cisco Systems

Document Number	ENG-####
Revision	1.58
Author	Dave Carrel
Project Manager	Project manager name

TACACS+

Protocol Specification

Project Headline

This is the TACACS+ protocol specification. TACACS+ is an access control protocol used in cisco products. It provides the services of Authentication, Authorization and Accounting

Reviewers

Department	Name	Acceptance Date
Engineering	Name (Title)	Date
Engineering	Name (Title)	Date

Modification History

Rev	Date	Originator	Comment
1.49	26 Jan 95	Dave Carrel	imported from ascii version
1.50	28 Jan 95	Dave Carrel	changed msg and data lengths to two bytes
1.51	31 Jan 95	Dave Carrel	Just made it prettier.
1.52	2 Feb 95	Dave Carrel	enhanced the intro and added some attrs
1.54	13 April 95	Dave Carrel	added “event_id” and cleaned up text.
1.55	24 April 95	Dave Carrel	added protocol value “unknown”. Changed protocol values to lowercase.
1.57	19 May 95	Lol Grant	added zonelist attribute
1.58	31 May 95	Lol Grant	Bring revision numbers up to date

1.0 Introduction

The TACACS+ protocol is the next generation of TACACS. TACACS is a simple UDP based access control protocol originally developed by BBN for the MILNET. cisco has enhanced (extended) TACACS several times, and cisco's current implementation, based on the original TACACS, is referred to as XTACACS. The TACACS protocol is described in [2]. TACACS+ improves on TACACS and XTACACS by separating the functions of Authentication, Authorization and Accounting. It allows for arbitrary authentication exchanges which will allow any authentication mechanism to be utilized with TACACS+ clients. It is extensible to provide for site customization and future development features. And it uses TCP for reliable delivery. The protocol allows the TACACS+ client to request very fine grained access control and allows the server to respond to each component of that request.

The separation of authentication, authorization and accounting is a fundamental component of the design of TACACS+. The distinction between them is very important so this document will address each one separately. It is important to note that TACACS+ provides for all three, but an implementation or configuration is not required to employ all three. Each one serves a unique purpose that alone is useful, and together can be quite powerful.

TACACS+ uses TCP for it's transport. The server must listen at port 49 which is the "LOGIN" port. This port is reserved in the assigned numbers RFC for both UDP and TCP. Current TACACS and extended TACACS implementations use UDP port 49.

2.0 Definitions

2.1 Authentication

Authentication is the action of determining who a user (or entity) is. Authentication can take many forms. Traditional authentication utilizes a name and a fixed password. Most computers work this way, and TACACS+ can work this way. But fixed passwords have limitations, mainly in the area of security. Many modern authentication mechanisms utilize "one-time" passwords or a challenge-response query. TACACS+ is designed to support all of these, and just about any future mechanisms. Authentication generally takes place when the user first logs in to a machine or requests a service of it. Authentication is not mandatory, it is a site configured option. Some sites do not require it. Others require it only for certain services (see authorization below). Authentication also takes place when a user attempts to gain extra privileges, and must identify themselves as someone who possesses the required information (passwords, etc., ...) for those privileges.

2.2 Authorization

It is important to distinguish Authorization from Authentication. Authorization is the action of determining what a user is allowed to do. Generally authentication precedes authorization, but again, this is not required. An authorization request may indicate that the user is not authenticated (we don't know who they are). In this case it is up to the authorization agent to determine if an unauthenticated user is allowed the services in question. In TACACS+, authorization does not merely provide yes or no answers, but it may also customize the service for the particular user. Examples of when authentication would be performed are: When a user first logs in and wants to start a shell, or when a user starts PPP and wants to use IP on top of PPP with a particular IP address. The TACACS+ server might respond to these requests by allowing the service, but placing a time restriction on the login shell, or requiring IP access lists on the PPP connection. For a complete list of authorization attributes, see the authorization section below.

2.3 Accounting

Accounting is typically the third action after authentication and authorization. But again, neither authentication or authorization are required. Accounting is the action of recording what a user is doing, and/or has done. Accounting in TACACS+ can serve two purposes. It may be used to account for services used, such as in a billing environment. And it may also be used as an auditing tool for security services. To this end, TACACS+ supports three types of accounting records. Start records indicate that a service is about to begin. Stop records indicate that a service has just terminated. And Update records are intermediate notices that indicate that a service is still being performed. TACACS+ accounting records contain all the information used in the authorization records, and also contain accounting specific information such as start and stop times (when appropriate) and resource usage information. A complete list of accounting attributes is defined in the accounting section below.

2.4 Session

The concept of a session is used throughout this document. A TACACS+ session is a single authentication sequence, a single authorization exchange, or a stream of consecutive accounting messages. The session is important because the session id is used as a part of the encryption, and is used by both ends to separate packets belonging to different sessions. Multiple sessions may be supported simultaneously on a single TCP connection if both the server and client support this. The session is an administrative concept that is maintained between the TACACS+ client and server. It does not necessarily correspond to a given user or user action.

3.0 The TACACS+ packet header

All TACACS+ packets always begins with the following 12 byte header. The header is always cleartext and describes the remainder of the packet:

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|major  | minor  |                |                |                |
|version| version|      type      |     seq_no     |   encryption   |
+----------------+----------------+----------------+----------------+
|                                                                   |
|                            session_id                             |
+----------------+----------------+----------------+----------------+
|                                                                   |
|                              length                               |
+----------------+----------------+----------------+----------------+
```

`major_version` - This is the major TACACS+ version number. It is always 0xc.

`minor_version` - This is the minor TACACS+ version number. It is currently set to 0x0. The minor version number is intended to allow revisions to the TACACS+ protocol while maintaining backwards compatibility.

`type` - This is the packet type. Legal values are:

```
TAC_PLUS_AUTHEN          := 0x01      (Authentication)
TAC_PLUS_AUTHOR          := 0x02      (Authorization)
TAC_PLUS_ACCT            := 0x03      (Accounting)
```

`seq_no` - This is the sequence number of the current packet for the current session. The first TACACS+ packet in a session should have the sequence number 1 and each subsequent packet will increment the sequence number by one. Thus clients only send packets containing odd sequence numbers, and TACACS+ servers only send packets containing even sequence numbers.

`encryption` - This flag describes the encryption mechanism that is used for the body of the TACACS+ packet (the entire portion after the header). The encryption methods are described below. Legal values are:

```
TAC_PLUS_ENCRYPTED       := 0x00
TAC_PLUS_CLEAR           := 0x01
```

`session_id` - The Id for this TACACS+ session. The session id should be randomly chosen. This field does not change for the duration of the TACACS+ session. (If this value is not random, it will compromise the protocol's security.)

`length` - The total length of the TACACS+ packet body (not including the header). This value is in network byte order.

4.0 The TACACS+ packet body

The TACACS+ body types are defined in the above header. The remainder of this document will address the contents of the different TACACS+ bodies. The following general rules apply to all TACACS+ body types:

- The entire body is protected by the encryption mechanism indicated in the header.
- Any variable length data fields which are unused MUST have a length value equal to zero.
- Unused fixed length fields SHOULD have values of zero.
- All data and message fields in a TACACS+ packet MUST not be null terminated.
- All length values are unsigned and in network byte order.

4.1 Body Encryption

The body of a TACACS+ packets may be encrypted. The following sections describe the encryption mechanisms that are supported. Only one encryption mechanism SHOULD be used for a single session.

When the encryption mechanism relies on a secret key, it is refering to a shared secret value that is known to the client and the server. This document does not discuss the management and storage of those keys. It is an implementation detail of the server, as to whether it will maintain a single key, or a single key per client. For security reasons, the latter option should be available, but it is a site dependant decision as to whether the use of separate keys is appropriate.

4.1.1 TAC_PLUS_ENCRYPTED

The packet body is encrypted by XOR-ing it byte-wise with a pseudo random pad.

```
ENCRYPTED {data} == data ^ pseudo_pad
```

The pad is generated by concatenating a series of MD5 hashes (each 16 bytes long) and truncating it to the length of the input data.

```
pseudo_pad = {MD5_1 [,MD5_2 [ ... ,MD5_n]]} truncated to len(data)
```

The first MD5 hash is generated by concatenating the `session_id`, the secret key, the version number and the sequence number and then running MD5 over that stream. All of the input values are available in the packet header, except for the secret key which is a shared secret between the TACACS+ client and server. The session id is used in the byte order in which it appears in the TACACS+ header. (i.e. In network byte order, not host byte order).

Subsequent hashes are generated by using the same input stream, but concatenating the previous hash value at the end.

```
MD5_1 = MD5{session_id, key, version, seq #}
MD5_2 = MD5{session_id, key, version, seq #, MD5_1}
...
MD5_n = MD5{session_id, key, version, seq #, MD5_n-1}
```

4.1.2 TAC_PLUS_CLEAR

The entire packet body is in cleartext. Encryption and decryption is a null operation. This method should only be used for debugging. It does not provide data protection or authentication and is highly succeptible to packet spoofing.

5.0 Body types

All further discussions of TACACS+ packet bodies assumes that any encryption/decryption has already been performed. We are only concerned with the cleartext data.

There are several constant fields in many of the following bodies. A few merit mention in global scope. The `user` is the username or user id that is authenticated or being authenticated. The `port` is an ascii description of the port on which the user is connected. The remote address is a "best effort" description of the remote location from which the user has connected to the client. In many cases, the remote address will not be available or will be unreliable at best, but it may be useful when included. The `user_msg` is always the ascii input from the user. The `server_msg` is always used to hold a message that is intended to be presented to the user. In some contexts it may be optional as to whether to actually present it. The `data` field has several uses and is not used in all packets. In messages from the client, it can contains status information in an ascii message. In authentication messages from the server that do not have `TAC_PLUS_AUTHEN_STATUS_ERROR` status, the `data` field can hold authentication information. In all types of packets from the server when status is `TAC_PLUS_AUTHEN_STATUS_ERROR`, it holds a message suitable for display on an administrative console or log.

5.1 Authentication

TACACS+ authentication has three packet types: `START`, `CONTINUE` and `REPLY`. `START` and `CONTINUE` are always sent by the client and `REPLY` is always sent by the server. Authentication begins with the client sending a `START` message to the server. The `START` message describes the type of authentication to be performed, and may contain the username. The server sends a `REPLY`. The `REPLY` message indicates whether the authentication is finished, or whether it should continue. If the `REPLY` indicates that authentication should continue, then it will also indicate what new information is requested. The client will get that information and return it in a `CONTINUE` message. The server must always send a `REPLY` to both the `START` and the `CONTINUE` messages, with the only exception being if the client indicates an abort in the `CONTINUE`, in which case the session is immediately aborted. There are zero or more pairs of the client sending a `CONTINUE` and the server sending a `REPLY`.

The TACACS+ client can send two types of authentication packets to the server, a `START` and a `CONTINUE` packet. The `START` body is only ever sent as the first message in a TACACS+ authentication session, or as the packet immediately following a restart. (A restart may be requested by the server.) All other client messages are `CONTINUE` messages.

5.1.1 The authentication **`START`** packet body

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|    action      |    priv_lvl    |  authen_type   |     service    |
+----------------+----------------+----------------+----------------+
|    user len    |    port len    |  rem addr len  |  reserved len  |
+----------------+----------------+----------------+----------------+
|    user ...
+----------------+----------------+----------------+----------------+
|    port ...
+----------------+----------------+----------------+----------------+
|    rem addr ...
+----------------+----------------+----------------+----------------+
|    reserved ...
+----------------+----------------+----------------+----------------+
```

`action` - Describes the authentication action to be performed. Legal values are:

```
TAC_PLUS_AUTHEN_LOGIN          := 0x01
TAC_PLUS_AUTHEN_CHPASS         := 0x02
TAC_PLUS_AUTHEN_SENDPASS       := 0x03
```

`priv_lvl` - Indicates the privilege level that the user is authenticating as. All values, not specified here are client specified. Pre-defined values are:

```
TAC_PLUS_PRIV_LVL_MAX          := 0x0f
TAC_PLUS_PRIV_LVL_ROOT         := 0x0f
TAC_PLUS_PRIV_LVL_USER         := 0x01
TAC_PLUS_PRIV_LVL_MIN          := 0x00
```

`authen_type` - The type of authentication that is being performed. Legal values are:

```
TAC_PLUS_AUTHEN_TYPE_ASCII       := 0x01
TAC_PLUS_AUTHEN_TYPE_PAP         := 0x02
TAC_PLUS_AUTHEN_TYPE_CHAP        := 0x03
TAC_PLUS_AUTHEN_TYPE_ARAP        := 0x04
```

`service` - This is the service that is requesting the authentication. Legal values are:

```
TAC_PLUS_AUTHEN_SVC_NONE         := 0x00
TAC_PLUS_AUTHEN_SVC_LOGIN        := 0x01
TAC_PLUS_AUTHEN_SVC_ENABLE       := 0x02
TAC_PLUS_AUTHEN_SVC_PPP          := 0x03
TAC_PLUS_AUTHEN_SVC_ARAP         := 0x04
TAC_PLUS_AUTHEN_SVC_PT           := 0x05
TAC_PLUS_AUTHEN_SVC_RCMD         := 0x06
TAC_PLUS_AUTHEN_SVC_X25          := 0x07
```

`user` - This is the username. It is optional in this packet.

`port` - This is the ASCII name of the client port on which the authentication is taking place. The value of this field is client specified. (For example a cisco will use "tty10" for the tenth tty line and "Async10" for the tenth async interface.)

`rem_addr` - This is an ASCII string that describes the user's remote location. This field is option (since the information may not be available). This is intended to hold a network address if the user is connected over a network, a caller ID is the user is connected via ISDN or a POTS, or any other remote location information that is available. This field value is client specified.

`reserved` - This field is unused at this time.

5.1.2 The authentication CONTINUE packet body

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|          user_msg len           |            data len             |
+----------------+----------------+----------------+----------------+
|     flags      |  user_msg ...
+----------------+----------------+----------------+----------------+
|    data ...
+----------------+
```

`user_msg` - This field is the string that the user entered in response to the server_msg (see below).

`data` - This field carries information from the client itself (not from the user).

`flags` - This holds the bitmapped flags that modify the action to be taken. The following values are defined:

```
TAC_PLUS_CONTINUE_FLAG_ABORT     := 0x01
```

5.1.3 The authentication REPLY packet body

The TACACS+ server sends only one type of authentication packet (the REPLY packet) to the client. The REPLY packet body looks like this:

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|     status     |      flags     |        server_msg len           |
+----------------+----------------+----------------+----------------+
|           data len              |        server_msg ...
+----------------+----------------+----------------+----------------+
|           data ...
+----------------+----------------+
```

status - This is the current status of the authentication. Legal values are:

```
TAC_PLUS_AUTHEN_STATUS_PASS        := 0x01
TAC_PLUS_AUTHEN_STATUS_FAIL        := 0x02
TAC_PLUS_AUTHEN_STATUS_GETDATA     := 0x03
TAC_PLUS_AUTHEN_STATUS_GETUSER     := 0x04
TAC_PLUS_AUTHEN_STATUS_GETPASS     := 0x05
TAC_PLUS_AUTHEN_STATUS_RESTART     := 0x06
TAC_PLUS_AUTHEN_STATUS_ERROR       := 0x07
TAC_PLUS_AUTHEN_STATUS_FOLLOW      := 0x21
```

flags - This holds the bitmapped flags that modify the action to be taken. The following values are defined:

```
TAC_PLUS_REPLY_FLAG_NOECHO         := 0x01
```

server_msg - This is the message to be displayed to the user. This field is optional. If it exists, it should always be provided to the user if Status equals TAC_PLUS_AUTHEN_STATUS_PASS, TAC_PLUS_AUTHEN_STATUS_FAIL, TAC_PLUS_AUTHEN_STATUS_GETDATA, TAC_PLUS_AUTHEN_STATUS_GETUSER or TAC_PLUS_AUTHEN_STATUS_GETPASS. It is up to the client if it wants to provide the Server_msg when Status equals TAC_PLUS_AUTHEN_STATUS_RESTART, TAC_PLUS_AUTHEN_STATUS_ERROR or TAC_PLUS_AUTHEN_STATUS_FOLLOW.

data - This field holds data that is a part of the authentication exchange and is intended for the client, not the user. Specific uses of this field are described below.

If status equals TAC_PLUS_AUTHEN_STATUS_PASS, TAC_PLUS_AUTHEN_STATUS_FAIL, TAC_PLUS_AUTHEN_STATUS_ERROR or TAC_PLUS_AUTHEN_STATUS_FOLLOW, then the authentication session is terminated and the TCP stream may be closed. If status equals TAC_PLUS_AUTHEN_STATUS_PASS, the authentication has succeed. If status equals TAC_PLUS_AUTHEN_STATUS_FAIL, the authentication failed. If status equals TAC_PLUS_AUTHEN_STATUS_ERROR, then an unknown error occurred and the authentication does not succeed, but may not necessarily fail, depending on the client implementation. If status equals TAC_PLUS_AUTHEN_STATUS_ERROR, server_msg MAY contain a message for the user and the data field MAY contain a message for an administrative console or log. Displaying these messages is optional. If status equals TAC_PLUS_AUTHEN_STATUS_FOLLOW, then the client should contact another TACACS+ server and start the authentication over again, but the authentication does not fail. TAC_PLUS_AUTHEN_STATUS_FOLLOW will, in the future, allow for another TACACS+ server to be identified in the data field of the packet.

If `status` equals `TAC_PLUS_AUTHEN_STATUS_GETDATA`, `TAC_PLUS_AUTHEN_STATUS_GETUSER` or `TAC_PLUS_AUTHEN_STATUS_GETPASS`, then the authentication continues and the client must prompt the user with the `server_msg` for more information, and then send a `CONTINUE` packet. `TAC_PLUS_AUTHEN_STATUS_GETDATA` is the generic form. All three cause the same action to be performed, but in the case of `TAC_PLUS_AUTHEN_STATUS_GETUSER`, the client can know that the information that the user responds with is a username, and for `TAC_PLUS_AUTHEN_STATUS_GETPASS`, the user response is a password.

If the `TAC_PLUS_REPLY_FLAG_NOECHO` flag is set, then the user response should not be echoed as it is entered.

If the Status equals `TAC_PLUS_AUTHEN_STATUS_RESTART`, then the authentication sequence should be restarted with a new `START` packet from the client. This packet indicates that the current `authen_type` value (as specified in the `START` packet) is not acceptable for this session, but that others may be. The `TAC_PLUS_AUTHEN_STATUS_RESTART` packet may contain a list of valid `authen_type` values in the `data` portion of the packet. The `authen_type` values are a single byte in length so the `data_len` value indicates the number of `authen_type` values included. This packet is only currently intended for PPP authentication when multiple authentication mechanisms are available and can be negotiated between the client and the remote peer. Currently PPP supports PAP and CHAP, but more are proposed. If a client chooses not to accept `TAC_PLUS_AUTHEN_STATUS_RESTART` packets, then it should be TREATED as a `TAC_PLUS_AUTHEN_STATUS_FAIL` packet.

The client may prematurely terminate an authentication session by setting the `TAC_PLUS_CONTINUE_FLAG_ABORT` flag in the `CONTINUE` message. If this flag is set, the `data` portion of the message may contain an ascii message explaining the reason for the abort. The session is terminated and no `REPLY` message is sent. The TCP connection may be closed.

TACACS+ authentication can support three actions, specified in the `START` message. The actions are: `TAC_PLUS_AUTHEN_LOGIN`, `TAC_PLUS_AUTHEN_CHPASS` and `TAC_PLUS_AUTHEN_SENDPASS`.

`TAC_PLUS_AUTHEN_LOGIN` - is a standard login process where the server asks questions that must be answered by the client (who will typically pass them onto the user).

`TAC_PLUS_AUTHEN_CHPASS` - is a password change, and this works similarly to a `TAC_PLUS_AUTHEN_LOGIN` in that the server will simply ask whatever questions it requires in order to make the password change.

`TAC_PLUS_AUTHEN_SENDPASS` - is sort of the opposite of a `TAC_PLUS_AUTHEN_LOGIN`. In some cases, the client needs to know the user’s cleartext password or secret in order to perform the authentication. Examples of this are CHAP and ARAP, where the cleartext password is required to perform the challenge/response portion of the authentication exchange. Another example is the outbound authentication done in PPP when a remote end asks the local peer to authenticate itself. The `TAC_PLUS_AUTHEN_SENDPASS` exchange typically consists of a `START` message that includes the username, followed by a `REPLY` with `status` equal to either `TAC_PLUS_AUTHEN_STATUS_PASS` or `TAC_PLUS_AUTHEN_STATUS_FAIL`. However the exchange can be longer and more complex. When the `status` is `TAC_PLUS_AUTHEN_STATUS_PASS`, then the requested password will be in the `data` field of the `REPLY`.

5.2 Authorization

TACACS+ authorization is an extensible way of providing remote authorization services. The authorization request messages contain a fixed set of fields that describe the authenticity of the user or process, and a variable set of arguments that describes the services and options for which authorization is requested. The response contains a variable set of response arguments which can restrict or modify the clients actions. The arguments are attribute-value pairs. The arguments in both a request and a response can be specified as either mandatory or optional. An optional argument is one that may or may not be used, modified or even understood. A mandatory argument must be understood and used. This allows for extending the attribute list while providing secure backwards compatibility. An authorization session is defined as a single pair of messages, a request followed by a response.

5.2.1 The authorization `REQUEST` packet body

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|  authen method |    priv_lvl    |  authen type   | authen service |
+----------------+----------------+----------------+----------------+
|    user len    |    port len    |  rem addr len  |    arg cnt     |
+----------------+----------------+----------------+----------------+
|   arg 1 len    |   arg 2 len    |      ...       |   arg N len    |
+----------------+----------------+----------------+----------------+
|   user ...
+----------------+----------------+----------------+----------------+
|   port ...
+----------------+----------------+----------------+----------------+
|   rem addr ...
+----------------+----------------+----------------+----------------+
|   arg 1 ...
+----------------+----------------+----------------+----------------+
|   arg 2 ...
+----------------+----------------+----------------+----------------+
|   ...
+----------------+----------------+----------------+----------------+
|   arg N ...
+----------------+----------------+----------------+----------------+
```

authen_method - This indicates the authentication method used to acquire the user information.

```
TAC_PLUS_AUTHEN_METH_NOT_SET     := 0x00
TAC_PLUS_AUTHEN_METH_NONE        := 0x01
TAC_PLUS_AUTHEN_METH_KRB5        := 0x02
TAC_PLUS_AUTHEN_METH_LINE        := 0x03
TAC_PLUS_AUTHEN_METH_ENABLE      := 0x04
TAC_PLUS_AUTHEN_METH_LOCAL       := 0x05
TAC_PLUS_AUTHEN_METH_TACACSPLUS  := 0x06
TAC_PLUS_AUTHEN_METH_RCMD        := 0x20
```

`priv_lvl` - This field matches the priv_lvl field in the authentication section above. It indicates the users current privilege level.

`authen_type` - This field matches the authen type field in the authentication section above. It indicates the type of authentication that was performed.

`authen_service` - This field matches the service field in the authentication section above. It indicates the service through which the user authenticated.

`user` - This field contains the user's account name.

`port` - This field matches the port field in the authentication section above.

`rem_addr` - This field matches the remote addr field in the authentication section above.

`arg_cnt` - The number of authorization arguments to follow

`arg` - An attribute-value pair that describes the command to be performed. (see below)

The authorization arguments in both the `REQUEST` and the `RESPONSE` are attribute-value pairs. The attribute and the value are in a single ascii string and are separated by either a "=" (`0X3D`) or a "*" (`0X2A`). The equals sign indicates a mandatory argument. The asterisk indicates an optional one. Optional arguments are ones that may be disregarded by either client or server. Mandatory arguments require that the receiving side understands the attribute and will act on it. If the client receives a mandatory argument that it cannot oblige or does not understand, it MUST consider the authorization to have failed. It is legal to send an attribute-value pair with a NULL (zero length) value. The attribute-value strings are not NULL terminated, rather their length value indicates their end. The maximum length of an attribute-value string is 256 characters. The following attributes are defined:

Table 1: Attribute-value Pairs

ATTRIBUTE	VALUE(s)
`service`	a primary service. Specifying a service attribute indicates that this is an authorization request for starting that service. Currently known values are slip, ppp, arap, shell and system. This attribute MUST always be included.
`protocol`	a network protocol that is a subset of a service. This attribute must be specified when the service is PPP to indicate that a protocol is being brought up as a secondary service. An example would be any PPP NCP. Currently known values are "lcp", "ip", "ipx", "atalk", "vines" and "unknown".

Table 1: Attribute-value Pairs

ATTRIBUTE	VALUE(s)
`cmd`	a shell (exec) command. This indicates the command name for a shell command that is to be run. This attribute must be specified if service equals “shell”. A NULL value indicates that the shell itself is being referred to.
`cmd-arg`	an argument to a shell (exec) command. This indicates an argument for the shell command that is to be run. Multiple cmd-arg attributes may be specified, and they are order dependent.
`acl`	ascii number representing a connection access list. Used only when `service=shell` and `cmd=NULL`
`inacl`	ascii number for an interface input access list.
`outacl`	ascii number for an interface output access list.
`zonelist`	A numeric zonelist value. (Applicable to Appletalk only).
`addr`	a network address
`routing`	Boolean. Specifies whether routing information is to be propogated to, and accepted from this interface.
`route`	Indicates a route that is to be applied to this interface. Values must be of the form “<dst_address> <mask> <routing_addr>”. If routing_addr is missing, the current interface will be used.
`timeout`	an absolute timer for the connection (in minutes).
`idletime`	an idle-timeout for the connection (in minutes).
`autocmd`	an auto-command to run. Used only when `service=shell` and `cmd=NULL`
`noescape`	Boolean. Prevents user from using an escape character. Used only when `service=shell` and `cmd=NULL`
`nohangup`	Boolean. Do no disconnect after after an automatic command. Used only when `service=shell` and `cmd=NULL`
`priv_lvl`	privilege level to be assigned.
`remote_user`	remote userid (`authen_method` must = `TAC_PLUS_AUTHEN_METH_RCMD`)
`remote_host`	remote host (`authen_method` must = `TAC_PLUS_AUTHEN_METH_RCMD`)
`callback`	Indicates that callback should be done. Value is NULL, or a dialstring.

Table 1: Attribute-value Pairs

ATTRIBUTE	VALUE(s)
`callback-line`	The line number to use for a callback.
`callback-rotary`	The rotary number to use for a callback.
`...`	

For all boolean attributes, valid values are "`true`" or "`false`".

If a host is specified in a `cmd-arg` or `addr`, it is recommended that it be specified as an numeric address so as to avoid any ambiguities.

In the case of rcmd authorizations, the `authen_method` will be set to `TAC_PLUS_AUTHEN_METH_RCMD` and the `remote_user` and `remote_host` attributes will provide the remote user and host information to enable rhost style authorization. The response may wish to set a privilege level.

The `protocol` attribute is intended for use with PPP. When `service` equals "ppp" and `protocol` equals LCP, the message describes the PPP link layer service. For other values of `protocol`, this describes a PPP NCP (network layer service). A single PPP session can support multiple NCPs.

The attributes `addr`, `inacl`, `outacl`, `route` and `routing` may be used for all network protocol types that are supported. Their format and meaning is deterined by the values of the `service` or `protocol` attributes. Not all are necessarily implemented for any given network protocol.

5.2.2 The authorization RESPONSE packet body

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8

+----------------+----------------+----------------+----------------+
|    status      |    arg cnt     |         server_msg len          |
+----------------+----------------+----------------+----------------+
+             data len            |    arg 1 len   |    arg 2 len   |
+----------------+----------------+----------------+----------------+
|      ...       |   arg N len    |         server_msg ...
+----------------+----------------+----------------+----------------+
|   data ...
+----------------+----------------+----------------+----------------+
|   arg 1 ...
+----------------+----------------+----------------+----------------+
|   arg 2 ...
+----------------+----------------+----------------+----------------+
|   ...
+----------------+----------------+----------------+----------------+
|   arg N ...
+----------------+----------------+----------------+----------------+
```

`status` - This field indicates the authorization status

```
TAC_PLUS_AUTHOR_STATUS_PASS_ADD  := 0x01
TAC_PLUS_AUTHOR_STATUS_PASS_REPL := 0x02
TAC_PLUS_AUTHOR_STATUS_FAIL      := 0x10
TAC_PLUS_AUTHOR_STATUS_ERROR     := 0x11
```

`server_msg` -This field contains an ascii message that SHOULD be presented to the user by the client.

`data` - May be used for administrative messages.

`arg_cnt` - The number of authorization arguments to follow.

`arg` - An attribute-value pair that describes the command to be performed. (see below)

There are three `status` values. If the `status` equals `TAC_PLUS_AUTHOR_STATUS_FAIL`, then the appropriate action is to deny the user action. If the status equals `TAC_PLUS_AUTHOR_STATUS_PASS_ADD`, then the arguments specified in the request are authorized and the arguments in the response are to be used in addition to those arguments. If the `status` equals `TAC_PLUS_AUTHOR_STATUS_PASS_REPL` then the arguments in the request are to completely replaced by the arguments in the response. If the intended action is to approve the authorization with no modifications, then the `status` should be set to `TAC_PLUS_AUTHOR_STATUS_PASS_ADD` and the `arg_cnt` should be set to 0. `TAC_PLUS_AUTHOR_STATUS_ERROR` indicates an unknown error occurred on the server.

If an error is set, then the `server_msg` MAY contain ascii text to be presented to the user and `data` MAY contain ascii text to be presented on an administrative display, console or log. None of the arguments have any relevance if an error is set.

5.3 Accounting

TACACS+ accounting is very similar to authorization. The packet format is very similar. There is a fixed portion and an extensible portion. The extensible portion uses all the same attribute-value pairs that authorization uses, and adds several more.

5.3.1 The account `REQUEST` packet body

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|      flags     | authen method  |    priv_lvl    |  authen type   |
+----------------+----------------+----------------+----------------+
| authen service |    user len    |    port len    |  rem addr len  |
+----------------+----------------+----------------+----------------+
|    arg cnt     |   arg 1 len    |   arg 2 len    |      ...       |
+----------------+----------------+----------------+----------------+
|   arg N len    |    user ...
+----------------+----------------+----------------+----------------+
|   port ...
+----------------+----------------+----------------+----------------+
|   rem addr ...
+----------------+----------------+----------------+----------------+
|   arg 1 ...
+----------------+----------------+----------------+----------------+
|   arg 2 ...
+----------------+----------------+----------------+----------------+
|   ...
+----------------+----------------+----------------+----------------+
|   arg N ...
+----------------+----------------+----------------+----------------+
```

`flags` - This holds bitmapped flags.

```
TAC_PLUS_ACCT_FLAG_MORE          := 0x01
TAC_PLUS_ACCT_FLAG_START         := 0x02
TAC_PLUS_ACCT_FLAG_STOP          := 0x04
TAC_PLUS_ACCT_FLAG_WATCHDOG      := 0x08
```

All other fields are defined in the authorization and authentication sections above and have the same semantics.

The following new attributes are defined for TACACS+ accounting only. When these attribute-value pairs are included in the argument list, they should precede any attribute-value pairs that are defined in the authorization section above. The accounting attributes are:

Table 2: Accounting Attribute-value Pairs

ATTRIBUTE	VALUE(s)
`event_id`	Start and stop records for the same event will have matching (unique) `event_id`'s
`start_time`	The time the action started (in seconds since the epoch, 12:00am Jan 1 1970).
`stop_time`	The time the action stopped (in seconds since the epoch.)
`elapsed_time`	The elapsed time in seconds for the action. Useful when the device does not have real time.
`bytes`	The number of bytes transferred by this action
`bytes_in`	The number of input bytes transferred by this action
`bytes_out`	The number of output bytes transferred by this action
`packets`	The number of packets transferred by this action.
`packets_in`	The number of input packets transferred by this action.
`packets_out`	The number of output packets transferred by this action.
`status`	The numeric status value associated with the action. This is a signed four (4) byte word in network byte order. 0 is defined as success. Negative numbers indicate errors. Positive numbers indicate non-error failures. The exact status values may be defined by the client.
`err_msg`	An ascii string describing the status of the action.

All numeric values in an attribute-value string are provided as decimal ascii numbers.

5.3.2 The accounting **REPLY** packet body

The response to an accounting message is used to indicate that the accounting function on the server has returned and presumably has committed the record. This provides the client the best possible guarantee that the data is indeed logged.

```
 1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8  1 2 3 4 5 6 7 8
+----------------+----------------+----------------+----------------+
|         server_msg len          |            data len             |
+----------------+----------------+----------------+----------------+
|     status     |         server_msg ...
+----------------+----------------+----------------+----------------+
|     data ...
+----------------+
```

`status` - This is the return status. Values are:

```
TAC_PLUS_ACCT_STATUS_SUCCESS      := 0x01
TAC_PLUS_ACCT_STATUS_ERROR        := 0x02
```

`server_msg` - This is an ascii string that may be presented to the user. The decision to present this message is client specific.

`data` - This is an ascii string that may be presented on an administrative display, console or log. The decision to present this message is client specific.

If the `TAC_PLUS_ACCT_FLAG_MORE` flag is set, then this indicates that further accounting messages are pending as part of this session. If it is not set, then there is no further accounting information and the server may close the TCP connection.

The `TAC_PLUS_ACCT_FLAG_START` flag indicates that this is a start accounting message. Start messages should only be sent once when a service is started. The `TAC_PLUS_ACCT_FLAG_STOP` indicates that this is a stop record and that the service has terminated. The `TAC_PLUS_ACCT_FLAG_WATCHDOG` flag means that this is an update record. Update records are sent at the client's discretion when the service is still running.

6.0 Notes to Implementors

For those interested in integrating one-time password support into TACACS+ servers, there are some subtle gotchas to consider. For most cases, TACACS+ is designed to make this very straighforward. But two cases come to mind that are not so simple. One-time password support with ARAP and PPP's CHAP authenticaiton protocols is NOT straightforward, but there are not-so-difficult work arounds. The problem lies in the nature of ARAP and CHAP authentication. Both are doing a challenge-response protocol that requires a copy of the cleartext password to be stored at both ends. Unfortunately, due to their cryptographic nature, one-time password systems rarely can provide the cleartext version of the next password. A simple workaround is to have the user enter their username as a combination of the username and the one-time password, separated by a special character. Then a fixed password can be sent in the password field. The fixed password can be assigned on a per user basis or as a single site-wide password. For the separator character, cisco Systems has been using the '*' (asterix) character. After some head scratching, it was decided that it was the least likely character to be found in a username.

7.0 References

[1] Carrel, D., Grant, L.,

"The TACACS+ API Definition"

[2] Finseth, C.,
RFC 1492, "An Access Control Protocol, Sometimes Called TACACS",
July 1993.