The following paper was originally presented at the
Ninth System Administration Conference (LISA '95)
Monterey, California, September 18-22, 1995

SPM: System for Password Management

Michael A. Cooper
University of Southern California

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

SPM: System for Password Management

Michael A. Cooper – University of Southern California

ABSTRACT

Most UNIX operating systems are not delivered with adequate facilities to address password management in medium to large, heterogeneous environments. The System for Password Management (SPM, pronounced *spam*) provides multiple levels of user verification beyond the normal password verification procedure, both local and centralized password database management, a consistent command interface across multiple platforms and multiple password database types, fast and efficient updates to large NIS *passwd* databases, proactive password checking, and password aging. SPM is intended to replace the **passwd**, **yppasswd**, **nispasswd**, **chfn**, **chsh**, **ypchfn**, **ypchsh**, and **rpc.yppasswdd** programs, although it can run without replacing these programs.

Introduction

The ever increasing number of UNIX users and rapidly expanding popularity of direct Internet access is quickly increasing the number of novice computer users. Subsequently, there is also an ever expanding number of people trying to subvert computer security systems. The large number of novice computer users provides more opportunities than ever for intruders to illegally access computer systems as well as greatly increasing the demand on system support staffs.

The tools provided by most UNIX vendors usually are not well designed to handle this situation in large, heterogeneous environments. Most UNIX systems provide inconsistent command interfaces, in some cases none at all, to change a user's password, full name (GECOS) information, or login shell. Sun's Solaris 2.4 operating system, for example, requires each user to explicitly know where their password information is stored and run the appropriate command (i.e., **passwd** for /etc/passwd, **ypppasswd** for NIS, and **nispasswd** for NIS+) Furthermore, Solaris 2.4 does not even provide a facility for user's to change their NIS or NIS+ full name (GECOS) field or login shell.

Most UNIX systems also do not provide any form of proactive password checking. This can lead to users using very weak passwords which are vulnerable to being compromised by software such as *crack*.

When user's identities are verified for purposes of changing their password, most systems only require the user to enter their current password. If an intruder has compromised an account's password, then the usual methods for maintaining password security, such as password aging, will fail to deny access to the intruder. The ability to query the user for multiple types of information known only to the user and the password system (such as the user's SSN, mother's maiden name, etc.) normally will prevent intruders from maintaining access to compromised accounts.

There are also some serious deficiencies in how changes to the NIS passwd database are performed. One of the most severe is that only one change to the NIS *passwd* file can be made at a time. While the NIS *passwd* file is being changed, all other update attempts are refused. On a master NIS server with a large number of passwd entries (e.g., more than 1000), it can take a small, but substantial amount of time to re-write the passwd file. If a large number of users, such as a class of new users, attempts to change their passwords at about the same time, most users will be unable to because of this single-step update mechanism.

SPM was designed to address these deficiencies. It currently supports */etc* style files and NIS/YP on SunOS 4.1.x (Solaris 1.x), SunOS 5.x (Solaris 2.x), and AIX 3.2.x. The code is very portable to new UNIX operating systems and support for additional password facilities can be added in a very straightforward manner.

The USC Environment

In order to provide more insight into the the design and implementation of SPM, it is useful to describe the environment at the University of Southern California (USC) that spawned SPM.

There are approximately 8,000 nodes attached to USC networks. Of that number, about 1,300 are UNIX machines, 2,000 are Macintosh machines, and 3,800 are PC class machines.

Computing at USC is separated into two main types – Academic Computing (student computing, research computing, basic computing, etc.) and Administrative Computing (financial records, student records, etc.). University Computing Services (UCS) is charged with providing support for Academic

Computing directly to researchers and students, providing support for computing resources owned by departments, and support of the campus networks and Internet connections.

UCS supports a number of specific computing facilities:

- The Research Computing Facility (RCF) consists of Sun, SGI, Convex, and IBM compute servers and is dedicated to providing a high-end environment for performing research.
- The Student Computing Facility (SCF) consists of about 10 Sun servers, and several hundred Sun workstations for use by graduate and undergraduate students for class related work.
- The Basic Computing Facility (BCF) consists of a single Sun server dedicated to providing free access to email, USENET News, and WWW to any USC faculty, staff, or student.

Most hosts supported by UCS are configured to use NIS[1] (formerly YP). While most hosts are split into small, independent NIS domains, there is one large NIS domain that that currently has over 20,000 *passwd* entries and over 1,700 *group* entries. (Most of the users in this domain are part of our Student Computing Facility.) Needless to say, this has allowed us to experience some of the major flaws in both the NIS design and implementation discussed in this paper.

Vendor Provided Interfaces

Before describing SPM in detail, it is useful to be familiar with the general functionality provided by most vendor's to allow user's to change their password information. Since it is not the intent of this paper to provide an in-depth analysis of such interfaces, most attention will be on generalities with a few examples.

Description of /etc files

The most basic UNIX password database provided by virtually all UNIX vendors is the **/etc/passwd** file. This file normally contains seven fields, each separated by a semi-colon:

- **username** A user's login name.
- **password** A user's encrypted password.
- **uid** A numeric user identification number.
- **gid** A numeric identification number specifying the user's primary group.
- **fullname** A text field contain the user's full name and/or other descriptive information. This field is sometimes referred to as the *GECOS* field.
- **home** The user's home directory.
- **shell** The user's login shell.

On most systems, there is a **passwd** command that a user can run to change the **password**, **fullname**, and **shell** fields of their password entry. For a user to change their **password** in **/etc/passwd** they would normally run

```
% passwd
```

The **passwd** program will prompt the user for their current password and confirm that it matches what's currently set in **/etc/passwd**. The user is then prompted for a new password and asked to re-enter the new password to confirm that it matches what the user typed the first time. Most **passwd** implementations perform a few basic evaluations of the new password to test for acceptability. Usually this only involves checking to insure the password is of a minimum length.

Once a new password is confirmed with the user, the **passwd** program will then lock the **/etc/passwd** file, usually by creating a **/etc/ptmp** lock file, then it will update **/etc/passwd** itself. Usually this is done by writing out a new version of **/etc/passwd** into the **/etc/ptmp** lock file, then installing **/etc/ptmp** as **/etc/passwd**.

On some newer systems there exists an auxiliary password database file, which is often called a *shadow* password file and is usually named **/etc/shadow**, that contains encrypted passwords as well as password aging information. The *shadow* password file is normally readable only by "root". This prevents normal users from accessing all the encrypted password fields and running a program such as **crack** to guess user passwords. On such systems, the **password** field in **/etc/passwd** will only contain something like "x" to indicate the encrypted password is really located in **/etc/shadow**.

Description of NIS/YP

The Network Information Service (NIS, formerly called Yellow Pages) developed by Sun Microsystems and licensed to most major UNIX vendors, provides distributed access to to various system databases, such as **automount** maps, **group**, **passwd**, **hosts**, and others. The NIS protocols are based on a client/server ONC RPC architecture.

Hosts utilizing NIS are grouped into NIS domains. Each NIS domain has a single NIS master server and may optionally have a number of NIS slave servers. The NIS master contains the source of all databases. The source databases are ASCII files that are normally identical to their /etc equivalents and, in fact, are often maintained and built directly from their standard /etc locations. For instance, on most systems the **/etc/passwd** file is actually the default location of the NIS **passwd** database source file on the NIS master.

NIS databases, or *maps* as they are usually referred to, are stored in *ndbm*(3) files. Changes to NIS maps are made by modifying the ASCII source file for a particular map using an editor such as *vi*(1) or by using a third-party program of some type or in

[1] We hope to move to NIS+ as soon as our internally developed user account management system is updated.

the case of the **passwd** map, through the **rpc.yppasswdd** server. Once the source file is updated, the *ndbm*(3) files are updated by running *make*(1) in the **/var/yp** directory. The *make*(1) rebuilds the affected *ndbm*(3) databases from scratch – there is no partial update mechanism provided in NIS. Once the *ndbm*(3) databases are updated, the *ypserv*(8) process on the NIS master (the local machine where the database was just updated) is notified of the update. Any NIS slave servers that are configured are notified that a map has been updated. When the *ypserv*(8) process on each NIS slave server is notified of an update, they consult with the NIS master server to check to see if the master has a newer version of the map, and if so, transfers the entire map to local disk storage.

The **passwd** database in NIS has a special set of protocols and servers who provide a mechanism to update NIS **passwd** data. When a NIS user wants to change their password, they run **yppasswd**[2] The user is prompted in a fashion very similar to that described in the previous section regarding normal **/etc/passwd** configurations. However, instead of reading and writing password information from **/etc/passwd** directly, the **yppasswd** command exchanges data with the **rpc.yppasswdd** server via ONC RPC on the NIS master host. The **rpc.yppasswdd** reads and writes the NIS **passwd** source file, which is usually either **/var/yp/src/passwd** or **/etc/passwd.** Changes to the NIS **passwd** file are done by creating a lock file (such as **/var/yp/src/ptmp** or **/etc/ptmp**), then creating a new **passwd** file with the requested changes, installing the new **passwd** file, and then removing the lock file. Once the **passwd** file is updated, **rpc.yppasswdd** will by default run a *make*(1) to update the **passwd** NIS map as previously described. For systems with medium to large numbers of users, it is a common practice to disable this feature and to run the NIS *make*(1) via a **cron**8 job once per hour or so.

Most NIS systems are configured by default such that when one user changes their **passwd** information, nobody else can change their own password information until the **passwd** map has been updated and pushed out to all NIS slave servers. This can take anywhere from a few seconds to several hours depending on the size and configuration of a given NIS domain. It is possible to modify the behavior of the standard **rpc.yppasswdd** and how the NIS Makefile **/var/yp/Makefile** operates such that users are only blocked from **passwd** changes while the actual NIS **passwd** source file is being updated. However, this can take from a few seconds to several minutes depending on how many users are in a given NIS domain. If several dozen users try to change their passwords at once (which often happens to user's who are taking a class on UNIX for the first time), most will be prevented from doing so due to the single threaded behavior of the NIS **passwd** database.

[2]Some systems provide a **passwd** program that has the ability to handle NIS password changes.

Description of NIS+

The Network Information Service Plus (NIS+) was created by Sun Microsystems as a replacement for NIS. NIS+ was designed to address many of the problems found in NIS. It specifically is intended to support larger, more diverse configurations as well as providing a much higher level of security and a much improved administration interface. It can be configured to support older NIS clients, but at the cost of being only as secure as NIS allows.

Like NIS, NIS+ has a client/server architecture utilizing ONC RPC as its communications protocol. Hosts configured for NIS+ are grouped into NIS+ domains. Unlike NIS, each NIS+ domain can be part of an tree structured enterprise namespace similar to the Domain Name System (DNS).

Each NIS+ domain has a primary server and may optionally have any number of secondary servers. Each NIS+ server runs the *nisd*(1m) program which provides service to NIS+ clients. Updates to NIS+ databases can be accomplished through a command line interface, a GUI interface, and through the NIS+ C API. Updates can be made to any server, either the primary or any secondary. Changes are automatically propagated to all NIS+ servers for a given domain. Unlike NIS, updates are made by propagating the changed information and not an entire database.

NIS+ databases are stored on disk in binary format and then loaded into memory by *nisd*(1m) at startup. Locking is done on an individual record basis and not on a whole database basis. This means that NIS+ servers can accept multiple changes to the **passwd** database simultaneously.

Description of SPM

SPM was designed to address the deficiencies in operation and functionality of password management found in most UNIX implementations. It is intended to provide a consistent interface to password management to users and system administrators on an enterprise-wide scale. It normally is used to replace vendor supplied programs such as **passwd**, **yppasswd**, **nispasswd**, **chsh**, **ypchsh**, **chfn**, **ypchfn**, and **rpc.yppasswdd**. SPM is a client/server based system that utilizes ONC RPC for communication.

Consistent User Interface

The SPM user interface presents a consistent look and feel across multiple versions of UNIX and multiple types of password facilities such as **/etc** files and **NIS.** This allows users to not have to focus on the details of how to change their password information on different UNIX systems. A nice

benefit of this is that users tend to spend a little more energy on what they are doing (choosing a secure password) instead of how they are doing it.

Enhancements for Large NIS Domains

In an environment where there are a large number of users in a single NIS domain, updates to the NIS **passwd** database can take a significant amount of time. The issue of users being locked out of making changes whenever any one user makes a change is also a major problem. To address these issues, SPM supports the ability to take password changes and quickly store them as transactions which are later processed asynchronously. This allows multiple users to change their password information simultaneously.

Security Features

There are a number of features of SPM that address different security issues.

User Verification

SPM supports the ability to query the user for multiple types of verification information beyond simply asking for a user's current password. This allows system administrators to have more confidence in the identity of a user whenever a user is forced to change their password.

SPM allows the system administrator to specify via the **VerifyList** variable in **spm.conf** which, if any, additional types of verification information are required. The system administrator can also specify that all or only a certain number of the verification information be verified correctly with the user.

The information used for verification is located in *Personal Data* databases on either or both the local host where the user runs the **spm** command and on a remote host specified by the system administrator. See the section on **Databases** for a description of the contents of the *Personal Data* database.

Password History Checking

SPM supports the ability to keep a history of each user's passwords. Each time a user changes their password, the *Password History* database is checked to see if the new password has been used before by that user. If it has been, the password is rejected.

When a successful password has been chosen, it is stored in its encrypted form in the *Password History* database. The system administrator specifies how many passwords per user are kept in the history database.

Password Checking

In addition to password history checking, SPM provides an extensive set of password checking procedures. When a user changes their password, the password is subjected to a series of rules to test how vulnerable it is to common algorithms used to guess passwords. The rules are partially based on most realistic checks performed by the **crack** software. The rules include checks against the user's username, full name field, a large dictionary, minimum length, character type mix, and other assorted rules. Passwords that fail any of these tests are rejected and the user is prompted to try another password.

Password Aging

Password aging[3] is the ability to specify various parameters about passwords on a per user basis. The parameters control such things as how often a user can or must change their password.

Security of SPM

A number of decisions were made during the initial design of SPM that impact the overall security of the SPM system itself. The basic thinking was that the design be as secure as possible, but not so secure that the time to implementation was severely compromised. What resulted is a system that is no less secure than the standard NIS configuration, but not as secure as we would like. We felt we did not have the time, or the will to implement a fully secure system that utilized some form of full encryption or public key access control.

The work being done to implement standard forms of encryption at the packet and RPC level also lead us to believe that our time is better spent concentrating on other aspects of security and functionality. One such possibility is Sun's *Secure RPC* [1] which allows the use of DES or Kerberos authentication and encryption in ONC RPC based applications. Unfortunately, *Secure RPC* is not yet widely available so SPM does not support it. A future version of SPM may incorporate *Secure RPC* depending on how available it becomes in the future.

The standard NIS configuration utilizing the ONC RPC based **rpc.yppasswdd** server is what we considered the weakest form of security in standard password management systems available today. When using NIS, the client (**yppasswd**) and the server (**rpc.yppasswdd**) exchange requests using un-encrypted ONC RPC calls. The client program sends the user's current (old) password to the server in un-encrypted, clear-text format in an RPC call. Changes to a user's password entry are sent by sending a **struct passwd** structure. This means that when changing a user's password, the old password is sent clear text and the new password is sent in its encrypted format. Anybody watching the network packets for such a session would see the user's old password in plain text and the user's new password in encrypted form. This would allow them to gather encrypted passwords on which they could use password guessing software such as **crack.**

[3]While SPM supports a *Password Aging* database, password aging support itself is not yet fully integrated into SPM.

In SPM, a similar approach to NIS is taken. The user's old password is sent plain text to the server (**spmd**) for verification. This is necessary in order to verify the user's password history and also prevents a rogue client from guessing passwords by querying **spmd.**

When changing password information in SPM, only the information being changed is sent. That is, if a user is changing their *login shell* then only the new *login shell* data, along with some verification data, is sent to the server. In NIS, the entire **passwd** structure is sent. This can allow a packet sniffer to obtain more data about a user than is possible in SPM.

In SPM, special care is given to the handling of *Personal Data* information. When a user changes their password, the **spm** program sends a query to the **spmd** program on the *Personal Data* server host. The server (**spmd**) responds by returning a code indicating no records for the specified user exist, or returns a *Personal Data* (**struct PDinfo**) structure that has a single "x" in every field for which information exists. The client then queries the user for each bit of information available and then sends the user-supplied information to the server for verification. While this information does travel plain-text to the server, this process does not allow a rogue client to download data from the *Personal Data* database.

The **spmd** program also limits access to itself to specific hosts and networks. The variables **SPMaccess** and **PDaccess** in the **spm.conf** file control this access. The **PDaccess** variable controls what hosts and networks can make *Personal Data* queries. The **SPMaccess** variable controls what hosts and networks can query all the SPM databases except for the *Personal Data* database.

Databases

SPM maintains and utilizes a number of databases. All databases are stored in *ndbm*(3) format in subdirectories under **/var/spm**. Database files are named by domain name and are located in subdirectories A domain name is defined by whatever naming facility is in use on a SPM client. Usually this is your **NIS** or **NIS+** domain name. On hosts that don't run a naming service, the default domain name is **local**.

An example configuration would be a site with a *sales* domain and an *eng* domain, the data for the two will be stored separately for each type of database. In this case, there would be the following database files:

```
/var/yp/age/sales.{dir,pag}
/var/yp/age/eng.{dir,pag}
/var/yp/person/sales.{dir,pag}
/var/yp/person/eng.{dir,pag}
/var/yp/pwdhist/sales.{dir,pag}
/var/yp/pwdhist/eng.{dir,pag}
```

This feature permits a single **spmd** server to support multiple domains.

Database Administration

The **spmadm** command is the primary tool available to system administrators to administer the SPM databases. This tool provides the ability to directly read and update all SPM databases. It can be used to lookup and delete individual records, as well as retrieve and store records in "raw" format. An administrator can use **spmadm** with the **-set** *key=value* option to add or modify a specific record. The **-write** option can be used to initially load a SPM database, such as the *Personal Data* database.

Personal Data

The *Personal Data* database contains information used by SPM to verify a user's identity. The type of data intended for this database are bits of knowledge, that when used together with each other, provide a higher level of confidence that a user is who they claim to be.

The following fields are currently supported:

- **SSN** The user's Social Security Number or other type of identification.
- **DOB** The user's Date Of Birth.
- **MomName** The maiden name of the user's mother.
- **HomeZIP** The user's Home ZIP code.

Each record is keyed by a username.

The information is stored as text fields in *ndbm*(3) databases usually located under **/var/spm/person**. Sites may choose to store whatever ASCII data they wish in any of these fields. However, there is no support at this time to customize the prompts presented to users when asked for each bit of information. Modifying the source code is the only means by which to do this, though this can be done fairly easily.

Password History

The *Password History* database contains a record of passwords that users have used before. The information is stored as text fields in *ndbm*(3) databases usually located under **/var/spm/pwdhist**.

Each password is stored in its original encrypted form. Each record is keyed by username. The data field for each record contains a colon separated list of previously used passwords in most-recently-used to least-recently-used order.

The system administrator specifies, through a configuration file, the maximum number, 10 by default, of passwords to maintain in each entry. When this limit is reached, the oldest password for a given entry is deleted.

Password Aging

The *Password Aging* database contains records that allow sites to specify certain policies regarding password changes.

Figure 1: SPM Component Overview

The following records are currently supported:

- **ModTime** Last time a password was modified.
- **MinChg** The minimum number of days before a user can change their password again.
- **MaxValid** The maximum number of days a password is valid for before a user must change their password.
- **WarnTime** The number of days a user is warned to change their password before some type of action is taken once **MaxValid** days is reached.
- **InActive** The number of days an account can be unused before a user must change their password.
- **ExpireDate** The absolute date that the account expires.

Architecture

The SPM software consists of a client and a server which use ONC RPC as a communications protocol and several stand-alone programs that use direct file I/O. Figure 1 provides an overview on how the various parts of SPM interact.

The **spmd** program is an RPC-based server that processes requests from the **spm** client program. It processes requests to access multiple password facility databases (e.g. **/etc/passwd** and **NIS**) as well as requests for the *Password Aging*, *Password History*, and *Personal Data* (verification) databases.

The **spm** program is the interface between end users and the **spmd** server. It allows users to change their password, full name (GECOS) field, and login shell. Normally the **passwd**, **yppasswd**, **nispasswd**, **chfn**, **ypchfn**, **chsh**, and **ypchsh** programs are replaced by symbolic links that point to **spm**.

The **spmtrans** program is used to process transactions created by **spmd** if the system administrator has enabled transaction logging. It uses direct file I/O to access transactions, each of which is stored in a separate file on the system where **spmd** and **spmtrans** are run. Usually **spmtrans** is run periodically (such as once per hour) via the *cron*(8) facility on an NIS master.

The **spmadm** program is used to administer the *Personal Data*, *Password Aging*, and *Password History* databases. Its purpose is to provide a system administrator a means by which data in the various SPM database can be loaded, viewed, modified, and deleted. It uses direct *ndbm*(3) file I/O on a specified SPM database.

Configuration

The **spm**, **spmadm**, **spmd**, and **spmtrans** programs all support the ability to configure SPM variables via the command line and via the **spm.conf** configuration file. Command line options override any value specified in **spm.conf**.

All SPM programs utilize a single **spm.conf** configuration file on each host. This file allows the system administrator to configure such things as SPM database locations, servers, and access controls, password facility files, transaction logging flags, and verification information flags.

On startup, each program searches for this file in a compile-time specified set of locations. Usually the locations are as follows:

```
/var/local/conf/spm.conf
/usr/lsd/conf/spm.conf
```

The first file found is parsed. See the *spm.conf*(5) man page in **Appendix A** for further details.

The **spm** program will do one of three things – change a password, a full name, or a login shell – depending on how it is invoked. The program's behavior is specified either by a command line option or by the name of the program it was invoked as. **Table 1** shows the matrix of what triggers each type of behavior.

Table 1: Behavior of **spm**

Command Line	Program Name	Behavior
–chpw	**passwd** **nispasswd** **yppasswd**	Change Password
–chfn	**chfn** **ypchfn**	Change Full Name
–chsh	**chsh** **ypchsh**	Change Login Shell

User Examples

The **spm** program is the primary end-user interface to SPM. It can be used to change a user's password, full name (GECOS), and login shell. It is designed to act like most UNIX **passwd**, **chsh**, and **chfn** commands whenever possible.

In the following example, a user named **smith** is changing his password. The system that **smith** is logged into is configured to run NIS/YP. The **spmd** server on the NIS/YP master is configured to do transaction logging and the SPM *personal data* server has **smith's** social security number on file. Here's what the session would look like:

```
alcor.usc.edu(1): passwd
Changing nis password for smith on yp1.usc.edu . . .
Current Password:Go4Sleep
Please enter your Social Security Number: 123456789
New Password:$iAmHome
Retype new Password:$iAmHome
Your request to change your password information has
been queued for later processing.
This change may take up to two hours to take effect.
alcor.usc.edu(1):
```

The message about being "queued for later processing" is triggered by **spmd** on **yp1.usc.edu** being configured to do transaction logging. The message that "This change may take up to two hours to take effect" is triggered by **spmd** on **yp1.usc.edu** also being a NIS master and configured to not push out a new **passwd** map itself.

In our next example, our test user **smith** is changing his office phone number and the comments field of his full name information on the same host once again:

```
alcor.usc.edu(3): chfn
Changing nis finger (GECOS) for smith on yp1.usc.edu . . .
Current Password:$iAmHome

Default values are printed inside of '[]'.
To accept the default, press the <RETURN> key.
To specify a blank entry, type the word 'none'.

Full name (e.g. 'John W. Smith') [John Smith]: RETURN
Office Location (e.g. 'SAL 125') [UCC 209]: RETURN
Office Phone Number (e.g. '740-2957') [02957]: 5551234
Home Phone Number (e.g. '213-777-3456') [none]: RETURN
Comments (e.g. 'My student account') [Student]: PhD Student

          Full Name: John Smith
     Office Location: UCC 209
Office Phone Number: 5551234
   Home Phone Number:
           Comments: PhD Student

Is this information correct [Yes]? RETURN
Your request to change your finger (GECOS) information
has been queued for later processing.
This change may take up to two hours to take effect.
alcor.usc.edu(4):
```

In our final example, our restless user **smith** is now going to change his login shell from **/usr/usc/bin/tcsh** to **/usr/bin/csh:**

```
alcor.usc.edu(7): chsh
Changing nis login shell for mtest on yp1.usc.edu . . .
Current Password:$iAmHome

Here is a list of shells you may choose from:
        /bin/sh
        /bin/csh
        /usr/bin/sh
        /usr/bin/csh
        /usr/usc/bin/bash
        /usr/usc/bin/tcsh
        /usr/usc/etc/admshells/newpwd
        /usr/usc/etc/admshells/ftp-only
        /usr/usc/etc/pwd/forcepwd
        /usr/usc/etc/ftp-only
        /bin/ksh
        /bin/rksh

Old Shell: /usr/usc/bin/tcsh
New shell: /usr/bin/csh

You have selected "/usr/bin/csh" as your new shell.
Is this information correct [Yes]? RETURN
Your request to change your login shell information
has been queued for later processing.
This change may take up to two hours to take effect.
alcor.usc.edu(8):
```

Operation

The **spmd** program is invoked by *inetd*(8) the first time a request for SPM service is received. Before each RPC request is processed, a check is performed to see if the client is authorized to make such a request. Access is controlled by use of the **SPMaccess** and **PDaccess** variables as described in the *spm.conf*(5) man page and in the **Security of SPM** section.

The **spm** command is invoked by a user directly (on the command line) or indirectly (another program such as *login*(1m) runs it). Upon startup, the program first determines what information should be changed as described in the **Configuration** section. Next **spm** determines which *Password Facility* (i.e **/etc files, NIS, NISplus**) to use by searching all supported *Password Facilities*[4] and using the first facility the user's *username* appears in. A command line option also allows a user to override this behavior and specify which specific *Password Facility* they wish to use.

The search is done by calling *Password Facility* specific lookup routines. For instance, the NIS function that is used is **NIScheck()**. It performs a *yp_match*(3N) library call to lookup a given user in NIS.

The **spm** program next determines what *domain name* to use in future requests. This name is also obtained in a facility specific manner. In the case of NIS, the NIS *domain name* is used. In the case of **/etc files** being the facility, the predefined domain name **local** is used.

Next, the **spm** program determines the name of the host that is the *SPMserver* (sometimes referred to as the *Facility Server*) for the *Password Facility* that has been selected. This host is where **spm** expects to contact the **spmd** server. This is also done through use of a facility specific routine. In the case of NIS, the *yp_master*(3N) library function is used and returns the name of the NIS master host for the local hosts' NIS domain. In the case of the **/etc** facility, the name of the local host is returned.

The **spm** program next will setup an RPC connection to the **spmd** program on the *SPMserver*. The *SPMserver* is then pinged by sending an RPC **NULLPROC** request to **spmd** to determine if the server is available. If the ping fails, an error is displayed and the program exits. This is done so that a user is not prompted for lots of information only to be told the server is "unavailable".

The user is next prompted to enter their **Current Password**. Once the user does so, the password is sent off to the *SPMserver* for verification. The **spmd** program on the *SPMserver*

[4] Which facilities are supported and the order the facilities are searched is determined at compile-time.

checks the password against the user's current password in the *Password Facility* specified in the request by **spm**. If that fails, the password is then checked against the password of "root" (on the *SPMserver*). If the password was successfully matched against the password for "root", a special flag is set. The results of these comparisons are then returned to the **spm** client. If the results were successful **spm** continues on, otherwise an error is displayed and the program exits.

If the user is changing their *full name* or *login shell* entries, they are next prompted for the new information as shown in the **User Examples** section. Default values are taken from their current entries in the *Password Facility* being changed.

If the user is changing their *password* and the user did not supply the "root" password previously, the **spm** program will try to verify the user's identity using the information found in the *Personal Data* database on the *Personal Data* server (PDserver). The *PDserver* is a host running **spmd** that is either specified by the system administrator in the **spm.conf** file or else is the same as the *SPMserver*. A query is first sent to the *PDserver* to determine what types of *Personal Data* are available for the specified user. The user is then prompted to enter each type of information the *PDserver* indicated it had on file. Each type of information is checked, as it is supplied by the user, by sending it to the *PDserver* for verification. If the information is incorrect, the user is told so and prompted to try again. When the information is correctly supplied, **spm** prompts the user for the next type of information and the process continues. The user is not allowed to proceed until all verification information is correctly supplied. The user will be asked a maximum number of times for a given type of information. If they cannot correctly supply the information within that maximum, an error is returned and the program exits.

Once a user successfully completes the PD verification process they are prompted to enter a **New Password**. The **New Password** is then subjected to a series of tests that are described in the **Password Checking** section. The new password is then sent to the *SPMserver* to be checked in the *Password History* database to see if the user has used this password previously. Upon successful completion of this stage, the user is then prompted to **Retype New Password**. This password is compared against the first one the user supplied to insure that they both match.

Once the new *password*, *full name* or *login shell* information is successfully entered, the **spm** program sends a change request with the new information to the *SPMserver*. The *SPMserver* then applies the changes immediately or the changes are logged for later application if **Transaction Logging** is enabled.

If the user has changed their *password*, the *SPMserver* also adds the user's old *password* to the *Password History* database and also updates, or creates if needed, the user's *Password Aging* database entry.

If the facility being changed is **NIS** and the system administrator has specified a value for the **NISPostCmd** variable in the **spm.conf** file (usually the value is "**cd /var/yp && make passwd**"), then that command is then run to push out the changes to all NIS servers.

Transaction Logging

The *Transaction Logging* feature in SPM allows updates to the NIS *passwd* database to be quickly accepted by **spmd** and batched up for later processing. This allows multiple users to simultaneously submit *passwd* change requests without being denied service because the NIS *passwd* database is locked by another user as is the case in a standard NIS environment.

The sysadmin enables *Transaction Logging* via the **TransFacList** variable in the **spm.conf** file. This variable contains a list of *Password Facilities* which should use *Transaction Logging*.

Transactions are created by the **spmd** server and later processed by the **spmtrans** program which is usually run once per hour via the *cron*(8) facility. If *Transaction Logging* is enabled when **spmd** receives a change request, the request is stored in its own file in a *Password Facility* specific directory under **/var/spm/transactions**, i.e., the directory used for the NIS facility is **/var/spm/transactions/nis**.

The transaction file name is in the format:

Format.ID.Revision

The *Format* portion of the name indicates what format the contents of the file are in. Currently, only the **spmtr1** format is used. The *ID.Revision* parts form a unique and ascending number such that new transactions have a greater numeric *ID.Revision* value than older transactions. This is done so that **spmtrans** knows the order in which to process the transactions. The *ID* portion is created using the value returned by the *time*(2) function. The *Revision* portion is a counter that starts at "0" and is used to help generate a unique file name.

Once the filename is generated, a call to *open*(2) with the **O_CREAT** (create file) and **O_EXCL** (exclusive file creation) flags specified. If the *open*(2) fails due to an **EEXISTS** (file exists) error, another call to *time*(2) is made, the *Revision* number is incremented, and another *open*(2) call is made. This process repeats itself until the exclusive file creation succeeds or a maximum threshold value (specified at compile-time) is reached.

Once the transaction file is successfully created, the file is locked and the transaction data is written in the following form:

UserName:*Type*:*OldValue*:*NewValue*

The *UserName* field indicates which *username* in the password database the transaction should be applied to. The *Type* field indicates what type of information the transaction applies to. Currently, the valid names are **pwd**, **shell**, and **gecos**. The *OldValue* field is used to indicate what the contents of the database was when the transaction was created. If this field does not match what is in the database when the transaction is processed, the transaction is discarded. The *NewValue* field contains the information that should be placed into the database.

Once **spmd** writes the data to the transaction file, the file is unlocked and closed. A message is logged via the *syslog*(3) facility detailing the change and a another message is returned to the client (**spm**) stating that the request "has been queued for later processing".

The transaction files are processed when **spmtrans** is run (usually via the *cron*(8) facility). Upon startup, **spmtrans** searches for and reads a **spm.conf** configuration file. The **TransFacList** variable is used to determine which *Password Facilities* should be checked for transactions that need to be processed. A directory search for transaction files (as described previously) is performed for each *Password Facility* transaction directory (**/var/spm/transactions/***facility*). Each transaction file that is found is locked and then read into memory. The transactions are sorted in memory by *username*, in newest to oldest order, which is determined by the numeric value of the transaction's filename.

Once all transaction files are read, all transactions for each type of possible change operation (**PWD, GECOS, SHELL**) are processed in one database update operation, if possible. e.g., The **/var/yp/src/passwd** file would be written once with all **PWD** (password) changes, then it would be written again with all **GECOS** changes (if any).

During the processing of transactions, the current value found in the facility database is compared against the *OldValue* field in the transaction. If the two fields do not match, the transaction is considered out-of-date and discarded. Since the transactions are processed in newest to oldest order, this eliminates having to do multiple updates to the same entry and insures only the most recent change is applied.

Once the transactions are all processed, the transaction files are unlocked and each transaction that was successfully applied is deleted.

Performance

Probably the most important aspect of performance for a password management system is how quickly a user can change their password. While there is not much empirical data available, some rough comparisons are still possible. During peek usage at USC, a class of 30 users in a single NIS domain with 20,133 users running **spmd**[5] with *Transaction Logging* enabled can change their passwords simultaneously with only about a 15 second wait once the change request is sent to **spmd**. The same requests take another 15 seconds or so to be applied to the NIS *passwd* source file by **spmtrans**. In a standard NIS environment with this same scenario, only one user at a time would be able to change their password.

Related Work

SPM is not the first attempt to supplement or replace vendor password systems. A number of public domain programs has been available for several years. Some comparisons to some of these programs are made in the following sections.

Comparison to npasswd

The **npasswd** [2] program has been distributed since about 1989. Like SPM, **npasswd** is intended to replace a vendor supplied **passwd** program. It supports SunOS 4.x **/etc/passwd** and NIS/YP password facilities.

The main feature this program supports is the ability to check a user's new password to "disallow simple-minded passwords" in a manner similar to SPM's pro-active password checking. It also allows a system administrator to use a configuration file to specify different policies and parameters that affect its password checking routines.

Unlike SPM, however, **npasswd** does not support the ability to change a user's *full name (GECOS)* or *login shell* fields. There is also no facility for handling password history or additional user verification procedures.

Comparison to passwd+

The **passwd+** [3] program has been in existence in various forms since at least 1992. Like **npasswd** it supports extensive pro-active password checking as its main feature. It also has an even more extensive configuration file that allows a system administrator to finely tune password checking policy and local environment characteristics. It does include support for changing *full name* and *login shell* information. However, it appears to only support **/etc/passwd** facilities, though a more current version of the software may support NIS. It also does not support SPM features such as password history, password aging, and additional user verification.

Current Status

SPM has been in campus-wide use at USC since August, 1994. It currently supports SunOS

[5]The server running **spmd** in this case is a heavily loaded Sun SPARCserver 490 with 128MB of memory running SunOS 4.1.3.

4.1.x, SunOS 5.x (Solaris 2.x), and AIX 3.2.5 operating systems. The */etc* file password files and NIS/YP are the currently supported *Password Facilities*. Support for HP/UX and IRIX are planned in the near future. There is some support for NIS+ in place, but that code has not been tested yet.

Future Directions

The issue of RPC encryption remains an important item awaiting a good solution. It is hoped that something such as Sun's *Secure RPC* software or DCE Security Services becomes widely available.

The verification database should be abstracted to allow sites to specify arbitrary types of verification information and the labels presented to users for that information. For example, not every site has SSN data. Some sites might have "employee numbers" they want to use instead. A system administrator should be able to configure SPM to prompt for **Employee Number:** instead of for a **Social Security Number:**.

It might also be useful to add the ability for the system administrator to specify different types of options for the pro-active password testing. This would require a bit of an overhaul of the password checking routines, but should not be a major problem.

Conclusions

Since SPM's initial deployment at USC, it has eliminated major problems with inconsistent user interfaces, bugs in vendor provided programs, enhanced security, and allowed us to continue to grow and better support a large NIS environment. Its portable design and implementation allow us to support multiple platforms and password facilities both now and into the future.

Author Information

Michael Cooper has been working on UNIX systems for 12 years. He has worked in the Research, Development, and Systems Group of University Computing Services at the University of Southern California since 1985 and has also been an independent UNIX consultant since 1988. Reach him via U.S. Mail at: University Computing Services University of Southern California Los Angeles, California, 90089-0251. Reach him electronically at mcooper@usc.edu .

Availability

SPM will be available for FTP from usc.edu:/pub/spm by September 1, 1995.

References

1. ''Secure RPC,'' *Solaris 2.4 System Administrator Answerbook*, pp. 46-48.
2. Clyde Hoover, *README for npasswd 1.6*, March 1990.
3. Matt Bishop, *README for passwd+*, June 2, 1992.