Journal of Network and Computer Applications (1999) **22**, 199–213
Article No. jnca.1999.0091, available online at http://www.idealibrary.com on IDEAL®

Towards role-based administration in network information services

Gail J. Ahn and Ravi Sandhu

ISE Department, Mail Stop 4A4, George Mason University, Fairfax, VA 22030, USA; E-mail: {*gahn*, *sandhu*}*@isse.gmu.edu*

Client-server networks have grown tremendously since the mid-1980s. The information they store changes much more rapidly than it did during the time at beginning. The size and complexity of these networks required new, autonomous administration practices. Network Information Services was designed to address these requirements. As one of the network information services, NIS+ is a widely used network protocol. It allows networked machines to have a common interface regardless of the workstation that a user logs into. NIS+ keeps all information into tables to support a common interface between networked machines. In the most of case these information are controlled by centralized manner. Centralized management of NIS+ tables in large systems is a tedious and costly task. An appealing possibility is to use role and role hierarchy to facilitate decentralized administration of NIS+ tables which has not been previously recognized in the literature. This paper presents decentralized administration of NIS+ tables using the notion of role and also shows how to simulate role hierarchy in NIS+ groups.

1. Introduction

Network information services store information that users, workstations, and applications must have to communicate across the network. Without a network information service, each workstation would have to maintain its own copy of this information. For example, take a simple network of three workstations; Alice, Bob and Chris. Before Alice can send a message to either Bob or Chris, it must know their network addresses. For this reason, it keeps a file, `/etc/hosts`, that stores the network address of every workstation in the network, including itself. Likewise, in order for Bob and Chris to communicate with Alice or with each other, they must keep similar files. However, addresses are not the only network information that workstations needs to store. They also need to store security information, mail information, information about their Ethernet interfaces, information about network services, and about groups of users allowed to use the network, about services offered on the network, and so on. As networks offer more services, the list grows. As a result, each workstation may need to keep an entire set of files similar to `/etc/hosts`. Unfortunately, some files change too frequently to be kept current. The problem of keeping system information current is especially troublesome with very dynamic information such as passwords. As this information changes, administrators must keep it current on every workstation

1084–8045/99/070199 + 15 $30.00/0

in the network. In a small network this is simply tedious, but on a large network, the job becomes not only time consuming, but unmanageable.

Network information services solve this problem by maintaining system information on a central host known as an NIS+ server. It stores network information on servers and provides it to any workstation that asks for it. Network Information Services allow networked machines to have a common interface regardless of the workstation that a user logs into. With this mecahanism we can use the same password and group files (the same uid and gid) over the network and the same home directory can be mounted on each machines. It also reduces the effort required to setup and maintain a network of Unix workstations. This is accomplished through the centralization on a *server* of the major configuration files required to setup a Unix machine for a particular site. Thus only the set of configuration files on the server need to be updated to effect all machines at a site. This has proven to be a very powerful network administration tool.

However, the information on the server is exclusively controlled by a centralized manner which does not scale gracefully to systems with large numbers of groups and users. More generally, it is possible to decentralize an administration by allowing administrators to selectively delegate authority to control the information. In this paper we describe decentralized administration of NIS+ can be implemented by means of setting access rights to NIS+ objects (specially to NIS+ table) supporting NIS+ administration mechanism.

The rest of the paper is organized as follows. In Section 2, we discuss the overview of network information services. In Section 3 we review the role concept and discuss its adoption in NIS+ followed by implementation of Role-based NIS+ in Section 4. Section 5 concludes the paper.

2. Overview of NIS and NIS+

In this section we describe the background of network information services. And we present the basic structure of NIS+ which we will focus on.

2.1 *Background on NIS and NIS+*

NIS was developed independently of DNS and had a slightly different focus.[1] Whereas DNS focused on making communication simpler by using workstation names instead of addresses, NIS focused on making network administration more manageable by providing centralized control over a variety of network information. As a result, NIS stores information not only about workstation names and address, but also about users, the network itself, and network services. This

[1] DNS, the Domain Naming Service, is the network information service provided by the Internet for TCP/IP networks. It was developed so that workstations on the network could be identified with common names instead of Internet addresses. NIS and NIS+ are other network information services developed by *SunSoft*^{TM}.

collection of network information is referred to as the NIS namespace. NIS stores information in a set of maps. However, NIS maps were designed to replace Unix etc files, as well as other configuration files, so they store much more than names and addresses. As a result, the NIS namespace has a large set of maps. NIS+ was designed to replace NIS. NIS addresses the administration requirements of client-server computing networks prevalent in the 1980s. Client-server networks have grown tremendously since the mid-1980s. The size and complexity of these networks required new, autonomous administration practices. NIS+ was designed to address theses requirements. NIS+ is upwardly compatible with NIS. The new service provides for a hierarchical name space, similar to that used by the Internet. This allows for a distributed authority mechanism.

NIS and NIS+ are based upon the Remote Procedure Call (RPC) protocol which uses the External Data Representation (XDR) standard. Below this level are the raw communications services of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) provided by the Internet Protocol (IP) [2]. The relationships between these protocols are shown in Fig. 1. RPC implements a method by which a *client* process on one machine can perform a virtual procedure call to a *server* process on a remote machine. The client is considered to be accessing a feature of a service provided by the server. The client calls an RPC procedure with the arguments for the remote procedure and does not return from the call until the request has been sent to the server, processed, and a reply received. The message is encoded using XDR so that RPC can be used between heterogeneous machines using different internal data representations [1, 3, 4]. The actual transmission of data is performed using either TCP or UDP depending on the desires of the client and the design of the server.

2.2 *Architecture overview of NIS+*

The NIS+ domain is composed of a directory object and all of its children as shown in Fig. 2. The NIS+ name space is made up of all the domains below the root directory. Each name is composed of a series of characters separated by

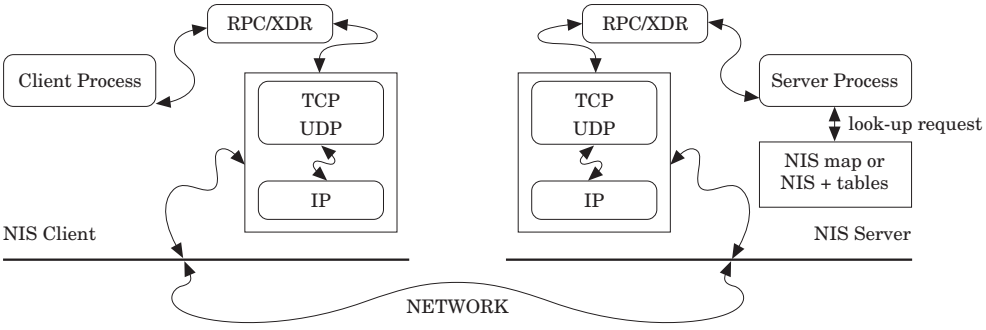


Figure 1. NIS mechanism.

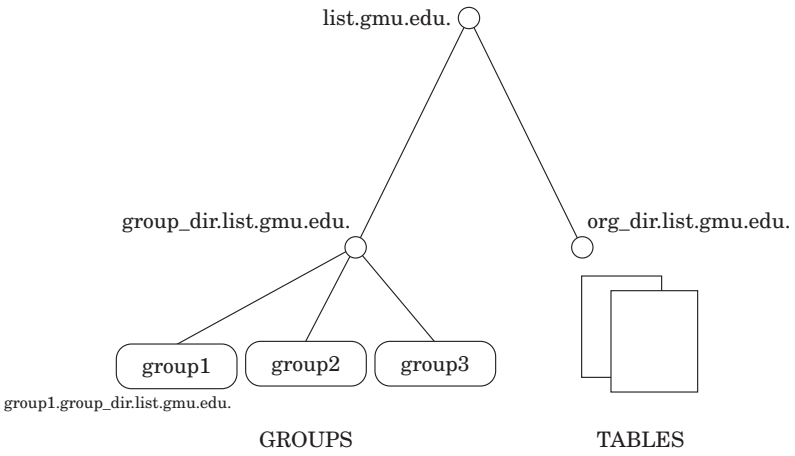


Figure 2. NIS architecture.

a (.). These character sequences are known as labels. There are three types of objects:

`objects`	description
`directory objects`	they form the framework of the name space
`table objects`	they store the information
`group objects`	they are used for security

The `directory objects` are at the top of the name space. Directory objects contain the names, addresses, and authentication information for systems within the domain. Objects within the database are stored as children of the directory object. The directory object at the top of the hierarchy is known as the root directory. The `table objects` identify table databases. The table object contains the scheme by which columns within the table can be identified and searched. Each table contains information about users, machines, or resources on the network.[2]

The `group objects` contain a list of members of the group. An NIS+ group is a collection of users and workstations identified by a single name. They are assigned access rights as a group. Essentially, this is used to set security. As shown in Table 1(a, b) NIS+ authorization allows four classes of principals and four access rights. NIS+ objects specify access rights for NIS+ principals in the same way that UNIX files specify permissions for UNIX users. Access rights specify the types of operations that NIS+ principals are allowed to perform on

[2] The normal set of 16 tables store information for: `hosts`, `bootparams`, `password`, `cred`, `group`, `netgroups`, `mail`, `aliases`, `timezone`, `networks`, `netmasks`, `ethers`, `services`, `protocols`, `rpc`, `auto_home`, `auto_master`.

Table 1. *The basic structure of NIS+*

(a) Principals

Principal	Description
Owner	owner of the object
Group	set of specified users
World	set of authenticated users
Nobody	all clients

(b) Access rights

Right	Functionality
read	read contents of objects
modify	change objects
create	add objects to tables
destroy	remove objects from tables

(c) Default access rights for NIS+ objects

Object	Nobody	Owner	Group	World
Root-directory object	r---	rmcd	rmcd	r---
Non-root directory object	r---	rmcd	rmcd	r---
group_dir directory objects	r---	rmcd	rmcd	r---
org_dir directory objects	r---	rmcd	rmcd	r---
NIS+ groups	----	rmcd	r---	r---
NIS+ tables	varies	varies	varies	varies

an NIS+ object. NIS+ operations vary among different types of objects, but they fall into for classes: read, modify, create, and destroy. NIS+ objects specify their access right as part of their object definitions. So, if the operation that a principal tries to perform on an object is authorized by the object's definition, the server performs it. There is one more wrinkle in this process. An object does not grant access right directly to a particular principal. Instead, it grants access rights to four classes of principal: `Owner`, `Group`, `World`, and `Nobody`. The principal who happens to be the object's owner gets the rights granted to the Owner class. The principals who belong to the object's Group class get the rights granted to the Group class.[3] The `World` class encompasses all NIS+ principals that a server has been able to authenticate. Finally, the Nobody class is reserved for everyone, whether an authenticated principal or not. Table 1(c) shows the default access rights for NIS+ objects.[4]

3. Roles in Network Information Systems

As metioned in Section 2, NIS+ maintains the tables to store information about users, machines, or resources on the network. And controlling these tables is done by a centralized method. Centralized management of NIS+ tables in large systems is a tedious and costly task. An appealing possibility is to use role and

[3] This is an NIS+ group, by the way, not a UNIX group or a netgroup. Information about NIS+ groups is not stored in the NIS+ group table. That table stores information about UNIX groups. Information about NIS+ groups is stored in NIS+ group *objects*, under the groups_dir subdirectory of every NIS+ domain.

[4] We can change permission attributes of an object with the `/usr/bin/nischmod` command. The `/usr/bin/nisls` command can be used to list the objects and permissions of an NIS+ directory.

role hierarchy to facilitate decentralized administration of NIS+ table which has not been previously recognized in the literature. In this section we address the needs of role and role hierarchy in network information systems.

3.1 *Groups and roles*

NIS+ tables store a wide variety of information, ranging from user names to Internet services. Although NIS+ tables have the same underlying sturcture, each table stores different types of information. For instance, `auto_master` table contains automounter map information which can be used in network filesytem. `Passwd` and `cred` tables store password information and credentials for users who belong to the domain. These two tables are used for authentication process. `Networks` and `netmasks` tables list the network masks used to implement standard Internet subsetting including the networks of the Internet created from the official network table maintained at the Network Information Control Center (NIC). Unlike `networks` and `netmask` tables, hosts table associates the names of all the workstations in a domain with their IP addresses. These NIS+ tables are normally administrated by an administrator or single administrator group. This kind of management in large systems is a tedious and costly task. Also, single mistake may affect whole systems because administration of NIS+ tables does not have a supervising method such as hierarchical structure between administrator group.

Instead of centralized and single-level administration, we introduce decentrailized and hierarchical administration. In order to do that we divide NIS+ tables into several categories according to their types and characteristics because each table has different types of information. For example, as shown in Fig. 3, we have four categories grouped by their similarity and sensitivity. For instance,

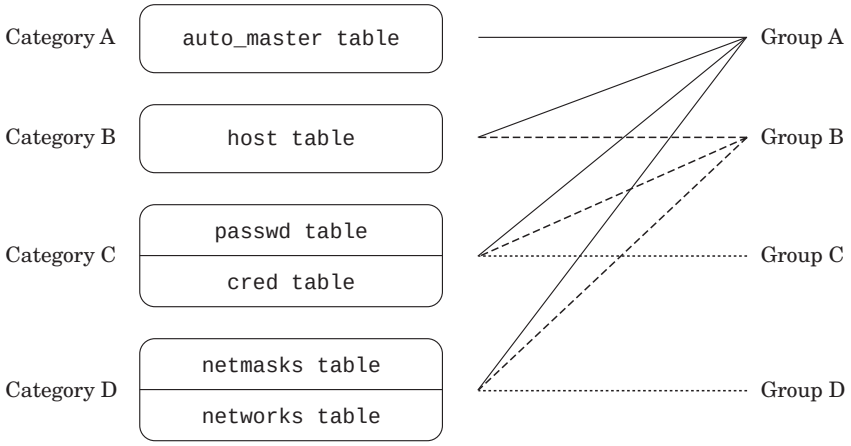


Figure 3. A scenario.

passwd and cred tables can be in the same category because those tables are used for administrating user accounts. Also netmasks and networks tables can be in the same category because those tables are referred to configure the network environment. host table can be another category itself because the modification of this table affects all network configuration so this table have more sensitive information than netmasks and networks tables. An auto_master table can be another important category which is used for mounting filesystem. In addition, these categories should be controlled by different administator groups instead of single administrator group. Also we use the notion of hierarchy between these administrator groups so that we can support least previlege in relationships between administrator groups. Therefore, senior group can do what junior group can do as well as his task(s). That is, a group that can control auto_master table should be able to control other categories because auto_master table has highly sensitive information.[5] Most of system including NIS+, however, does not support hierarchical relationship between groups. We bring role concept here to support this. We can define a role for each administrator group such as senior security officer (SSO), junior security officer (JSO), account security officer (ASO), and network security officer (NSO). And these different roles can access each categories in Fig. 3 as shown in Table 2.

The question of what is different between roles and groups inevitably arises. A group is a named collection of users and possibly other groups. A users can be directly made a member of a group or indirectly by means of including one group in another. Users are brought together in a group for some access control purpose. Groups serve as a convenient shorthand notation for collection of users and that is the main motivation for introducing them. Role is a job function within the organization that describes the authority and responsibility conferred on a user assigned to the role [6, 7]. Roles are organized in a partial order, so that if $x > y$ then role x inherits the permissions of role y, but not vice versa. In such cases, we say x is senior to y. A role is a named collection of permissions, and possibly other roles. The motivation for roles is convenience in administration and convenience in articulating policy. Also that the name of a role has significance and indicates the purpose of the role. Permissions enable activity in the system. In

Table 2. *Example of roles*

Role	Accessible Tables
SSO	auto_master, host, netmasks, networks, passwd, cred
JSO	host, netmasks, networks, passwd, cred
NSO	netmasks, networks
ASO	passwd, cred

[5] This hypothetical categorization is just for example purpose. We can make any category according to an organization's policy.

terms of abstract operations, a physician role may have the permission to write prescription. Similarly, a manager role have may permissions to hire and fire employees. A role could be viewed as a collection of users in which case there is no difference between a role and a group. Since the notion of a role is similar to that of groups in NIS+, particulary when we focus on the issue of user-role and user-NIS+ group membership, we adopt the notion of role.[6]

3.2 *Role hierarchy*

NIS+ notably lacks a facility for expressing relationship between groups. In practice, it is often desirable that groups bear some relationship to each other. For instance, consider a project divided into several independent tasks assigned to different teams. We can define a group for each task team so its members have common access to files relevant to the task. Since some files may pertain to the entire project we can define a project group such that members of the individual task groups are thereby also members of the project group. The project-wide files are then made explicitly available to the project group alone. This is certainly more convenient than having to explicitly make such files available to every task group. We iterate, hierarchical groups are useful. It is also more convenient than explicitly making every member of a task group a member of the project group. By allowing membership in a group to automatically imply membership in some other groups we can reduce the number of explicit access decisions that need to be made by users and administrators. Many commercial database management systems, such as Informix, Oracle and Sybase, provide facilities for hierarchical groups (or roles). Commercial operating systems, however, provide limited facilities at best for this purpose.

Let $x > y$ signify that group x is *senior* to y, in the sense that a member of x is also automatically a member of y but not vice versa. Note that a member of x has the power of a member of y and may have additional power, hence a member of x is considered senior to a member of y. It is natural to require that seniority is a partial ordering, i.e. $>$ is irreflexive, transitive and asymmetric. The irreflexive property is obviously required since every member of x is already a member of x. Transitivity is certainly an intuitive assumption and perhaps even inevitable. After all, if $x > y$ and $y > z$ then a member of x is a member of y and so should also be a member of z. The asymmetric requirement eliminates redundancy by excluding groups which would otherwise be equivalent. We write $x \geq y$ to mean $x > y$ or $x = y$. If x is senior to y we also say that y is *junior* to x. For convenience we use the term hierarchy to mean a partial order.

Figure 4 shows a role hierarchy using groups in NIS+. The senior-most role is the senior security officer. Junior to SSO is a junior security officer role, an

[6] Although NIS+ groups are different from our concept of roles, in certain situations NIS+ groups can implement roles. That is, the group mechanism closely resembles our role concepts.

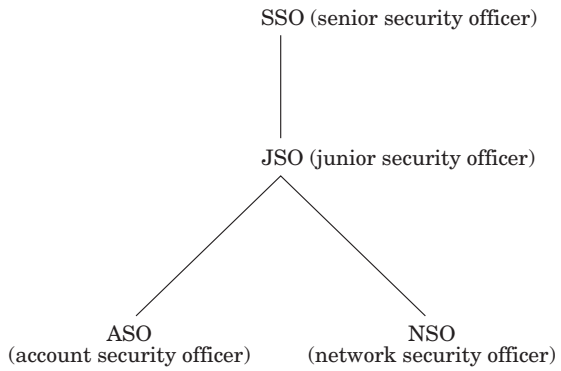


Figure 4. An example administrative group hierarchy.

account security officer role, and a network security officer role. These groups in NIS+ are authorized to access and modify NIS+ tables as we will see shortly. We will use this example throughout the rest of this paper.

4. Implementation of role-based NIS+

Different types of groups have proliferated throughout UNIX. NIS+ creates one more type: NIS+ group. The member of NIS+ group can be the principal and other group. If a NIS+ group is a member of other group, it is called as recursive group. For instance, assume that a NIS+ group `junior_admin` is a member of a NIS+ group `senior_admin`. In this case `junior_admin` is a recursive group and will be represented with the symbol '@' as follows. It is the representation of NIS+ group object in NIS+. It defines that `senior_admin` is group object and `junior_admin` is one of its members.

```
Object Name : senior_admin
    ...             ...
Object Type : GROUP
    ...             ...
Group Members :
      @junior_admin
```

An NIS+ group is used only as a means to provide NIS+ access rights to several NIS+ principals at one time; it is used only for NIS+ authorization. In order to do NIS+ authorization, each NIS+ group is assigned to table's (or object) Group class which allows the principal who belong to the Group class to get the rights granted to the Group class. In Table 1(c) we notice that the access rights for NIS+ tables is various. Therefore, we can assign different NIS+ group to each table's Group class.

That is, member users of a table's group owner can have special privileges to that object. For example, we can add several junior administrators to the administrator group so that they can only modify the password and hosts tables, but they would be unable to modify any other tables. By using this mechanism, we can distribute administration tasks across many users and not just reserve them for the superuser of the entire table [5]. Each administrator group can be role which performs certain control over the NIS+ tables. In our case study, we also have four NIS+ groups and each table has different group ownerships as shown in Table 4.

In order to achieve group hierarchy in NIS+, we use the recursive group membership functionality in NIS+. First, we need to create NIS+ group as below. The first argument is the group name. Using NIS+ group, we create a network security officer role.

```
# nisgrpadm -c NSO.list.gmu.edu
  Group ''NSO.list.gmu.edu.'' created.
```

In order to assign a user to network security officer, we can use the following command. The first argument is a group name. The remaining argument is the name of the administrator. So we assign Dave to network security officer role.

```
# nisgrpadm -a NSO.list.gmu.edu Dave.list.gmu.edu
Added ''Dave.list.gmu.edu.'' to group ''NSO.list.gmu.edu.''
```

To remove members from NIS+ group, we can use the `-r` option as below.

```
# nisgrpadm -r NSO.list.gmu.edu Dave.list.gmu.edu
Removed ''Dave.list.gmu.edu.'' from group ''NSO.list.gmu.edu.''
```

As mentioned before, we use one of NIS+ features, a recursive group, to simulate the hierarchy between NIS+ groups. For example, the following statement add a recursive group, `JSO` as a member of `NSO`. Therefore, all members of junior security officer are also the member of network security officer group.

```
# nisgrpadm -a NSO.list.gmu.edu @JSO.list.gmu.edu
Added ''@JSO.list.gmu.edu.'' to group ''NSO.list.gmu.edu.''
```

Table 3 shows how we can construct hierarchy between NIS+ group. Alice is a member of senior group `SSO` and also a member of `JSO`, `ASO`, and `NSO` according to the hierarchical structure as shown in Fig. 4. Bob who is a member of `JSO` becomes a member of `ASO` and `NSO`. That is, this table implies that `SSO` is the senior-most group and `JSO` who is junior to `SSO` is senior to `ASO` and `NSO`. Assigning NIS+ groups as shown in Table 3 we can achieve the hierarchical

Table 3. *Simulating group hierarchy*

GROUP NAME	GROUP MEMBERS
SSO	Alice
JSO	Bob, @SSO
ASO	Chris, @JSO
NSO	Dave, @JSO

structure of NIS+ groups. We say a user is an *explicit* member of a group if the user is explicitly assigned as a member of the group. A user is an *implicit* member of a group if the user is an explicit member of some senior group. For example, Alice can be an *explicit* member of SSO, in which case she is also an *implicit* member of JSO, ASO, and NSO. We have showed how to construct hierarchical structure between NIS+ groups so far. Now we should set up the ownership and access rights for each table in NIS+ with hierarchical structure of NIS+ groups since the hierarchical NIS+ groups itself do not mean that senior group can do more than junior group(s). We need to modify group ownership of NIS+ table objects and set access rights for each table as shown in Table 4. That is, member users of a table's group owner can have special privileges to that object.

Firstly, we have to modify a table's group ownership using `nischgrp` command. For example, we use the following statement to change `networks` table's group ownership to NSO.

```
# nischgrp NSO.list.gmu.edu. networks.org_dir.list.gmu.edu.
```

Secondly, we modify access rights to an NIS+ table. It ensures that certain NIS+ group(s) can have special privilege over an NIS+ table. In order to change access rights to an NIS+ table, we use +/− operator and an indexed name with `nischmod` command. The following statement adds `read` and `modify` rights to group owner for an entry in the `networks.org_dir.list.gmu.edu.` table.

```
# nischmod g+rm networks.org_dir.list.gmu.edu.
```

In order to prevent other classes of principal such as `Nobody` and `World` from accessing the entry of an NIS+ table, we may need to remove the default access right from them using `nischmod` command with the − option. For example, we remove `modify, create`, and `destory` rights from group owner for an entry in the `networks.org_dir.list.gmu.edu.` table.

```
# nischmod g-mcd networks.org_dir.list.gmu.edu.
```

As shown in Table 4, we can add several junior administrators to the ASO group so that they can only modify the password and credential tables, but they would be unable to modify any other tables. And JSO can access all tables which ASO and JSO can access but cannot access `auto_master` table which only SSO can

Table 4. *Example of the selected table*

Table	GROUP	Access Rights			
		Nobody	**Owner**	**Group**	**World**
host table	JSO	r---	rmcd	rmcd	r---
passwd table	ASO	r---	rmcd	rmcd	r---
cred table	ASO	r---	rmcd	rmcd	r---
netmasks table	NSO	r---	rmcd	rmcd	r---
networks table	NSO	r---	rmcd	rmcd	r---
auto_master table	SSO	r---	rmcd	rmcd	r---

access. By setting access right like this table, we can simulate group hierarchy supporting inheritance of access rights from junior to senior.

We can use `nisls` and `niscat commands` to confirm the current configuration in NIS+. The following scripts shows the list of current NIS+ groups. Currently, NIS+ group consists of four admin groups (actually role) such as `SSO`, `JSO`, `ASO`, and `NSO`.

```
% nisls groups_dir

groups_dir.list.gmu.edu: SSO
JSO
ASO
NSO
```

The following scripts displays the description of NIS+ group NSO and it shows the list of all members such as Dave and all members of `JSO`.

```
%niscat -o NSO.groups_dir

Object Name   : NSO
Directory     : groups_dir.list.gmu.edu.
Owner         : list.list.gmu.edu.
Group         : NSO.list.gmu.edu.
Access Rights : ----rmcdr---r---
Time to Live  : 1:0:0
Creation Time : Thu Jul 30 03:31:55 1998
Mod. Time     : Thu Jul 30 07:12:31 1998
Object Type   : GROUP
Group Flags   :
Group Members :
            Dave.list.gmu.edu.
            @JSO.list.gmu.edu.
```

We can also check the properties of NIS+ table. This last scripts presents the NIS+ table `networks` which is table object and has `NSO` as its group owner, including access rights which are allowed group owner `NSO` to perform.

```
%niscat -o networks.org_dir

Object Name          : networks
Directory            : org_dir.list.gmu.edu.
Owner                : list.list.gmu.edu.
Group                : NSO.list.gmu.edu.
Access Rights        : ----rmcdrmcdr---
Time to Live         : 12:0:0
Creation Time        : Thu Jul 30 03:32:55 1998
Mod. Time            : Thu Jul 30 07:13:31 1998
Object Type          : TABLE
Table Type                 : networks_tbl
Number of Columns          : 4
Character Separator        : :
Search Path                :
```

In this section, we showed that we can construct the hierarchical structure between NIS+ groups assigning recursive groups as a NIS+ group and we can simulate group hierarchy supporting inheritance of access rights from junior to senior by setting access rights and appropriate group ownerships.

In order to make this work more convenient we developed two graphical user interfaces (GUIs) which interact with NIS+ command to add a user (or group) to NIS+ group and remove a user (or group) to NIS+ group. The graphical user interfaces are illustrated in Figs 5 and 6 and are called `NIS+ Administrator Tool 1` and `NIS+ Administrator Tool 2` respectively. `NIS+ Administrator Tool 1` is used to initiate NIS+ command. And `NIS+ Administrator Tool 2` is used to look up the NIS+ tables. These interfaces are convenient for administrative groups to do their tasks.[7]

5. Conclusion

In this paper, we described the overview of Network Information Services which allows networked machines to have a common interface regardless of the workstation that a user logs into. We have also described our experiment to provide a useful extension to the NIS+ group mechanism by means of recursive group and setting appropriate access rights and group ownerships. Our case study indicated that we can simulate hierarchical decentralized administration of NIS+ table. Also we could distribute *administrator's power* in NIS+ using Role-based administration.

In summary, this case study will be one step towards making current access control such as Role-based access control (RBAC) more acceptable to administrators as an enabling and empowering technology.

[7] Most of NIS+ commands which reveal the important credentials were blocked to prevent non-administrator from executing NIS+ commands.

Figure 5. User interface:TOOL 1.

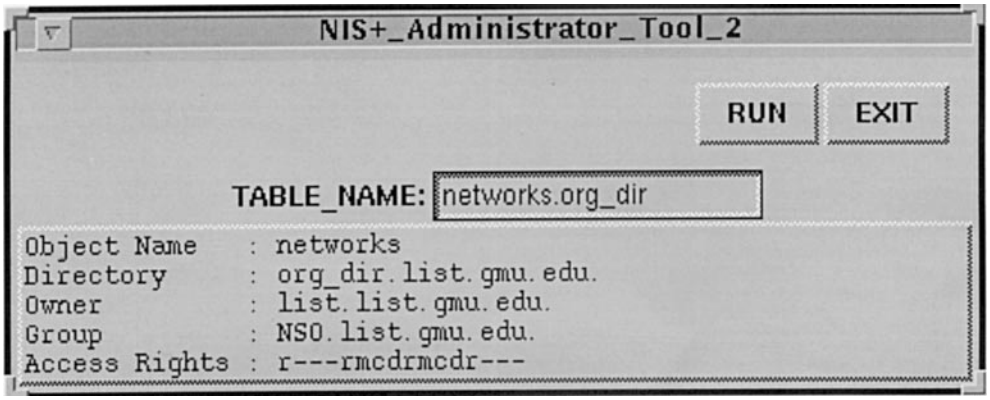
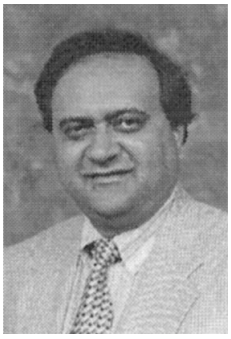


Figure 6. User interface:TOOL 2.

References

1. A. D. Birrell 1985. Secure communication using remote procedure calls. *ACM Transactions on Computer Systems*, **3**(1):1–14, 1992.
2. D. K. Hess, D. R. Safford and U. W. Pooch 1992. A Unix network protocol security study: Network information service. *Technical report, Texas A&M University*.
3. Sun Microsystems Inc. 1987. *XDR: External Data Representation Standard*. ARPA Network Information Center, June 1987. RFC 1014.
4. Sun Microsystems Inc. 1988. *RPC: Remote Procedure Call Protocol Specification*. ARPA Network Information Center, April 1988. RFC 1050.
5. R. Ramsey 1994. *All about Administering NIS+*. Prentice Hall.
6. R. Sandhu. Roles versus groups. In *Proceedings of the 1st ACM Workshop on Role-Based Access Control*. ACM, 1997 pp. 25–27.
7. R. S. Sandhu, E. J. Coyne, H. L. Feinstein and C. E. Youman 1996. Role-based access control models. *IEEE Computer*, **29**(2):38–47.

Gail-Joon Ahn received his BS degree in Computer Science from SoongSil University, Seoul, Korea in 1994 and MS degree in Computer Science from George Mason University at Fairfax, Virginia in 1996. He joined the Laboratory for Information Security Technology since 1996, and was actively involved in rearch in information security. During 1999 he received the doctoral fellowship to continue his PhD studies from School of Information Technology and Engineering, George Mason University. His research interests include access control, security, distributed objects, and secure information system.



Ravi Sandhu is professor of Information and Software Engineering at George Mason University, Fairfax, Virginia; and director of the Laboratory for Information Security Technology at GMU. His principal research and teaching interests are in information and systems security. Sandhu received PhD and MS degrees from Rutgers University, New Jersey, and BTech and MTech degrees from IIT Bombay and Delhi, India, respectively. Sandhu chairs ACM's Special Interest Group on Security Audit and Control.