter* mechanism to invoke a set of authentication programs rather than directly incorporating the code into the LPRng software. This allows a variety of mechanisms to be used.

One of the arguments for running client programs SUID ROOT is that they are enabled to connect to the server from a privileged port, and the information provided will be authenticated in some manner by the operating system. Unfortunately, the LPRng software uses various network databases to obtain connecting host information; by attacking the the system by spoofing database (DNS) server activities, it is possible to forge authentication.

The use of NFS exported and mounted spool directories exposes the LPRng software to extreme attack. One of the assumptions made by most spooling systems is that only the *trusted* spooling software or trusted application programs will have write access to the spool directory; when the directory is NFS mounted or exported this may no longer be true. Several spooling systems operate by writing job control and data files into an NFS mounted spool directory. By appropriately forging network identification, credentials, and various RPC calls, attackers can create or modify unprotected files in the spooling directory. The ability to read information in job or other files may also give them the ability to launch other forms of attack. One of the more malicious denial of service attacks is to create a file that cannot be removed or modified; the spooler software may end up repeatedly attempting to print the file, blocking other users from using the spool queue and consuming printer resources.

In order to protect the LPRng software from NFS spoofing based attacks, the printcap `cd=`*directory* entry specifies a separate *control* file directory to be used by `lpd` for all spool queue files except the job and data files. This directory should *not* be NFS mounted or exported, and should reside on the local host file system. This directory should be carefully created so as to be accessible only by user *daemon*. Printcap and other information can be safely placed in this directory as it cannot be modified by NFS operations.

Avoiding printing accounting procedures has long been a tradition at educational institutions; while minor infringements are usually ignored, persistent and blatant offenses are worrisome. In addition, once an individual discovers a method then it apparently is rapidly copied by others, leading to widespread abuse. One difficulty faced by administrators is determining the resources used by a job. As part of the printing algorithm, the LPRng software provides a set of *hooks* to allow the invocation of accounting programs before and after the actual job is printed. For example, most PostScript printers have a *page count* register whose value can be easily read by a simple Postscript Program. By reading this before and after a job the total usage can be calculated.

However, some students have discovered that by aborting a job in the middle of its printing or by printing a job that contains information that causes the printer to hang and not report the total pages used at the end of a job they can avoid the normal accounting procedures. By recording information *before* as well as *after* a job completes such incomplete jobs can be found.

Filters are a major security loophole, as most filters are shell scripts and inherit shell script vulnerabilities. To combat this, the LPRng software defaults to running all filters either as the user or as *daemon,* and provides a predefined and limited set of environment variables. Some network printer filters need to open a privileged port and must have root permissions. This is a serious vulnerability, and the `lp=host%port` printer specification has been provided to ameliorate this problem. It has been recommended that filters run as user *nobody*, restricting capabilities to an even greater exent, and this consideration is under study.

Filters which are actually shell scripts are vulnerable to attacks using metacharacters in option strings. These are passed as options to the filter, and are then rexepanded by the shell. For example, a job spooled by the user named `` `root -rf /` `` (note the backquotes) would have interesting results.

The `vintage` printer filters are particularly vulnerable to attacks of this type. To combat this, the LPRng software ruthlessly purges all non-alphanumberic, whitespace and simple punctuation (minus, period, slash, and comma) characters from filter options. The raw option information is available in the PRINTCAP and CONTROL_FILE environment variables. Administrators would be wise to examine shell based printer filters for similar security loopholes.

Deliberate denial of service attacks are almost impossible to avoid. However, heavy usage of the printer system can produce almost the same symptoms. For example, when a large number of print jobs are queued it is possible to exhaust the spool queue file space. The printcap `mx` (maximum job size) entry specifies the maximum job size (in Kbytes) to be queued and the `mi` (minimum free space) entry specifies the minimum free space (in Kbytes) needed.

Summary and Acknowledgments

The LPRng software continues to evolve as users find problems and develop new printing requirements. One of the areas to be pursued is the use of encryption for end to end authentication of users and print jobs. Another is adding interfaces to other network based spooling systems. Finally, documentation and automated management continues to be pursued.

The network based interfaces for client programs almost trivialize development of user specified GUI systems. PERL scripts and Tkl/Tk based front ends can be developed rapidly and easily.

The development of the PLP and LPRng software would not have been possible without the aid and assistance of literally hundreds of users. The main developer of the software was Patrick Powell `<papowell@sdsu.edu>`, and Justin Mason `<jmason@iona.ie>` generated the PLP4.0 distribution, contributed much of the portability code, and organized the `plp@iona.ie` mailing list. Subscribe by sending email to `plp-request@iona.ie` with the word `subscribe` in the body. Marty Leisner `<leisner@sdsp.mc.xerox.com>`, Ken Lalonde `<ken@cs.toronto.edu>`, and Michael Joosten `<joost@ori.cadlab.de>` performed invaluable portability testing and debugging of the LPRng Alpha Minus release; they discovered and provided fixes for literally hundreds of bugs.

LPRng was based on PLP Release 4.0, to which the following people (in alphabetical order) contributed:

Dave Alden	<alden@math.ohio-state.edu>
Julian Anderson	<jules@comp.vuw.ac.nz>
Jan Barte	<yann@uni-paderborn.de>
Baba Z Buehler	<baba@beckman.uiuc.edu>
Lothar Butsch	<but@unibw-hamburg.de>
David M Clarke	<dmc900@durras.anu.edu.au>
Panos Dimakopoulos	<dimakop@cti.gr>
Angus Duggan	<angus@harlequin.co.uk>
Martin Forssen	<maf@math.chalmers.se>
Michael Haardt	<u31b3hs@POOL.Informatik.RWTH-Aachen.DE>
Eric C Hagberg	<hagberg@mail.med.cornell.edu>
Paul Haldane	<Paul.Haldane@edinburgh.ac.uk>
George Harrach	<ghharrac@ouray.Denver.Colorado.EDU>
Stefano Ianigro	<w_stef@unibw-hamburg.de>
Helmut Jarausch	<jarausch@igpm.igpm.rwth-aachen.de>
Michael Joosten	<joost@ori.cadlab.de>
Stuart Kemp	<stuart@cs.jcu.edu.au>
Hendrik Klompmaker	<Hendrik.Klompmaker@Beheer.zod.wau.nl>
Rick Martin	<rickm@cs.umb.edu>
Todd C. Miller	<Todd.Miller@cs.colorado.edu>
Corey Minyard	<minyard@wf-rch.cirr.com>
Dorab Patel	<dorab@twinsun.com>
Ed Santiago	<esm@lanl.gov>
Bjarne Steinsbo	<bjarne@hsr.no>
Harlan Stenn	<harlan@landmark.com>
Julian Turnbull	<jst@dcs.edinburgh.ac.uk>
Bertrand Wallrich	<Bertrand.Wallrich@loria.fr>
Greg Wohletz	<greg@cs.unlv.edu>

Author Information

Patrick Powell <papowell@sdsu.edu> is faculty in the Dept. of Computer and Electrical Engineering at San Diego State University, San Diego CA 92182, where he teaches Computer Networks, Real Time Systems, and Distributed Computing.

Justin Mason is a sysadmin for IONA, Corp. in Ireland, where he administers the various UNIX systems and fixes broken printer software.

References

Pow95. Patrick A. Powell, *LPRng – Enanced Printer Spooler Software Reference Manual,* Dept. of Electrical and Computer Engineering, San Diego State University, San Diego, CA 92182, 1995. FTP://ftp.iona.ie/pub/LPRng/, FTP://dickory.sdsu.edu/pub/LPRng/

Cam94. Ralph Campbell, ''4.3BSD Line Printer Spooler Manual,'' 4.4 Berkeley Software Distribution, Computer Systems Research Group, U.C. Berkeley, Berkeley CA, 1994. USENIX Association and O'Reilly & Associates, Inc.

Pow95a. Patrick A. Powell, ''PLP – The Public Line Printer Spooler Reference Manual,'' PLP 4.0 Software Distribution, 1995. FTP://ftp.iona.ie/pub/plp-4.0

McL90. Leo J. McLaughlin III, *RFC1179 Line Printer Daemon Protocol,* Internet Advisory Board, 1990.

GNU91. GNU, *GNU General Public License,* Free Software Foundation, Inc., 675 Mass. Ave. Cambridge, MA 02139, 1991.