CHAPTER 4

The AA Database

The AA database stores authentication and authorization information about each user that is allowed access to the network. User configuration information is typically similar between groups of users, so the AA database includes the concept, or feature, of groups. There is also an *unknown_user*, one that defines the parameters for any user who is not specifically listed in the database. Properly configured, the unknown-user feature can be used to allow users to automatically register themselves on the network.

The AA database implements the following concepts to identify users and enforce restrictions on services:

- Users and groups—Individuals and groups of individuals who are authorized to use a service or a set of services on the network
- Inheritance—The ability to derive attributes, such as authorization for a particular service, from other users or groups
- Qualification attributes—Attributes such as an expiration value that constrain or restrict services and the way they are used
- Authentication attributes—Attributes that the CiscoSecure UNIX server can use to enforce system-wide control of specific services

Users and Groups

When authorized users log into your network, they are immediately identified by the AA database as users of a service or a set of services. If you are administering the activity of many users on your network, however, the task of individually assigning all the necessary attributes to each user would be unmanageable.

One way to manage large numbers of users is to group them together according to the services they will use. Using the AA database, you can define each group and authorize it to use the appropriate set of services. You can then add each new user to the appropriate group.

You could restrict access to account information with explicit and implicit group assignments—for example, as follows:

- Local sales staff could be explicitly assigned to an account information group.
- External sales managers could be explicitly assigned to a district information group.
- External sales executives could be explicitly assigned to a complete sales information group.
- All other external users could be assigned to the unknown_user group that does *not* have access to any of the information.

With this kind of implicit and explicit grouping, you can also control the access of users to mission-critical network services. For example, rather than controlling the access to a database, you could control the ability of a user to log into a specified machine.

To further simplify the process of assigning access privileges to users, each group can be a member of some other group. In the above example, the complete sales information group might be a member of another larger group that has access to other services and accounting information.

Such hierarchical grouping can significantly simplify the task of ensuring a secure network in which users have easy access to necessary services and information, but no access to other services, which are unrelated to their jobs. Moreover, you can reliably and easily ensure the security of the entire network regardless of its size or complexity.

Inheritance

As an organization grows, hierarchical grouping of users becomes invaluable to system administrators. This hierarchy has very little value, however, unless it uses *inheritance*. Inheritance is the ability to assign attributes (such as the authorization to access specific services on a network) to a group or a user by copying attributes from another group or user.

Users and groups inherit attributes from their immediate parent entity, which means that they have the same attributes as the group from which they were derived. Members of the district information group, in the example in the previous section, “Users and Groups,” each have all the attributes assigned to that group. If they are not locally known users, they inherit the attributes of the unknown_user group.

The AA database provides a default user from which all users implicitly inherit attributes. Thus, if a user is not explicitly assigned to a particular group, the user inherits the attributes assigned to the default user. Where you explicitly specify attributes for a group, all members of that group inherit those attributes. Where attributes are specified for a user, these attributes supersede any settings that might have been inherited from either a group to which the user belongs or from the default user.

To achieve inheritance, CiscoSecure UNIX Server software first searches the specified user or group. If it does not find the attribute in question, it searches the parent entity. The search continues up the inheritance chain until the attribute is found. If the attribute is not found in the topmost parent, the default values are used (see the section “Implicit Declarations” later in this chapter).

To explicitly define an attribute, you must assign a value to it using an attribute-value pair. An attribute-value pair is a pair of strings of the form *attribute=value* , for example:

```
timeout=180
```

This attribute-value pair specifies a time-out period of three minutes.

Overriding Inherited Attribute Values

The availability of inherited service values can be overridden by prefixing the service specification with the keywords **prohibit** or **permit**. Although default permissions exist, you can explicitly prohibit or enable particular services using the **prohibit** or **permit** keywords. The **permit** keyword allows specified services; the **prohibit** keyword disallows specified services. Using these keywords together, you can construct “everything except” configurations. For example, the following configuration allows access from all services except X.25:

```
default service = permit
prohibit service = x25
```

Qualification Attributes

One technique you can use to ensure the security of your network is to qualify users when they attempt to log in or request a service. For example, you might know that your organization intends to employ several new people beginning on a particular date. Depending on your needs, you can immediately add these new users to the AA database and specify that they cannot log in until a specified date.

Qualification attributes can be associated with users, groups, and services. If a qualification attribute is found, then its condition must be matched or the operation in progress will fail. The following defined qualification conditions are supported:

- **expires**—No operation involving this entity can succeed after the date specified.
- **time**—No operation involving this entity can succeed outside the specified time or day period.
- **valid**—No operation involving this entity may succeed before the specified date.
- **allow** *expression expression*—Permit operations involving this entity when performed from a port or network-access-server combination that matches the specified regular expressions.
- **refuse** *expression expression*—Deny operations involving this entity when performed from a port or network-access-server combination that matches the specified regular expressions.

See the section The AA Database File later in this chapter for specific examples of using these qualifiers.

Authentication Attributes

Authentication attributes are attributes that are designed specifically to help you modify the network environment of users when they log in. The network environment includes access lists, IP address assignment pools, AppleTalk zone lists, specific route entries, and so forth. Some of the authentication attributes that are available are as follows:

- **inacl**—Modifier used with **service=ppp** and **protocol=ip**. Contains an inbound IP access list for SLIP or PPP/IP, for example, **inacl=3**. See the section “Authentication in the AA Database” later in this chapter for more information about authentication levels.

- **outacl**—Modifier used with **service=ppp** and **protocol=ip**. Contains an outbound IP access list for SLIP or PPP/IP, for example, **outacl=5**. See the section "Authentication in the AA Database" later in this chapter for more information about authentication levels.
- **addr-pool**—Modifier used with **service=ppp** and **protocol=ip**. Specifies the name of a local address pool from which to get the address of the remote host.
- **zonelist**—Modifier used with **service=arap**. Specifies an AppleTalk zonelist for the ARA protocol—for example, **zonelist=8**.
- **routing**—Modifier used with **service=slip**, **service=ppp**, and **protocol=ip**. Equivalent in function to the routing flag in SLIP and PPP commands. Can either be **true** or **false**.
- **timeout**—Modifier used with **service=arap**. The number of minutes before an ARA session disconnects—for example, **timeout=120**.
- **autocmd**—Modifier used with **service=shell** and **cmd=NULL**. Specifies a command to be automatically executed at EXEC startup—for example, **autocmd=telnet server1.foo.com**.
- **noescape**—Modifier used with **service=shell** and **cmd=NULL**. Specifies **noescape** (users cannot use the escape character), equivalent to the **noescape** option in the **username** configuration command. Can be either **true** or **false**—for example, **noescape=true**.
- **nohangup**—Modifier used with **service=shell** and **cmd=NULL**. Specifies a **nohangup** option, equivalent to the option for the **username** command. Can be either **true** or **false**, for example—**nohangup=false**.
- **priv-lvl**—Modifier used with **service=shell** and **cmd=NULL**. Specifies the privilege level for a command authorization. Can be any number from **0** to **15**—for example, **priv-lvl=15**.

If you set these attributes, their values are returned to the NAS on successful completion of an authentication exchange. These attributes allow system-wide controls to be administered from a central server.

The AA Database File

This section describes the format of the AA database. See the appendix “CiscoSecure UNIX Server File Formats and Syntax” for the database grammar and sample files.

Four basic components are defined in the AA database, as follows:

- Comments—Comment lines begin with the pound symbol (#). All characters from the # symbol up to and including the end of a line indicated by the newline character (\n) are treated as a comment and are ignored.
- Unknown_User—Defines the parameters and services to be used for any user whose configuration is not defined in the AA database.
- Group—Contains the parameters and services to be applied to a group of users. This allows you to specify the configuration for sets of users with a common parameter and service configuration.
- User—Specifies the parameters and services for a specific user. One of the parameters may be the group to which the user belongs, if any.

The task of administering a large number of users is simplified by the features of default services and attributes for user and user groups. User groups are useful where many users share the same or similar configuration information.

The format of the AA database is generally *name = value*. Some values allow additional subparameters to be specified and, in these cases, the subparameters are enclosed in curly braces ({}). Below is a simple example of an AA database showing the default user, one group, two users who belong to the group, and one user who does not:

```
# Sample AA Database 1
unknown_user = {
   password = system #Use the system’s password file (/etc/passwd)
}
group = staff {
   # password for staff who do not have their own
   password = des “sefjkAlM7zybE”
   service = shell {
     # allow any commands with any attributes
     default cmd = permit
     default attribute = permit
   }
}
user = joe {# joe uses the group password
```

```
   member = “staff”
}
user = pete {# pete has his own password
   member = “staff”
   password = des “alkd9Ujiqp2y”
}
user = anita {
   # Use the “default” user password mechanism defined above.
   service = shell {
     cmd = telnet {# allow telnet to any destination
     }
   }
}
```

The *Unknown _user* configuration element specifies the use of the system’s password-checking mechanism for authorization verification of any user who is not specifically listed in the AA database. For SunOS, this would involve checking passwords against those in the /etc/passwd file. A group staff has been defined, with a group DES-based password and authorization to use any commands with any arguments. Users joe and pete are both members of the staff group, although pete has his own password. The user anita is not a member of any group and is only authorized to use the **Telnet** command.

Implicit Declarations

In the absence of a default declaration, the following conditions apply:

- Users that are not explicitly declared in the database cannot be authenticated.
- Users that are not explicitly declared in the database are not authorized to use any service.

In the absence of a specification at the appropriate level, the following default conditions apply:

- In a **declaration**, all services not explicitly permitted are denied.
- In a **declaration**, all attributes not explicitly specified are denied.
- In a **declaration**, all qualifiers not explicitly specified are allowed.
- In a **service**, all attributes not explicitly permitted are denied.
- In a **shell service**, all commands not explicitly permitted are denied.

- In a **ppp service**, all protocols not explicitly permitted are denied.

Circular declarations are not supported and only a single default declaration is allowed.

Explicit Declarations

The most fundamental functions supported by the AA database are those that enable you to group specific authorizations inside relevant service authorizations. For example, the authorization of a Telnet command is expressed as follows:

```
service = shell {
   cmd = telnet {
          ...
   }
}
   ...
```

Similarly, protocol authorizations for the Point-to-Point Protocol (PPP) service are explicitly enclosed in the PPP service authorization, as shown in the following example:

```
service = ppp {
   protocol = ip {
   ...
   }
   protocol = ncp {
   ...
   }
}
```

You can set default permissions for services, protocols, and commands. Explicit restrictions are also permitted, allowing the creation of “everything except” configurations, for example:

```
   default service = permit
   prohibit service = x25
   ...
```

Also, because populations of users and groups are volatile, you can restrict such declarations with start dates and end dates, using a variety of formats, for example:

```
user = john {
   password = file /etc/passwd
```

```
   valid = “1 Sep 1995”
   expires = “September 30 95”
}
```

When a user or group has expired or is not yet valid, it is as though the relevant declarations were absent. The only exception is that declarations such as the following will not be faulted even when the administrator’s group is no longer valid:

```
member = administrators
```

Finally, unknown users (in other words, those not specifically listed in the AA database) are managed separately, which allows you to identically administer services and functions for all users, for example as follows:

```
unknown_user = {
   service = shell {
       ....
   }
   member = others
   default service = deny
}
```

The AA database allows for mandatory and optional attributes to be defined. You declare attributes as in the following example:

```
   service = shell {
       cmd = telnet {
           set addr = "192.168.151.20"
           set optional timeout = 1200
       }
```

In cases where the AA database is null (because the file is not specified, does not exist, or the database contains no valid declarations), the previously specified authentication and authorization defaults apply.

Example AA Database

The following is an example display of an AA database:

```
# CiscoSecure Example AA Database File
```

```
# October 2 1996
#
# Unless otherwise specified, the following are applied:
#       default service = deny
#       default protocol = deny
#       default attribute = deny
#       default cmd = deny
#

unknown_user = {
  # The default password is the one used by the system (i.e. /etc/passwd)
  password = system
  set autocmd = “telnet registration-server”
}

group = staff {
  password = des “sefjKaLm7zybE”
  privilege = clear “operator” 2
}

# The admin group is also a member of ‘staff’ and can enable to
# privilege level 15 with a DES password (a UNIX encrypted password.)
group = admin {
  member = staff
  password = des “Aj2pwjbnZlsoh”
  privilege = des “sefjKaLm7zybE” 15
}

# Fred uses an ‘skey’ password, is a member of ‘admin’, and can do
anything.
user = fred {
  member = admin
  password = skey
  password = chap “fred-chap”  # Fred’s CHAP password
  default service = permit
  default protocol = permit
  default cmd = permit
  default attribute = permit
}

# Joy can use any PPP protocol except IPX and can issue any exec command
# except the ‘enable’ command.
user = joy {
  member = staff
  password = clear “My ClearText Password”
```

```
  service = ppp {
    default protocol = permit
    prohibit protocol = ipx
  }
  service = shell {
    default cmd = permit
    prohibit cmd = enable
  }
}

# Tom can only run PPP/IP on NAS0 - NAS9, any tty port.
# However, he cannot run IPX on NAS12, any tty port.
user = tom {
  service = ppp {
    protocol = ip {
      allow “nas[0-9]” “tty.*”
    }
protocol = ipx {
        refuse "nas12" "tty.*"
        allow ".*" ".*"
}
# Ralph can run an EXEC, but he falls into the ‘default’ above,
# which specifies an autocommand.  His account is valid after 1 January
1996,
# and expires on 31 December 1996.
user = ralph {
  service = shell {
  }
  valid = “1 Jan 96”
  expires = “31 Dec 96”
}

# Frank can only start an EXEC session at night and on weekends.
user = frank {
  service = shell {
    default cmd = permit
    default attribute = permit
    time = Any 2300 - 0559
    time = Sat, Sun 0000 - 2359
  }
}

# Joe can run an EXEC, but only for the one year period specified.
user = joe {
  service = shell {
```

```
    # No cmd defaults to EXEC
  } from “1 Jan 96” until “31 Dec 96”
}

# Robert can start PPP/IP, but only if his machine uses IP address
# 131.108.12.3, and with the input and output access lists specified.
user = robert {
  service = ppp {
    protocol = ip {
      set addr = 131.108.13.3
      set inacl = 103
      set outacl = 105
    }
  }
}

# Anita can run PPP/IP, but gets an address from an IP address
# pool named ‘alternet’ on the NAS.  She may only connect on NAS1,
# any tty port.
user = anita {
  service = ppp {
    protocol = ip {
      set addr-pool = alternet
      allow “nas1” “tty.*”
    }
  }
}

# Sam can use ARAP, which may have an Appletalk access list 601
# applied.
user = sam {
  service = arap {
    set optional acl = 601
  }
}

# Bob can only issue the exec command “show users”
user = bob {
  service = shell {
    cmd = show {
      permit “users”
    }
  }
}
```

```
# Rob can only telnet only to hosts with IP addresses from
# 131.101.13.2 - 131.101.13.254 and issue any ‘show’ commands.
user = rob {
  service = shell {
    cmd = telnet {
      deny “131\.101\.13\.1”
      permit “131\.101\.13\.[0-9]+”
    }
    cmd = show {
      permit “.*”
    }
  }
}
```

Authentication in the AA Database

When users log in, they are identified first on the network access server and then by the CiscoSecure UNIX Server software.

The authentication function uses two keywords, **password** and **privilege**. The **password** keyword specifies the login password for a specific user. In the AA database, there is an entry for each user that contains the user’s password, for example as follows:

```
password = des “sefjKaLm7zybE”
```

The quoted string is a DES-encrypted password that users must provide before they can log into the network. A previous entry in the AA database identifies users by name. See the section “AA Database” in the appendix “CiscoSecure UNIX Server File Formats and Syntax” for a complete example.

Privilege Levels

The CiscoSecure UNIX Server software supports 16 privilege levels, which are numbered from 0 through15. The most privileged user, root for example, is a level 15 user, user EXEC is a level 1 user, and level 0 might be reserved for a guest who is only allowed to connect or disconnect.

Privilege levels offer a range of access from guest users to system administrators who are responsible for the network.

Supported Password Types

Both the **password** and **privilege** keywords allow an extensible range of authentication methods, and you can install additional authentication methods by reconfiguring the CiscoSecure server even while the server is running. In addition, authentication methods for PAP, CHAP, and ARA protocol can use the CiscoSecure server.

CiscoSecure UNIX Server software includes the following password support:

- DES
- Clear
- No password
- File
- System

You can also configure support for the following recognized authentication methods:

- PAP
- CHAP
- ARA protocol

For example, **skey** is used here as the authentication method for attempts to enable services at level 4:

```
user = fred {
   password  = des "sHki4ylq9"
   privilege = skey 4
   password  = arap "my ara secret"
   privilege = no_password 1
   privilege = clear "it's a secret" 15
   password  = chap chap.secret
....
}
```

The **privilege** keyword specifies the password that users must provide in order to use services of a specified security level, for example as follows:

```
privilege = des “sefjKaLm7zybE” 3
```

The quoted string specifies the password that allows this user to enable and use any service that requires a level 3 authorization. The password specified by the **privilege** keyword is often referred to as the enable password, which means that users must enter this password when they use the **enable** command to enable a service.

In some cases, you might want to specify a login password that is different from the privilege or enable password. Using different passwords for different security levels is an additional security measure to prevent unauthorized access to mission-critical information.

You can also use *inheritance* to restrict access to the privilege-level password by making it an attribute of a group and restrict the number of users who can use the privilege password, even if it is widely known. (See the section “Inheritance” earlier in the chapter.)

Password and Privilege Expiration

You can configure **password** and **privilege** attributes to expire. For example, the following DES-encrypted password is valid from March 1, 1996 until October 31, 1996:

```
user = tom {
        member = staff
       password = des "sefjKaLm7zybE" from "1 March 96" until "31 Oct 96"
}
```

You can also configure a warning period to alert a user of an impending expiration date whenever the user is authenticated. To do this, you need to add or modify the control file variable “config_warning_period.” For example, to configure a ten-day warning period, edit the CiscoSecure control file to include this variable:

```
NUMBER config_warning_period = 10;
```

When a user changes his or her password, CiscoSecure keeps the changed password in its internal database and also records it in the file specified by the control file variable “config_update_log_filename.” Using the -u command line option or the SIGUSR1 signal to update the AA database will cause the until date to be updated for any user who has changed his or her password. For more information, see the chapter “Using CiscoSecure UNIX Server Software. The new until date will be changed to be the date of the password change plus the number of days specified in the control file variable “config_expiry_period.”

Changing Passwords

Users can change their passwords when they log in. Passwords must be between 6 and 13 characters and contain at least one numeric and one alphabetic character. CiscoSecure UNIX Server software checks passwords when they are changed to make sure that easily guessed or deciphered passwords are not used.

Take the following steps to change a password when logging in:

Step 1 Connect to the network access server.

Step 2 Enter your username at the network access server prompt.

Step 3 Press **Return** at the prompt requesting you to enter a password.

Step 4 Enter **yes** at the prompt asking if you want to change your password.

Step 5 Enter your existing password at the prompt.

Step 6 Enter your new password at the prompt.

Step 7 Enter your new password a second time to verify that it is correct.

The password retries variable specifies the number of invalid password attempts a user can make before being faulted. The password retries variable is configured per network access server and is located in the 'config_nas_config' configuration section of the CiscoSecure control file.

This variable must be set to a number greater than 1 for the change password function to work. You can only change the password if you enter a return at the password prompt (thus entering a NULL password).

If the retry-count is set to 1, then the NULL password is counted as the one password entry attempt and you will be disconnected. In addition, the config_priv_level_for_own_CHPASS variable in the control file should be set to 1. This number is the privilege level at which a user can change his or her own password. When someone logs into a router, that user's default level is 1, so if this variable is not set to 1, the user will not be able to change his or her password.

Authorization in the AA Database

You can establish global default settings for the name of the network-access-server and port of the caller, as well as set them up for individual services, commands, and protocols. The following complex example illustrates the flexibility of this approach:

```
user = fred {
allow ".*" "tty.*"
refuse ".*" "async.*"
service = shell {
   cmd = telnet {
       allow nas1.* .*
   }
   refuse nas2.* ".*"
}
```

Time-of-day and day-of-week qualifications are also supported, allowing system administrators to control access to highly contended or expensive resources during periods of demand. For example, the following declaration allows the **Telnet** command to be used at any time on weekends and outside normal office hours Monday through Thursday, that is, from 5 p.m. until 9 a.m. the following morning (1700 to 0900):

```
service = shell {
   cmd = telnet {
       time = Mo, Th 1700 - 0900
       time = Sa, Su 0000 - 2359
   }
}
...
```

The CiscoSecure UNIX Server software also allows for multiple declarations of the same service, protocol, or command. Because each declaration can include different attributes and qualifications, administrators can place restrictions on users at certain times or under certain conditions, but without those restrictions in other circumstances.