The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

The Design and Implementation of a Network Account Management System

J. Archer Harris, James Madison University
Gregory Gingerich, Bell Atlantic

The Design and Implementation of a Network Account Management System

J. Archer Harris – James Madison University
Gregory Gingerich – Bell Atlantic

ABSTRACT

In many heterogeneous UNIX networks, users require accounts with attributes that vary by machine. This creates a complex management job. This paper reports on a software system for creating and maintaining network accounts.

Commercial and publicly available tools are briefly examined. Disadvantages of commercial tools are found to be high expense, difficulty of modification, and uncertainty of availability on future platforms. NIS and NIS+ are found to lack sufficient flexibility. Kerberos is determined to be an authentication system, not a user management tool.

The Network Account Management System (NAMS) is described. It addresses two key requirements. First, that users be able to have a single login across machines on a network, creating the abstraction of a ''network account.'' Second, that these accounts allow for attributes such as home directory, expiration date, and login shell to vary by machine.

The Need for an Account Management System

The Computer Science department at James Madison University (JMU) owns approximately fifty machines that run a variety of implementations of UNIX (SCO, Sun, Ultrix, AIX, LINUX, etc.). There are approximately 300 users of these machines. The machines are spread over a number of labs, with various collections of machines serving different functions. One collection of machines might provide support for a geographic imaging class, while another collection might support a class in systems programming. Thus the set of users requiring access to a machine differs greatly from machine to machine. The set of users also changes from semester to semester. This leads to a complex management task.

Without using an account management system, a system administrator must set up accounts individually on each machine. If three hundred persons need access to fifty machines, this would require the entry of 15,000 accounts. Because it is often desirable to give users varying access levels and attributes on different machines, simply copying information from one system to another is not sufficient. Clearly it is also neither efficient nor feasible to individually maintain account records spread across fifty systems.

The primary goal of this project was to find or create a system that would allow for the creation of a single ''network account'' for each user. However, account information would not have to be identical for all machines on the network. For any piece of account information, one or more ''clusters'' of machines could be defined which would share information. Such clusters would have the ability to be independent and overlapping.

Consider the following example for users 'alpha' and 'beta'; machines **APE**, **BOA**, **CAT**, **DOG**, and **ELK**; and two pieces of account information, passwords and login shells. User 'alpha' may wish to have one password for machines **APE**, and **CAT** and another for machines **DOG** and **ELK**. They also may want one login shell for machine **APE** and another for machines **CAT**, **DOG** and **ELK**. The user 'beta' might also require password and login shell clusters, but they might be quite different from user 'alpha'. Figure 1 summarizes clustering in this example.

ACCOUNT	**ITEM**	**CLUSTER**
alpha	password	APE, CAT
alpha	password	DOG, ELK
alpha	login shell	APE
alpha	login shell	CAT, DOG, ELK
beta	password	APE, BOA, CAT
beta	login shell	APE, BOA
beta	login shell	CAT, DOG, ELK

Figure 1: Cluster summary

We sought a system which created the abstraction of a network account, but which had the flexibility to allow different attributes to be associated with various systems on the network. We wanted the account administrator to be able to control all accounts from a single workstation.

An Evaluation of Commonly Used Tools

Part of this project included researching network account management tools currently in use in UNIX environments. A search of the literature revealed that there are two types of systems available for heterogeneous UNIX system management. First, there is software commonly distributed with the UNIX operating system or publicly available via Internet. This includes NIS, NIS+, and Kerberos. The other option is commercial products such as Tivoli System's TME, Xchange by Z/Max and CA-Unicenter by Computer Associates.

Literature research was augmented by discussions with several system administrators (two from the MITRE corporation, two at Bell Atlantic, and one at JMU), and various Internet resources such as Usenet postings, FTP sites, and the World Wide Web. These sources all seemed to suggest that commercial software is not often used. Reasons given for excluding commercial software included: (1) high expense, (2) reliance on a "closed" solution with no source code available for modification, and (3) no guarantees the software will work with future systems.

NIS, an Information Distribution System

The first system evaluated was the Network Information System (NIS), created by Sun Microsystems. This system, also known as yellow pages (yp), is widely used in UNIX environments today.

NIS is a service originally intended to allow the construction of "distributed computing systems" that are consistent in nature [Stern-92]. It does this by providing a "distributed database system" for common configuration files. Configuration files typically distributed by NIS include the UNIX hosts table, network services table, password file, and many others. By distributing configuration files from one central "server" to many "clients", NIS effectively allows multiple machines to behave as one.

One traditional use of NIS has been account administration via the */etc/passwd* and */etc/group* files. User accounts can be created on the server machine and distributed to all the clients. This approach effectively creates a network account of sorts, and works well when the account information needs to be consistent across all machines. One instance of this is when a number of identical machines have been purchased for collective use by a group. It is assumed that the users will want their accounts to have the same attributes, regardless of which machine they login to.

The environment within the Computer Science department at JMU is considerably different, as discussed earlier. It is desirable to be able to vary account information and create clusters of machines sharing information. NIS simply does not have this capability. Account information such as login name, password, and home directory must be shared either in its entirety or not at all. For example, it is not possible to distribute only an account's password and set expiration dates differently on each individual machine. (There is an exception mechanism built into NIS for the */etc/passwd* file. This allows each individual client to specify that accounts be added to, or removed from, the server's standard distribution. This process is cumbersome and does not provide partial sharing of an individual account's information.)

The inability to partially share account information makes it impossible to create independent and overlapping clusters under NIS. This was viewed as a major shortcoming and is due to the architecture of NIS. It is not likely simple modifications to NIS can overcome this problem. Because of this, it is difficult to justify the use of NIS in this project.

NIS+, the Successor to NIS

NIS+, released with the Solaris 2.0 version of UNIX by Sun, attempts to correct what many people see as drawbacks to NIS. Major advantages of NIS+ over NIS include secure data access, hierarchical network administration, and access to resources across domains [McManis-91]. However, a new problem has been created; NIS+ is not as widely available as NIS.

NIS+ improves security by adding authorization, authentication, and encryption not present in standard NIS. Secure RPC, also used by newer versions of the Network File System (NFS), provides for authentication and encryption during network communication. Secure RPC makes use of both DES and public key encryption [Stern-92]. This prevents sensitive (particularly password) information from being compromised through network sniffing and spoofing. Authorization has been addressed through the addition of access rights to each NIS+ database table. Read and Write permissions can be granted to the table's owner, group, and world. These access rights follow the standard UNIX filesystem model [McManis-91].

The ability to administer networks in a hierarchical fashion, and access resources across multiple domains is likely very useful in large corporate networks. However, the underlying structure of information distribution for NIS+ remains essentially the same as NIS. All account information must either be shared equally or not at all. Thus, NIS+ was also considered an inadequate solution to the problem.

Kerberos, an Authentication System

Kerberos is a product of Massachusetts Institute of Technology's Project Athena. It was designed with the goal of securing access to computers in an insecure network [Jaspan-93, Curry-90]. Kerberos authenticates users; it does not authorize them [Jaspan-93, Arnold-93, Sjogren-92]. There is a critical difference between these two activities. While Kerberos focuses on verifying that users and machines are who they say they are, this project was designed to centralize control of users' access privileges (authorizations) on the network.

Certainly authentication is an important network management issue, and can improve the security of account management systems. Future enhancements to this project will probably be concerned with authentication of machines on the network. There are several known mechanisms for authentication including Kerberos and RSA public key exchange. Authentication simply was not the issue this project was initially concerned with.

Tivoli's TME, an Enterprise Management Tool

Given the major problems of cost and lack of source code, commercial tools were not a primary focus of this project. We did take a closer look at TME [Tivoli-96], an enterprise management tool designed to allow centralized control of workstations, networks, and users with a common look and feel.

TME supports the central administration of workgroups with facilities for rapid addition or modification of the workstations and/or users associated with a workgroup. It appears well thought out, and covers most aspects of daily system administration. For our needs, it has two major weaknesses. First, it is not available on all the systems we are running. Second, it does not appear to support independent and overlapping information clusters. Although a more complete solution than NIS, it lacks the flexibility we sought.

Real Life System Management Experiences

As this project was not purely a theoretical problem, but also an attempt to create a practical solution to an everyday problem, it was deemed important to gather information "from the field." Discussions with system administrators at MITRE [Brown-95, Cwiek-94] and Bell Atlantic [Boss-95, Jordan-95] were very useful in providing some insights regarding commonly used techniques for account management. Both companies appear to rely on the use of NIS and NIS+. The administrators contacted were generally aware of the existence of Kerberos, and thought it was used "elsewhere" within their companies, but did not have any direct experience with it. It appears Kerberos may only be in use in applications considered critical or involving national security. None of these administrators had seriously considered 'enterprise' management systems such as TME and CA Unicenter. This may be due in part to the fact that these companies handle system administration primarily at the departmental level.

These discussions also revealed some interesting observations regarding the use of NIS and NIS+. NIS+ seems to work quite well for some groups at Bell Atlantic, particularly in networks consisting of nearly all Sun systems. An additional property of the Bell Atlantic networks is that a group of machines is typically assigned to a single project and users are assigned the same attributes on each machine belonging to the project. At MITRE, NIS+ was not used due to a more heterogeneous mixture of workstations (SGI, HP, IBM). NIS seemed a mediocre solution at best. While NIS worked well for distribution of some system configuration files, it failed at account administration. One reason for this failure was that users required different types of access to different machines. For one lab at MITRE, the situation became complex enough that the administrator used a "cut and paste" technique for accounts. Essentially, he was administering each machine's accounts independently. This only worked because there were few machines involved (about ten), and a small number of users (about thirty). On a larger scale, this would quickly become a nightmare. For a second administrator at MITRE, the situation was a little different. NIS was used for some (but not all) machines. Exceptions were handled by making changes on individual machines. This meant that if a user in an NIS domain shouldn't receive access to a particular machine, that user was removed manually. This required logging into multiple machines to make changes to multiple configuration files. Again, this is a far from ideal situation.

One conclusion that can be drawn from these observations is that the usefulness of NIS and NIS+ for account administration depends on the makeup of the network and users. NIS and NIS+ both work best in an environment that: (1) consists of homogeneous workstations, and (2) does not require user access levels or configuration (i.e., home directories) to vary by machine. Conversely, these systems are not well suited to heterogeneous networks and varied user information. These are the conditions that have complicated account administration at JMU and they apparently are not unique.

NAMS, A New Approach

One conclusion that can be drawn from these observations is that the usefulness of NIS and NIS+ for account administration depends on the makeup of the network and users. NIS and NIS+ both work best in an environment that: (1) consists of homogeneous workstations, and (2) does not require user access levels or configuration (i.e., home directories) to vary by machine. Conversely, these systems are not well suited to heterogeneous networks and varied user information. These are the conditions that have complicated account administration at JMU and they apparently are not unique. The Network Account Management System (NAMS) is the result of an effort to design a better solution for account management. NAMS is an operational system currently responsible for distributing account information within JMU's Computer Science network.

Like NIS and NIS+, NAMS distributes information from a central server to individual machines. However, NAMS is a very specific system created for one purpose: to administer network accounts. If desired, NIS/NIS+ services can be used in conjunction with NAMS for distribution of other UNIX configuration files.

NAMS is considerably more flexible than NIS with respect to sharing of account information. It incorporates the notion of clusters, and maintains a database designed to store account information containing these clusters.

NAMS is a modular system, made up of programs which comprise the central database, network communications, and user interface. This modular structure has allowed NAMS to be ported to new UNIX systems rather quickly, and should also allow its functionality to be easily expanded in the future.

Database Structure

Perhaps the most significant difference between NAMS and NIS is the nature of the data stored. NIS creates its database from the standard UNIX configuration files. It then distributes the information in the configuration files to all machines within its domain. NAMS was designed from the ground up with the idea of being an network account database as opposed to being a distribution system. The NAMS database contains the ''items'' of information considered relevant to user accounts on the network. The database structure is open in nature to allow the system administrator to customize it for his or her needs. A typical database would contain some standard information such as username, password, and home directory. However, NAMS is also capable of storing information like disk quota space, email addresses, email forwarding, and network directories owned by the user. These items are defined by creating a file named *items.data* in the NAMS data directory. Figure 2 below is a current listing of this definition file for the Computer Science department.

The three columns define: the information item's ''name'', the access permissions associated with that item, and a description of the item. The access permissions determine who can read and modify each item. Write access (w) also confers read access (r) to an item. The first letter indicates read and write permissions for the account owner. The second letter indicates read and write permissions for all other users in the world.

In the example above, users are only permitted to change their passwords. To also allow users the ability to change their login shell, email, and forwarding addresses, one would need to change the rights field in the ''shell'', ''email'', and ''forward'' lines from ''rr'' to ''wr''.

While not shown above, NAMS also maintains a list of system administrator accounts that have read and write access to all items in the database.

Clusters

Information stored in the NAMS database has the ability to be grouped into clusters. For example, a single account might have two passwords defined on the network. Password A could be used for common, unsecured machines. Password B might apply to more sensitive machines. When either of these passwords is changed on one machine, all the other machines in that cluster will receive the updated information.

An asterisk at the end of an item name indicates the entry defines a machine cluster. In the above example, **passwd**, **home**, and **status** are clustered information items. Each account may partition the machines in the network into one or more clusters which share that particular information item. The other items are defined as global items, the data associated with that item is common to all machines on the network.

Figure 3 shows typical account entry in the NAMS database.

```
#
#       :ItemName:Access:Description
#
:uid     :rr:numerical user id
:gid     :rr:numerical group id
:realname:rr:real name
:deptid  :r-:department id number
:passwd* :r-:encrypted password clusters
:passwd  :w-:encrypted password
:home*   :rr:home directory clusters
:home    :rr:home directory
:shell   :rr:login shell
:comment :r-:comment
:email   :rr:JMU email address
:forward :rr:email forwarding address
:status* :r-:account status/expiration clusters
:status  :r-:account status/expiration date
:create  :r-:date account created
:expire  :r-:overall account status/expiration
:quota   :r-:disk space quota
:dirlist :r-:list of owned directories
```

Figure 2: A Listing of the Item Definition File

In the example above, the first column of data is the account name. The second column holds the name of the information being stored and corresponds to the items of information defined in the *items.data* file described earlier. A starred item in the second field indicates this line defines the systems in a cluster. The third column stores the ‘‘cluster number’’. A cluster number of ‘0’ is used for non-clustered information such as a user’s real name. Each defined cluster occupies two lines: one line lists the machines in the cluster, the other specifies the value for the cluster. For example, the following two lines would define a password cluster for machines named **ruff**, **sys10**, and **tch3**, and the encrypted password used by those machines.

```
:gingergl :passwd* :1:ruff,sys10,tch3
:gingergl :passwd  :1:encrypted_entry_A
```

A second cluster is used to define the password for all remaining systems.

```
:gingergl :passwd* :2:*
:gingergl :passwd  :2:encrypted_entry_B
```

Note the use of the ‘*’ above to indicate all machines not otherwise defined. This entry is required in the database for each clustered item to ensure that no machine will have an undefined value for an item. Another example of clustering in the above entry is the ‘‘status’’ field. The following two lines define a group of machines to which the user has access through 12/95.

```
:gingergl :status* :1:sys1,sys2,tch3
:gingergl :status  :1:12/95
```

A second cluster defines systems this user does not have access to.

```
:gingergl :status* :2:thor,mutt
:gingergl :status  :2:noaccess
```

The third cluster defines an expiration date of 05/95 for all remaining systems.

```
:gingergl :status* :3:*
:gingergl :status  :3:05/95
```

Access could be denied to systems in any of these clusters by changing the expiration date to a value of ‘‘noaccess’’.

Information Distribution

In order to distribute information NAMS relies on the use of TCP/IP network communications [Comer-91, Hunt-92, Stevens-90]. There are currently two types of information flow between machines running NAMS. The first type consists of updates to the central database located on the server. The second type consists of updates sent from the server to individual clients.

Server Updates

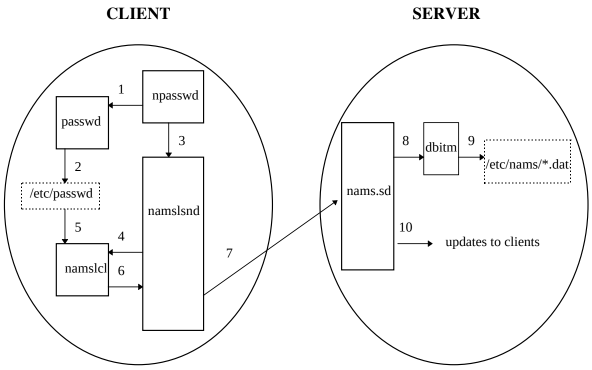
When an item of information (such as an account password) has been changed on a client machine, the server can also be updated with the change using the program **namslsnd** (short for NAMS Local SeND). This program retrieves information stored locally using a subprogram (shell script) named **namslcl** (NAMS LoCaL information). It then connects with a daemon running on the server called **nams.sd** (NAMS Server Daemon) and passes it the updated information

```
:gingergl :gid     :0:500
:gingergl :realname:0:gregory gingerich
:gingergl :deptid  :0:csmajor
:gingergl :passwd* :1:ruff,sys10,tch3
:gingergl :passwd  :1:encrypted_entry_A
:gingergl :passwd* :2:*
:gingergl :passwd  :2:encrypted_entry_B
:gingergl :home*   :1:*
:gingergl :home    :1:/homes/gingerg
:gingergl :shell   :0:/bin/ksh
:gingergl :email   :0:ginger@vax1.jmu.edu
:gingergl :forward :0:mcro_greg@jmu.edu
:gingergl :status* :1:sys1,sys2,tch3
:gingergl :status* :2:thor mutt
:gingergl :status* :3:*
:gingergl :status  :1:12/95
:gingergl :status  :2:noaccess
:gingergl :status  :3:05/95
:gingergl :create  :0:01/92
:gingergl :expire  :0:12/95
:gingergl :quota   :0:30
:gingergl :dirlist :0:/homes/gingerg
```

Figure 3: An Account Entry in the NAMS Database

via a UNIX socket. The server daemon places the updated information in the NAMS database using a program called **dbitm** (DataBase ITeM) and sends a completion code back to the client. After terminating the connection with the client, the server makes a list of all other clients in the updated item's information cluster. It then broadcasts the update to those systems, as described below.

Perhaps the item most frequently updated is an account password, which is accomplished using the shell script **npasswd**. The steps involved in changing one's password are described below and illustrated in Figure 4.



0. Execute the NAMS program **npasswd**.
1. **npasswd** executes **passwd**.
2. **passwd** obtains a new password and modifies */etc/passwd*.
3. **npasswd** executes **namslsnd**.
4. **namslsnd** executes **namslcl**.
5. **namslcl** extracts account information from */etc/passwd*.
6. **namslcl** returns account information to **namslsnd**.
7. **namslsnd** sends the account information to **nams.sd**.
8. **nams.sd** passes the account information to **dbitm**.
9. **dbitm** modifies the NAMS database.
10. **nams.sd** broadcasts the update to all other machines in the same password cluster.

Figure 4: Update password

Client Updates

There are two mechanisms used for updating clients with information stored on the server. The first involves the client contacting the server and requesting an update. The second method has the server broadcasting messages to client machines listening continuously for updates.

The client may contact the server daemon to request an update using a program called **namsupc** (NAMS UPdate Client). The request is either for information on an individual account or for all accounts (which are valid on the client machine). The program **namsupc** establishes a UNIX socket connection with server daemon **nams.sd** and sends the list of items it wishes to be sent. The server daemon then checks to see which machine is requesting the update using an IP number lookup. If the request is for all accounts, it constructs a list of the accounts that are valid on the machine making the request. The socket is then used by the server to send the information to **namsupc** which uses **namslcl** to store the information in the client's standard account files. The process is illustrated in Figure 5.

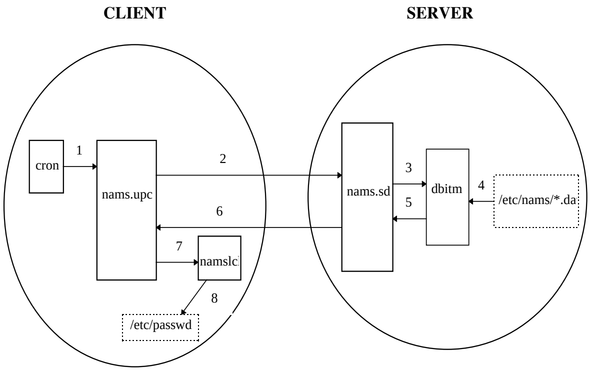


Figure 5: Update information on client

The second form of client update involves the use of the client daemon **nams.cd** constantly listening on a privileged UNIX socket for updates. When the server daemon receives an update, it broadcasts the update to **nams.cd** on all the clients in the same information cluster. The client daemon uses **namslcl** to store that information in the standard system account files. See Figure 6.

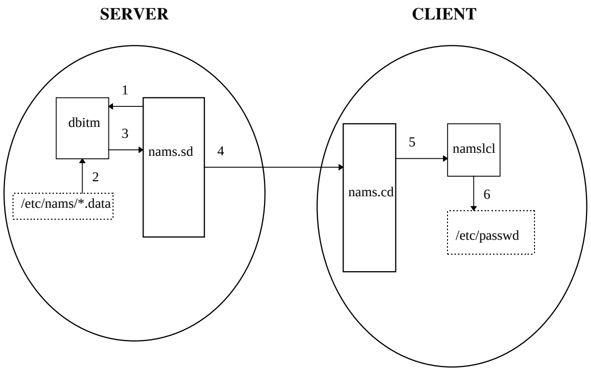


Figure 6: Update clients

The **namsupc** program is typically used at startup time to synchronize the client with the information stored on the server. It can also be scheduled by the system administrator to run (using a UNIX cron entry) several times daily. While this mechanism alone could provide updated information, it does not allow for changes to be distributed throughout the network quickly. The client daemon **nams.cd** provides a method for faster distribution of changes. This is particularly desirable in the case of password updates.

Security, A Growing Concern

When discussing security issues, there are two important rules. The first is ‘‘need to know.’’ Users of a system should not be told more about underlying security than necessary [Arnold-93]. The second rule is that if you must rely on the first rule for security of your system, you are in real trouble. Keeping this in mind, we will try to explore the security issues related to NAMS at a high level of abstraction.

Security in NAMS

Weaknesses in the design or configuration system software can have drastic effects on UNIX security [Arnold-93, Garfinkel-91, Curry-90, Spafford-88]. Associated with a system like NAMS are a number of security concerns. Potential problem areas include NAMS programs disclosing or corrupting data in NAMS data files or in standard system files (like the */etc/passwd* file), sending information to unauthorized machines, and receiving bogus information from unauthorized machines.

All NAMS data is stored in a directory to which normal users need no access. All NAMS programs access NAMS data through the **dbitm** NAMS utility. By creating a special user id ‘nams’, setting permission so only user ‘nams’ (and ‘root’) may access the data, and running **dbitm** setuid to ‘nams’, access to NAMS data can be restricted.

The only other program requiring special privileges is the program which sends local client information to the server, **namslsnd.** It must run setuid to ‘root’ so it can access standard accounting files and thus is a potential security loophole. For this reason, special care has been taken in its development. It is designed to refuse an account (other than root) access to another account’s information.

The client and server daemons are not setuid programs and therefore are of no security concern when called directly by a user. However, they must be run by ‘root’ when started as daemons. All network communication uses privileged UNIX sockets to prevent non-‘root’ access. Additionally, limited authentication is performed based on the IP number of the client sending information.

Sniffing and Spoofing

Concern over ‘‘sniffing’’ and ‘‘spoofing’’ security attacks has risen dramatic in recent years. Sniffing is the monitoring all network traffic, searching for critical information such as passwords. Spoofing occurs when a rogue machine pretends to be another machine. If the machine being impersonated (spoofed) is trusted by other machines on the network, this can be exploited through protocols such as **rlogin** and early versions of NIS and NFS. The degree to which sniffing and spoofing is a problem depends greatly on such factors as a network’s physical security, the degree of access to the network from global networks, the type of users on the system, and the sensitivity of information stored on the systems. In our case, sniffing and spoofing were initially a secondary concern.

Given NAMS must send account information over the network, the initial version was susceptible to an attack by a sniffer. Although NAMS does not send plaintext passwords over the network, the first version does send encrypted passwords from one machine to another. On many networks, a sniffer could pick up these encrypted passwords and attempt to break them through the use of a program such as Crack. The first version of NAMS provides minimal protection against such attacks.

Spoofing is another method by which NAMS could be attacked. Clients running the daemon **nams.cd** could be updated with false information by a machine masquerading as the server. However, physically securing the server machine would make it relatively difficult to spoof. One also has the option of not running the client daemon and obtaining updates only with explicit requests via **namsupc.**

A more likely scenario is that a client would be spoofed. If users have physical access to client machines, it would be easy for them to turn off one of these machines. They could then plug a personal laptop into the network or replace the operating system on an existing machine. By assuming the IP number of the disabled client machine, the rogue system could fool the server into believing it was communicating with a legitimate client. Assuming access to the NAMS software or knowledge of its protocols, an update request from this bogus client could then change passwords or other information on the server.

To address these sniffing and spoofing concerns, Version 2 of NAMS protects network communications using independent one-time encryption keys. Based on a system described in [Rubin-96], each client is assigned a unique ‘‘client-key’’, known only to the client and the server. The client-key is the concatenation of three 8-byte DES keys. At the start of each session, the client sends the server a unique ‘‘session id’’. The session id is an integer between 1 and 32767, and changes with each communication between client and server. A pseudo-random function takes the ‘‘session id’’ and the client-key, and using three iterations of DES, generates a ‘‘session-key’’ using the formula shown in Figure 7. For the remainder of the session, all communications between client and server are encrypted with the session key. The ability to encrypt or decrypt with the session key authenticates the client or server system.

```
S-Key = DES(DES^-1(DES(S-Id,C-Key1),C-Key2),C-Key3)
```

Figure 7: Session key encoding

A Security Flaw Discovered

One interesting flaw discovered during the writing of this paper is related to a design problem with user-definable clusters. If users were given permission to change both their password and their password clusters, they could defeat the security inherent in the clustering.

1. A user establishes two password clusters. One cluster of machines is considered more sensitive than the other.
2. The user's account is compromised on the 'low security' cluster, either through a weakness of the system or user carelessness.
3. A cracker who has gained access to one of the lower security machines takes the following action to gain access to one of the sensitive machines. First they change the account's password clustering to include the compromised machine in the "sensitive" cluster. Next they update the password on the compromised machine. The cracker now has access to any machine in the sensitive cluster.

This problem is particularly bothersome because compromise of one system's root account (allowed to update all accounts) could lead to compromise of an entire network of machines running NAMS. For this reason, NAMS now limits the ability to change clustering to system administrators on the NAMS master. Potential threats such as this must be dealt with as they are discovered.

Performance

NAMS has been in use in the Computer Science Department at James Madison University since the fall of 1995. Currently, NAMS is managing 294 accounts on a network of six Sparc Stations running SunOS and one Intel based PC running SCO ODT. In the past, the network managed by NAMS has also included five IBM Risc systems running AIX and thirty additional Intel based PCs running SCO ODT (reallocations of resources made those systems no longer available for UNIX use).

On a lightly loaded system, execution time for updating account information on a client machine averages around 2 seconds. Updating all 294 accounts takes around 130 seconds. No significant difference has been found in the speed necessary to update a PC client and a Sparc 20. Performance degrades if a number of clients request simultaneous updates of all their accounts. For four simultaneous updates of all accounts, execution time averaged 271 seconds.

It should be noted that this level of performance has been acceptable to date, since the update times only impact the speed with which changes to accounts occur. Validation of logins is nearly instantaneous, since each client machine still relies on its own */etc/passwd* and */etc/shadow* files to authenticate users. The current performance limits do create a scalability issue. Performance of a network of several hundred or thousand machines is not known at this time. Options for enhancing performance would include using **ndbm** for the database engine, converting pieces of code from shell scripts to C/C++ code, and using a "Master/Slave" server configuration similar to NIS.

Summary, Future Enhancements to NAMS

This project has resulted in the successful development of a network account management system. The current implementation has successfully shown that the concept of clustering is both useful and viable. It has also raised some interesting security issues related to user-definable clusters.

NAMS was purposefully built in a modular fashion with the anticipation that it will grow over time. A menu-driven or window-based interface for the system administrator would also provide significant benefits. Currently, the administrator utilities consists primarily of line oriented add-account and remove-account commands. Beyond this, the system administrator must understand the database structure of NAMS and edit it manually or through use of the **dbitm** command. Also a utility is needed which includes a well organized structure for accessing and updating information, some intelligence to simplify the administrator's job and automate database changes, and possibly another layer of abstraction to hide complex details.

NAMS software is available via anonymous FTP at "ftp.cs.jmu.edu".

Author Information

J. Archer Harris received his MS and Ph.D. degrees in Computer Science from SUNY at Stony Brook. He is currently an associate professor of Computer Science at James Madison University. He can be reached via U.S. Mail at Department of Computer Science; James Madison University; Harrisonburg, VA 22801. Reach him electronically at <arch@cs.jmu.edu>.

Greg Gingerich received a BS-CS from James Madison University in 1995. He is currently a programmer with the Broadband Systems group at Bell Atlantic working on a Data Network Management System. Reach him via U.S. Mail at Bell Atlantic Broadband Systems; 13100 Columbia Pike, POD A33; Silver Spring, MD 20904-5247. Reach him electronically at <bftxddm@Bell-Atl.com>.

References

Arnold, N. Derek, *Unix Security: A Practical Tutorial,* McGraw, 1993.

Boss, Todd, Bell Atlantic (Silver Spring, MD), Personal discussions, 1995.

Brown, Philip, The MITRE Corporation (McLean, VA), Personal discussions, 1994.

Comer, Douglas, *Internetworking with TCP/IP,* Prentice Hall, 1991.

Curry, David, ''Improving the Security of Your Unix System'', Technical Report ITSTD-721-FR-90-21, SRI International, April 1990.

Cwiek, Daniel, The MITRE Corporation (McLean, VA), Personal discussions, 1994.

Garfinkel, Simson, and Gene Spafford, *Practical Unix Security,* O'Reilly and Associates, 1991.

Hunt, Craig, *TCP/IP Network Administration,* O'Reilly and Associates, 1992.

Jaspan, Barry, Kerberos Users' Frequently Asked Questions, OpenVision Technologies, November 1993.

Jordan, Thomas, Bell Atlantic (Silver Spring, MD), Personal discussions, 1995.

McManis, Chuck, and Saqib Jang, Network Information Service Plus (NIS+): A White Paper, Sun Microsystems, 1991.

Rubin, Aviel, ''Independent One-Time Passwords'', *Computing Systems,* Vol.9, No.1, Winter 1996

Sjogren, Sam, ''A Bite Out of LAN Crime'', *UNIX Review: Interoperability Supplement,* Spring 1992.

Spafford, Eugene H, ''The Internet Worm Program: An Analysis'', Technical Report CSD-TR-823, Purdue University: Department of Computer Science, November 1988.

Stern, Hal, *Managing NFS and NIS,* O'Reilly & Associates, 1992.

Stevens, W. Richard, *UNIX Network Programming,* Prentice Hall, 1990.

About Tivoli, ''http://www.tivoli.com/products/products.html''.