Network Working Group
Request for Comments: 1492

C. Finseth
University of Minnesota

July 1993

An Access Control Protocol, Sometimes Called TACACS

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

Background

There used to be a network called ARPANET. This network consisted of end nodes (hosts), routing nodes (IMPs) and links. There were (at least) two types of IMPs: those that connected dedicated lines only and those that could accept dial up lines. The latter were called "TIPs."

People being what they were, there was a desire to control who could use the dial up lines. Someone invented a protocol, called "TACACS" (Terminal Access Controller Access Control System?), which allowed a TIP to accept a username and password and send a query to a TACACS authentication server, sometimes called a TACACS daemon or simply TACACSD. This server was normally a program running on a host. The host would determine whether to accept or deny the request and sent a response back. The TIP would then allow access or not, based upon the response.

While TIPs are -- shall we say? -- no longer a major presence on the Internet, terminal servers are. Cisco Systems terminal servers implement an extended version of this TACACS protocol. Thus, the access control decision is delegated to a host. In this way, the process of making the decision is "opened up" and the algorithms and data used to make the decision are under the complete control of whoever is running the TACACS daemon. For example, "anyone with a first name of Joe can only login after 10:00 PM Mon-Fri, unless his last name is Smith or there is a Susan already logged in."

The extensions to the protocol provide for more types of authentication requests and more types of response codes than were in the original specification.

The original TACACS protocol specification does exist. However, due to copyright issues, I was not able to obtain a copy of this document

and this lack of access is the main reason for the writing of this document. This version of the specification was developed with the assistance of Cisco Systems, who has an implementation of the TACACS protocol that is believed to be compatible with the original specification. To be precise, the Cisco Systems implementation supports both the simple (non-extended) and extended versions. It is the simple version that would be compatible with the original.

Please keep in mind that this is an informational RFC and does not specify a standard, and that more information may be uncovered in the future (i.e., the original specification may become available) that could cause parts of this document to be known to be incorrect.

This RFC documents the extended TACACS protocol use by the Cisco Systems terminal servers. This same protocol is used by the University of Minnesota's distributed authentication system.

1. Protocol Semantics

This section will describe the requests and responses. The following two sections will describe two different ways of encoding the protocol.

A request/response pair is the basic unit of interaction. In this pair, the client sends a request and the server replies with a response. All requests must be acknowledged with a response. This requirement implies that all requests can be denied, although it is probably futile to attempt to deny a "logout" request.

1.1 Connections

In some cases, a string of request/response pairs forms a larger unit, called a "connection."

There are three types of connections:

1) Authenticate only, no connection:

```
client:  sends an AUTH packet
server:  responds with a REPLY
```

2) Login connection:

```
        client:  sends a LOGIN packet
        server:  responds with a REPLY

        repeat zero or more times:
                client:  sends a CONNECT packet
                server:  responds with a REPLY

        client:  sends a LOGOUT packet
        server:  responds with a REPLY
```

3) SLIP connection:

```
        client:  sends a LOGIN packet
        server:  responds with a REPLY

        repeat zero or more times:
                client:  sends a CONNECT packet
                server:  responds with a REPLY

        client:  sends a SLIPADDR packet
        server:  responds with a REPLY

        repeat zero or more times:
                client:  sends a CONNECT packet
                server:  responds with a REPLY

        client:  sends a SLIPON packet
        server:  responds with a REPLY
        client:  sends a LOGOUT packet (immediate)
        server:  responds with a REPLY

        client:  sends a SLIPOFF packet
        server:  responds with a REPLY
```

1.2 Requests

This section lists the requests supported by the protocol. The responses are described in the later encodings sections.

AUTH(username, password, line, style)

This request asks for an authentication. The parameters are:

- the username
- the password
- an indication of which line the request is for, and
- a style of authentication

The username is a string that identifies the user. In principle, it can be of any length and contain any characters. In practice, it should be no longer than 128 characters and should contain only the ASCII characters "!" (33 decimal) through "~" (126 decimal), inclusive.

The password is a string that is used to authenticate the user identified by the username. In principle, it can be of any length and contain any characters. In practice, it should be no longer than 128 characters and should contain only the ASCII characters "!" (33 decimal) through "~" (126 decimal), inclusive.

The line is a non-negative decimal integer. If the client supports multiple physical access channels, this value identifies the particular channel. By convention, lines are numbered starting from one, although this should be taken with a grain of salt. For example, Cisco Systems' implementation uses zero to designate the console port, then continues with one for the "main" serial lines. Clients that support only one channel should use line zero.

The authentication style is a possibly empty string. It identifies the particular style of authentication to be performed. Its syntax and semantics are local.

Example:

AUTH("fin@unet.umn.edu", "fake-password", 0, "staff")

This specifies a username of "fin@unet.umn.edu" (which happens to be my e-mail address), a password, an indication that no line is associated with this request, and a style of "staff". The semantics for this style might be that I am required to be a staff member (in addition, of course, to supplying a valid username and password). The server would presumably consult an external database to verify the staff status.

As a local option, the implementation may choose to encode the style information by using alternate port numbers. E.g. port 4001 would mean style 1, 4002 would be style 2, etc.

Note that the AUTH request type cannot be sent using the UDP encoding.

LOGIN(username, password, line) returns (result1, result2, result3)

This request asks for an authentication and signals that -- if the authentication succeeds -- a login connection is starting. The parameters are:

- the username
- the password
- an indication of which line the request is for

The meanings of the input fields are the same as the AUTH request. If the request is successful, this request returns three result values in addition to the success status. The result values are non-negative integers. Their interpretation is local. For example, Cisco Systems terminal servers interpret result3 to be the identifier of a local access list to use for additional validation.

CONNECT(username, password, line, destinationIP, destinationPort) returns (result1, result2, result3)

This request can only be issued when the username and line specify

an already-existing connection. As such, no authentication is required and the password will in general be the empty string. It asks, in the context of that connection, whether a TCP connection can be opened to the specified destination IP address and port.

The return values are as for LOGIN.

SUPERUSER(username, password, line)

This request can only be issued when the username and line specify an already-existing connection. As such, no authentication is required and the password will in general be the empty string. It asks, in the context of that connection, whether the user can go into "super-user" or "enable" mode on the terminal server.

As an example of the flexibility inherint in this whole scheme, the TACACSD supplied by Cisco Systems ignores the username part and instead checks wether the password matches that of the special user "$enable$".

LOGOUT(username, password, line, reason)

This request can only be issued when the username and line specify an already-existing connection. As such, no authentication is required and the password will in general be the empty string. It indicates that the connection should be terminated (but see

SLIPON). It must be acknowledged, but the success/fail status of the acknowledgment is irrelevant. The reason value indicates why the connection is terminating. A null reason value is supplied when the connection is going into SLIP mode.

SLIPON(username, password, line, SLIPaddress) returns (result1, result2, result3)

This request can only be issued when the username and line specify an already-existing connection. As such, no authentication is required and the password will in general be the empty string. It asks, in the context of that connection, whether the specified SLIPaddress can be used for the remote end of the connection.

If the server replies with a success, the client can proceed to a

SLIPON request. (It need not do so right away, however.)

Note that semantics of "username" can get hairy. For example, the Cisco Systems implementation encodes information in this way:

- If the user just requested the default address be assigned, this field holds the username in lower case.

- If the user requested a specific IP address or host name for the SLIP connection, this field contains the requested host name in UPPER case.

If the server replies with a success, the client will immediately send a LOGOUT request. However, the connection will remain established until a SLIPOFF request is sent. No other authentication requests will be sent for that connection.

SLIPaddress specifies the IP address used by the remote host. If a SLIPADDR request has been made, it will be that address. Otherwise, it will be the default address assigned by the client (e.g., Cisco terminal server).

The return values are as for LOGIN.

SLIPOFF(username, password, line, reason)

This request can only be issued when the username and line specify an already-existing connection that is in "SLIP" mode. As such, no authentication is required and the password will in general be the empty string. It indicates that the connection should be terminated. It must be acknowledged, but the success/fail status of the acknowledgment is irrelevant. The reason value indicates why the connection is terminating.

2.0 UDP Encoding: TACACS

This section describes the UDP encoding of the requests that have just been described. It also describes the responses. This UDP encoding forms the basis of the historical TACACS protocol.

This protocol uses port 49. This assignment continues to be confirmed by the IANA in the Assigned Numbers RFCs. (I can't say

that it was assigned by the IANA as the assignment preceded the organization.)

The basic packet format is shown here. All multi-bytes values are in network byte order. Unless otherwise specified, all values given are in decimal. Unused fields should be set to zero, but the recipient should not depend on that setting.

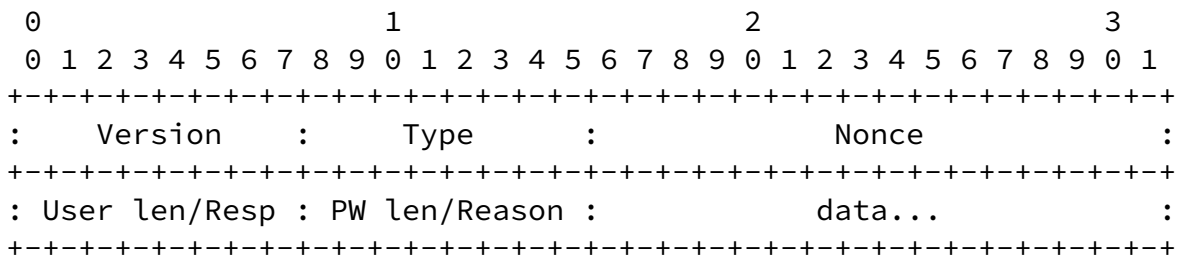
As was mentioned earlier, there are both simple and extended forms, of which the simple form is a proper subset of the extended form. A server should support both. I will describe both forms in parallel.

Simple Form

The fields are:

offset	length	field
0	1	version
1	1	type
2	2	nonce value
4	1	username length (to server) / response (to client)
5	1	password length (to server) / reason (to client)

in the usual packet layout format:

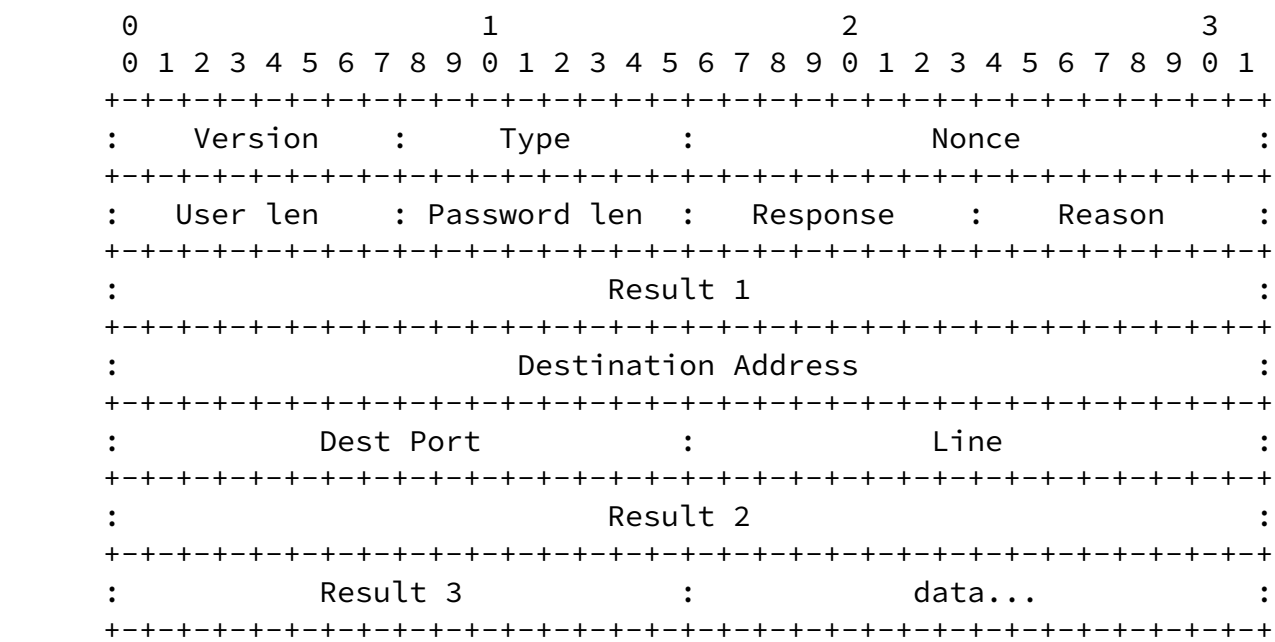


Extended Form

The fields are:

offset	length	field
0	1	version
1	1	type
2	2	nonce value
4	1	username length
5	1	password length
6	1	response
7	1	reason
8	4	result1
12	4	destination host, IP address
16	2	destination port
18	2	line
20	4	result2
24	2	result3
26	varies	data: username + password

in the usual packet layout format:



2.1 Fields

The VERSION field specifies the version number. It must be zero for simple or 128 (80 hexadecimal) for extended.

The TYPE field encodes the request type. Values are:

LOGIN	1	
RESPONSE	2	(server to client only)
CHANGE	3	
FOLLOW	4	
CONNECT	5	
SUPERUSER	6	
LOGOUT	7	
RELOAD	8	
SLIPON	9	
SLIPOFF	10	
SLIPADDR	11	

Other values below 128 are reserved for future use. Values from 128 to 255 are reserved for local use.

Note that the semantics of the CHANGE, FOLLOW, RELOAD requests have not been determined.

The NONCE field is set by the client to an arbitrary value. Its purpose is to allow clients that may have multiple outstanding requests to determine which request a response is for. The server must copy this value to the reply unaltered.

The USERNAME LENGTH field is set by the client to the length of the username in characters. Legal values are 0 to 255, inclusive. The server must copy this value to the reply unaltered.

The PASSWORD LENGTH is set by the client to the length of the password in characters. Legal values are 0 to 255, inclusive. The server must copy this value to the reply unaltered.

The RESPONSE field should be set by the client to zero. The server sets the value to one of:

value	meaning
1	accepted
2	rejected

Other values below 128 are reserved for future use. Values from 128 to 255 are reserved for local use.

The REASON field should be set by the client to zero, except for LOGOUT and SLIPOFF requests, which may use values of 4, 5, or 6. The server sets the value to one of:

value	meaning	notes
0	none	used for ACCEPTED or if the server is ornery
1	expiring	
2	password	
3	denied	
4	quit	user quit normally
5	idle	idle timeout
6	drop	carrier dropped
7	bad	too many bad passwords

The values from 4 to 6 will only be used for reasons for LOGOUT or SLIPOFF requests: they will not be returned by the server. Other values below 128 are reserved for future use. Values from 128 to 255 are reserved for local use.

The RESULT1 field should be set by the client to zero. For LOGIN or CONNECT requests, it is set by the server as specified in the request description. For all other requests, it should be set by the server to zero.

The DESTINATION HOST field is set by the client. On CONNECT, SLIPON, and SLIPOFF requests it specifies an IP address. It should be set to zero on all other requests. For SLIPON and SLIPOFF request, this value should be the IP address assigned to the line. For CONNECT requests, this value is the IP address of the host that the user is attempting to connect to. The server copies this value to the reply.

The DESTINATION PORT field is set by the client. On CONNECT requests it specifies the port number that the user is attempting to connect to. It should be set to zero on all other requests. The server copies this value to the reply.

The LINE field is set by the client to the line number that the

request is for. The server copies this value to the reply.

The RESULT2 field should be set by the client to zero. For LOGIN or CONNECT requests, it is set by the server as specified in the request description. For all other requests, it should be set by the server to zero.

The RESULT3 field should be set by the client to zero. For LOGIN or CONNECT requests, it is set by the server as specified in the request

description. For all other requests, it should be set by the server to zero.

The DATA field contains just the text of the username and password, with no separator characters (you use username length and password length to sort them out). The server does not copy the values to the reply. (However, the server does copy the username length and password length fields to the reply.) The username data may be in upper case: comparisons should be case-insensitive.

2.2 What a Client Does

The client must format and send a UDP request to port 49. It constructs the request by following these steps:

- set the version to 128
- set the type to that of the request
- set the nonce to a unique value that is different from all outstanding requests
- set the username length
- set the password length
- set the response to zero
- set the reason to zero (except for LOGOUT and SLIPOFF)
- set the result1 to zero
- if CONNECT, SLIPON, or SLIPOFF, set the destination address to the IP address, otherwise set it to zero
- if CONNECT, set the destination port to the port, otherwise set it to zero
- set the line
- set the result2 to zero
- set the result3 to zero
- copy the username to the location just after result3

- copy the password to the location just after the end of the username

Send the request. Wait for a reasonable (and hopefully configurable) period of time. If no response has been received, retry a reasonable (and hopefully configurable) number of times. Reasonable default wait times are 5 seconds and retries are 2.

If a response has been received, use the nonce value (and as many other fields as you like) to match it to an outstanding request. If there is no matching outstanding request, take appropriate (and hopefully configurable) action such as discarding and/or logging the packet.

If the response matches an outstanding request, examine the response and reason codes and take whatever action you deem correct. For

responses to LOGIN and CONNECT requests, also incorporate the result1, result2, and result3 values as you deem correct.

2.3 What a Server Does

Upon receipt of a UDP format request, the server examines the data in the request packet and determines its response. It constructs the reply by following these steps:

- set the version to 128
- set the type to RESPONSE (2)
- copy the nonce value
- copy the username length value
- copy the password length value
- set the response value to the desired response
- set the reason value to the desired reason
- if LOGIN or CONNECT, set the result1 else zero the result1
- copy the destination host value
- copy the destination port value
- copy the line value
- if LOGIN or CONNECT, set the result2 else zero the result2
- if LOGIN or CONNECT, set the result3 else zero the result3
- do NOT copy the username or password data

(As always, be liberal in what you expect and conservative in what

you send.) Send the response. Do not attempt to retry, as you have no basis for determining whether a retry is required. Any retries are up to the client. This, of course, implies that requests are idempotent. They aren't, of course, so the retries must be considered when trying to assemble requests into connections.

3.0 TCP Encoding

This section describes the TCP encoding of the requests and responses. This encoding is not compatible with the historical TACACS protocol. However, it is somewhat more "modern" in that it has been updated to provide for current feature needs.

This protocol does not use a reserved port. Instead, it must be possible to configure the ports used by both the the client and server.

The basic request format is shown here. The request consists of four lines of ASCII text. All numeric values are expressed in ASCII as decimal integers.


```
<version> <type> <parameters>
<username>
<password>
<line>
```

Each line in the example corresponds to one line of text. That is, the lines are separated with <CR>/<LF> (13/10 decimal) pairs. In no event may "bare" <CR> or <LF> characters appear within a field. In addition, <NUL> (0 decimal) characters may not be sent.

The <version> and <type> fields are separated with one or more <SPACE> (32 decimal) or <TAB> (9 decimal) characters.

The <parameters> field is optional. If present, it is separated from the <type> field and internal parameters separated from each other by or more <SPACE> or <TAB> characters. Any trailing <SPACE> or <TAB> characters present on this line should be ignored by the server: they should not be taken to imply a trailing empty field.

In theory there are no line length limits. In practice, lines should not exceed 255 characters (counting the <CR> and <LF>) and probably should be 80 characters or less.

3.1 Fields

The VERSION field specifies the version number. It must be 1. Other values below 128 are reserved for future use. Values from 128 to 255 are reserved for local use.

The TYPE field encodes the request type. Values are:

```
AUTH
LOGIN
CONNECT
SUPERUSER
LOGOUT
SLIPON
SLIPOFF
```

I.e., the keyword simply encodes itself. It must be in upper case. Keywords that begin with the letter "X" are reserved for local use.

The USERNAME field contains the text of the username. Leading and trailing <SPACE> or <TAB> characters are considered significant. The username data may be in upper case: comparisons should be case-insensitive.

The PASSWORD field contains the text of the password. Leading and

trailing <SPACE> or <TAB> characters are considered significant.

The LINE field is set by the client to the line number that the request is for.

3.2 Responses

Appendix E of STD 10, RFC 821 describes the general theory of reply codes. The this protocol follows the format described in that document. In a nutshell, replies are of the form:

<number> <text>

Where <number> is a three-digit decimal value and <text> is an arbitrary text string, presumably containing only printing text characters (<SP> (32 decimal) through "~" (126 decimal)). At least one <SP> (32 decimal) character separates the number from the text. A <CR>/<LF> sequence follows the text.

The three digit codes completely determine the response. The text should be considered an explanatory comment for human understanding. However, even without knowing all values, the first digit can be used to determine the overall nature of the response. The encodings are:

1 Positive Preliminary: the request is acceptable, but no action will be taken until an additional request is made (not used in this version of the protocol)

2 Positive Completion

3 Positive Intermediate: the request is acceptable so far, but has not been completely transferred (not used in this version of the protocol)

4 Transient Negative: the request is not acceptable for now. It is acceptable to retry, as another instance may have a different result.

5 Permanent Negative: the request is not acceptable

The text portion is optional (i.e., may be the empty string) and it describes the meaning of the message in human readable form.

While different server implementations will result in different messages, the following are suggested:

201 accepted: # # #

```
202 accepted, password is expiring: # # #
401 no response; retry
501 invalid format
502 access denied
```

The ": # # #" in the first two messages is the suggested way of returning the three result codes for LOGIN and CONNECT requests.

3.3 What a Client Does

The client opens a TCP connection to the locally-configured address and port. It sends the request by sending:

- the character "1"
- one or more <SPACE> or <TAB> characters
- the request type as an ASCII string
- if an AUTH, send one or more <SPACE> or <TAB> characters and the authentication style
- if a CONNECT, SLIPON, or SLIPOFF, send one or more <SPACE> or <TAB> characters and the IP address in dotted decimal notation
- if a CONNECT, send one or more <SPACE> or <TAB> characters and the port number in decimal
- a <CR>/<LF>
- the username (or hostname for SLIPADDR)
- a <CR>/<LF>
- the password
- a <CR>/<LF>
- the line
- a <CR>/<LF>

Then read one line from the connection and close the connection. This encoding lets TCP take care of waiting, retries, and matching up requests and responses.

Examine the response line and take whatever action you deem correct.

3.4 What the Server Does

The server waits on the locally-specified port for requests. When one is made, it reads four lines of input.

It examines the first line for a valid version number and request. It also records any optional parameters.

It uses the username, password, and line number along with any other information it deems fit to determine its response.

It then sends exactly one line of response, terminated by a <CR>/<LF>, and closes the connection.

4.0 Pros and Cons

Advantages to using the UDP format:

- lower overhead
- compatible with historical standard
- some existing equipment supports it

Advantages to using the TCP format:

- easier to implement, especially on machines with no or poor UDP support
- simpler, cleaner syntax
- potentially wider range of error codes, and support for temporary and negotiated authentication sequences

5.0 Security Notes

While the protocol itself has been described, there are a number of other considerations worth mentioning.

First, the protocol carries the username and password in clear text in either a single UDP packet or a TCP stream. As such, if an attacker is capable of monitoring that data, the attacker could capture username/password pairs. Implementations can take several steps to minimize this danger:

- Use point-to-point links where possible.

- Physically secure the transmission medium.

- If packets must traverse multiple network segments, use a secure routing subsystem. This implies:

  - Tight control over router configurations.
  - Tight control over routing protocols.
  - Avoid use of bridges, as they can be silently fooled into duplicating packets.

Second, this protocol potentially opens up a new way of probing

usernames and passwords. Thus, implementations may wish to have

servers:

- limit responses to a controlled list of clients,
- throttle the rate of responding to requests,
- log all failures (and possibly successes, too).

Third, this protocol essentially allows clients to offload accept/reject decisions to servers. While an obvious implementation would simply use the server's native login mechanism to make the determination, there is no reason to limit implementations to that mechanism. Servers could:

- use alternate lists of accounts (e.g., password files),
- use alternate mechanisms for accessing the accounts (e.g., a database, NIS),
- use alternate algorithms (e.g., SecureID cards),
- translate the request to another protocol and use that protocol to make the determination (e.g., Kerberos).

Fourth, the use of a "fanout" server (described in the next section) allows for:

- centralized logging of usage for attack analysis
- centralized policy:
  - ability to block selected specific users
  - ability to block selected user names (e.g., don't allow "root" or "guest")
  - ability to block poor passwords (e.g., none or weak)

6.0 Case Study

This section presents the basic steps used by the implementation at the University of Minnesota. Two examples will be used. The first is a basic terminal login. The second is a database access verification.

Usernames are in one of three forms:

First.M.Last-#@umn.edu

```
        First.M.Last-#
        user@host
```

A name in the first form is converted to one in the second.

A name in the second form is looked up in the University-wide directory system. If found, the associated electronic mail address is treated as if the third form was entered.

The third form specifics the name of a computer whose manager has agreed to perform validations and the name of an account on that computer.

The system that we use allows for many requesting clients (typically modem pools). Further, each client can support multiple distinct pools of users. For example, lines 1-20 could be general access, but lines 21-25 could be 800-numbers with a restricted set of valid users. The system supports this distinction by specifying which validation computers are legal for each modem pool.

6.1 Terminal Login

On the Cisco Terminal Server:

- accept a connection
- request a username and password
- pack the request into a UDP TACACS packet and send to the central fanout

Central Fanout:

- accept a request
- if the request is not in a valid format, return "nope"
- log the request
- if the source IP address is not in a list of valid clients, status = "nope"
- else if the username contains invalid characters, status = "nope"
- else

```
        if the username is of the form First.M.Last-#@umn.edu,
                convert to First.M.Last-#
        if the username is of the form First.M.Last-#,
                look up the name in the directory
```

```
                if the name is not found, status = "nope"
                otherwise, use the e-mail address as the username

        if the user is on a special "blocked" list, status = "nope"
                and send mail warning that access to a blocked
                account was attempted
        split the username into user and host parts
        if the host is not on a list of known servers,
                status = "nope"
        else if the host is not allowed to validate this type of
                request for this pool, status = "nope"

        now format a request for validation of the user and send it
        to the specified host
```



```
        if no response, status = "nope"
        otherwise set the status to the returned status
```

- log what response is going to be returned
- return the response

Validation Host:

This machine can run a "stripped down" version of the central fanout. It need perform no special validation or logging, with one exception.

- accept a request
- if the request is not in a valid format, return "nope"
- if the request is not from the central fanout, return "nope"
- figure the return status
- return the response

6.2 Database Access Verification

In this example, assume that a database is only to be accessed by faculty and staff.

Mainframe:

- the user is on the mainframe and makes a request
- the program requests username and password

```
- the program packs the request into a UDP TACACS packet and send to
  the central fanout

Central Fanout:

- accept a request
- if the request is not in a valid format, return "nope"
- log the request
- if the source IP address is not in a list of valid clients,
  status = "nope"
- else if the username contains invalid characters, status = "nope"
- else
        if the username is of the form First.M.Last-#@umn.edu,
                convert to First.M.Last-#
        if the username is of the form First.M.Last-#,
                look up the name in the directory
                if the name is not found, status = "nope"
                otherwise, use the e-mail address as the username
                   and obtain the staff status from the directory
        if the user is on a special "blocked" list, status = "nope"
                and send mail warning that access to a blocked
                account was attempted
```



```
        split the username into user and host parts
        if the host is not on a list of known servers,
                status = "nope"
        else if the host is not allowed to validate this type of
                request for this pool, status = "nope"

        now format a request for validation of the user and send it
        to the specified host

        if no response or status is "nope", status = "nope"

        else if the user originally gave a user@host mail address,
                do a directory lookup and obtain the staff status

        set the status to the staff status
- log what response is going to be returned
- return the response

Note that the validation host is unchanged.
```

References

[RFC821] Postel, J. "Simple Mail Transfer Protocol", STD 10, RFC 821, USC/Information Sciences Institute, August 1982.

[RFC1340] Reynolds, J. and J. Postel, "Assigned Numbers," STD 2, RFC 1340, USC/Information Sciences Institute, July 1992.

Anderson, Brian; Ruth, Greg; Ditmars, Peter; Eisner, Sharon; Delsignore, John (1985) TAC Access Control System Protocols, Second Edition: August 16 1985. BBN Tech Memo CC-0045.

Cisco Systems, Inc. (September 1992) Communications Server Configuration and Reference. Menlo Park, California.

Security Considerations

Security issues are the main topic of this memo.

Author's Address

Craig A. Finseth
Networking Services
Computer and Information Services
University of Minnesota
130 Lind Hall
207 Church St SE

Minneapolis MN 55455-0134

Phone: +1 612 624 3375
Fax: +1 612 626 1002

EMail: Craig.A.Finseth-1@umn.edu, or
       fin@unet.umn.edu