Network Working Group
INTERNET-DRAFT
draft-grant-tacacs-02.txt

D. Carrel, Lol Grant
Cisco Systems,
January, 1997

Category: DRAFT

The TACACS+ Protocol
Version 1.78

Status of This Memo

Abstract

TACACS+ provides access control for routers, network access servers and other networked computing devices via one or more centralized servers. TACACS+ provides separate authentication, authorization and accounting services. This document describes the protocol that is used by TACACS+.

1. Introduction

The TACACS+ protocol is the latest 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 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 and by encrypting all traffic between the NAS and the daemon. It allows for arbitrary length and content 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 to ensure reliable delivery. The protocol allows the TACACS+ client to request very fine grained access control and allows the daemon 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. A very important benefit to separating authentication from authorization is that authorization (and per-user profiles) can be a dynamic process. Instead of a one-shot user profile, TACACS+ can be integrated with other negotiations, such as a PPP negotiation, for far greater flexibility. The accounting portion can serve to provide security auditing or accounting/ billing services.

TACACS+ uses TCP for its transport. The daemon should listen at port 49 which is the "LOGIN" port assigned for the TACACS protocol. This port is reserved in the assigned numbers RFC for both UDP and TCP. Current TACACS and extended TACACS implementations use port 49.

2. Technical Definitions

This section provides a few basic definitions that are applicable to this document.

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 also work this way. However, 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 should be powerful enough to handle 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 may also take 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.

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 authorization 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 over PPP with a particular IP address. The TACACS+ daemon might respond to these requests by allowing the service, but placing a time restriction on the login shell, or by requiring IP access lists on the PPP connection. For a list of authorization attributes, see the authorization section below.

Accounting

Accounting is typically the third action after authentication and authorization. But again, neither authentication nor 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. 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 list of accounting attributes is defined in the accounting section.

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 single accounting exchange.

The session concept is important because a session identifier is used as a part of the encryption, and it is used by both ends to distinguish between packets belonging to multiple sessions.

Multiple sessions may be supported simultaneously and/or consecutively on a single TCP connection if both the daemon and client support this. If multiple sessions are not being multiplexed over a single tcp connection, a new connection should be opened for each TACACS+ session and closed at the end of that session. For accounting and authorization, this implies just a single pair of packets exchanged over the connection (the request and its reply). For authentication, a single session may involve an arbitrary number of packets being exchanged.

The session is an operational concept that is maintained between the TACACS+ client and daemon. It does not necessarily correspond to a given user or user action.

NAS

A NAS is a Network Access Server. This is any device that provides access services. Nowadays, a NAS is typically more than just a terminal server. Terminal servers usually provide a character mode front end and then allow the user to telnet or rlogin to another host. A NAS may also support protocol based access services and may support PPP, ARAP, LAT, XREMOTE, and others.

MUST

This word means that the definition is an absolute requirement of the specification.

MUST NOT

This phrase means that the definition is an absolute prohibition of the specification.

SHOULD

This word, or the adjective "recommended", means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and carefully weighed

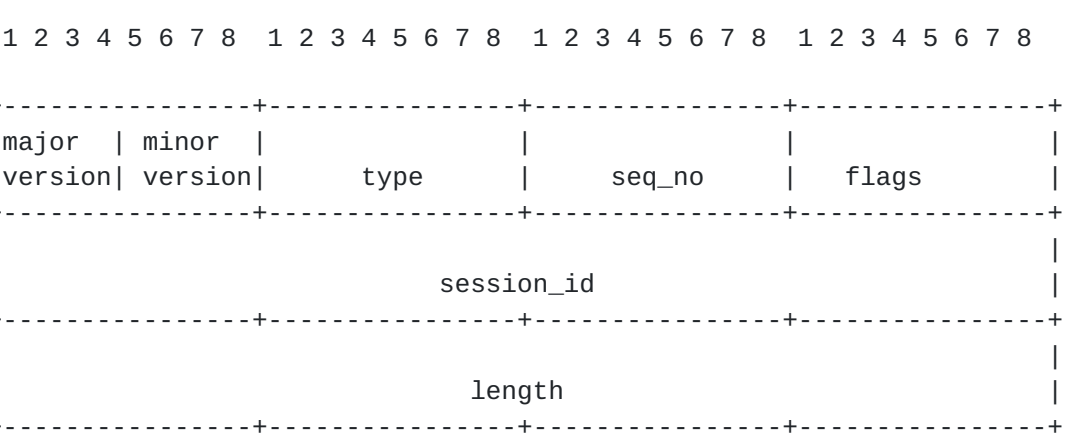
before choosing a different course.

MAY

This word, or the adjective "optional", means that this item is one of an allowed set of alternatives. An implementation which does not include this option MUST be prepared to interoperate with another implementation which does include the option.

3. The TACACS+ packet header

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



major_version

This is the major TACACS+ version number.

TAC_PLUS_MAJOR_VER := 0xc

minor_version

The minor TACACS+ version number. This is intended to allow revisions to the TACACS+ protocol while maintaining backwards compatibility.

Minor version 1 is currently defined for some commands. It may only be used for commands that explicitly call for it in this document. All other requests must use the default value.

TAC_PLUS_MINOR_VER_DEFAULT := 0x0

TAC_PLUS_MINOR_VER_ONE := 0x1

See the compatibility section at the end of the document.

When a daemon receives a packet with a minor_version that it does not support, it should return an ERROR status with the minor_version set to the closest supported value.

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 MUST 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+ daemons only send packets containing even sequence numbers.

The sequence number must never wrap i.e. if the sequence number 2^8-1 is ever reached, that session must terminate and be restarted with a sequence number of 1.

flags

This field contains various bitmapped flags.

The unencrypted flag bit says whether encryption is being used on the body of the TACACS+ packet (the entire portion after the header).

TAC_PLUS_UNENCRYPTED_FLAG := 0x01

If this flag is set, the packet is not encrypted. If this flag is cleared, the packet is encrypted.

Unencrypted packets are intended for testing, and are not recommended for normal use.

The single-connection flag:

TAC_PLUS_SINGLE_CONNECT_FLAG := 0x04

If a NAS sets this flag, this indicates that it supports multiplexing TACACS+ sessions over a single tcp connection. The flag need only be examined on the first two packets for any given connection since the single-connect status of a connection, once established, should not be changed. The connection must instead be closed and a new connection opened, if required.

If the daemon sets this flag in the first reply packet in response to the first packet from a NAS, this indicates its willingness to support single-connection over the current connection. The daemon may set this flag even if the NAS does not set it, but the NAS is under no obligation to honor it.

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 a cryptographically strong random number, it will compromise the protocol's security. [6])

length

The total length of the TACACS+ packet body (not including the header). This value is in network byte order. Packets are never padded beyond this length.

4. The TACACS+ packet body

The TACACS+ body types are defined in the packet 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.

- There should be no padding in any of the fields or at the end of a packet.

5. Body Encryption

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

When the encryption mechanism relies on a secret key, it is referring to a shared secret value that is known to both the client and the daemon. This document does not discuss the management and storage of those keys. It is an implementation detail of the daemon and client, as to whether they will maintain only one key, or a different key for each client or daemon with which they communicate. For security reasons, the latter options should be available, but it is a site dependent decision as to whether the use of separate keys is appropriate.

The encrypted flag field may be set as follows:

TAC_PLUS_UNENCRYPTED_FLAG == 0x0

In this case, 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.

Whenever used in this document, MD5 refers to the "RSA Data Security, Inc. MD5 Message-Digest Algorithm" as specified in [3].

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 those input values are available in the packet header, except for the secret key which is a shared secret between the TACACS+ client and daemon.

The version number is the one byte combination of the major and minor version numbers.

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 of the input stream.

MD5_1 = MD5{session_id, key, version, seq_no}

MD5_2 = MD5{session_id, key, version, seq_no, MD5_1}

....

MD5_n = MD5{session_id, key, version, seq_no, MD5_n-1}

TAC_PLUS_UNENCRYPTED_FLAG == 0x1

In this case, the entire packet body is in cleartext. Encryption and decryption are null operations. This method should only be used for debugging. It does not provide data protection or authentication and is highly susceptible to packet spoofing. Implementing this encryption method is optional.

NOTE: implementations should take care not to skip decryption simply because an incoming packet indicates that it is not encrypted.

After a packet body is decrypted, the lengths of the component values in the packet should be summed and checked against the cleartext datalength value from the header. Any packets which fail this check should be discarded and an error signalled. Commonly such failures may be expected to be seen when there are mismatched keys between the NAS and the TACACS+ server.

If an error must be declared but the type of the incoming packet cannot be determined, a packet with the identical cleartext header but with a sequence number incremented by one and the length set to zero may be returned to indicate an error.

6. Body types

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

There are several constant fields in many of the following bodies. A few merit mention here as they apply to most packet bodies.

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 rem_addr 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 but is not used in all packets.

6.1. 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 daemon.

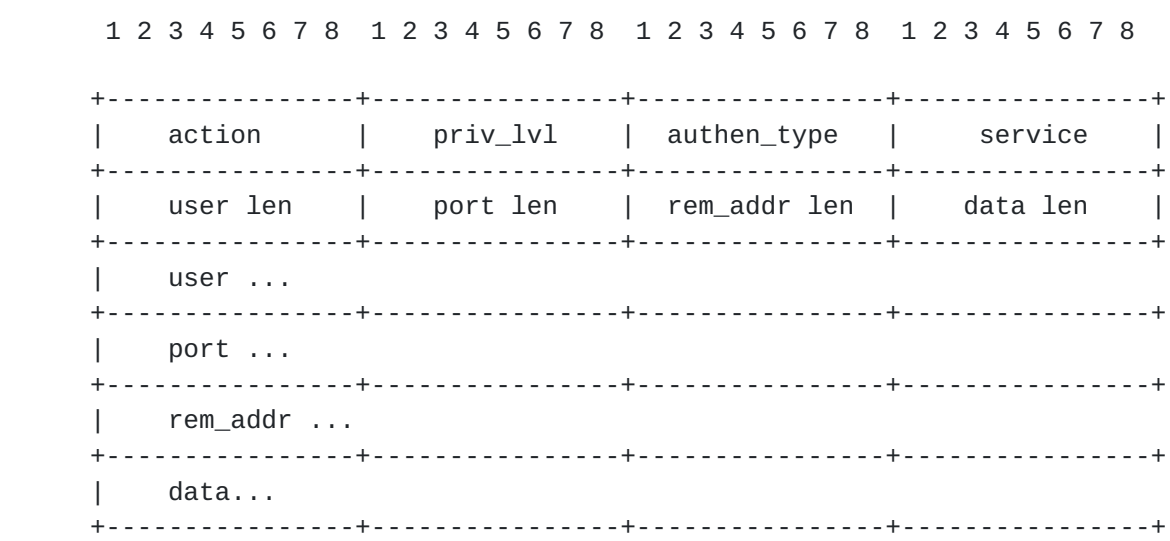
Authentication begins with the client sending a START message to the daemon. The START message describes the type of authentication to be performed, and may contain the username and some authentication data. The START packet is only ever sent as the first message in a TACACS+ authentication session, or as the packet immediately following a restart. (A restart may be requested by the daemon in a REPLY packet). A START packet always has seq_no equal to 1.

In response to a START packet, the daemon 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 daemon MUST always send a REPLY to both the START and the CONTINUE messages, the only exception being if the client indicates an abort in the CONTINUE, in which case the session is immediately aborted.

Thus, there are zero or more pairs of packets where the client sends a CONTINUE and the daemon sends a REPLY.

The authentication START packet body



Packet fields are as follows:

action

This 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 (deprecated)

TAC_PLUS_AUTHEN_SENDAUTH := 0x04

priv_lvl

This indicates the privilege level that the user is authenticating as. Privilege levels are ordered values from 0 to 15 with each level representing a privilege level that is a superset of the next lower value. If a NAS client uses a different privilege level scheme, then mapping must be provided. 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

TAC_PLUS_AUTHEN_TYPE_MSCHAP := 0x05

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

TAC_PLUS_AUTHEN_SVC_NASI        := 0x08

TAC_PLUS_AUTHEN_SVC_FWPROXY     := 0x09
```

The ENABLE service refers to a service requesting authentication in order to grant the user different privileges. This is comparable to the Unix "su(1)" command. A service value of NONE should only be used when none of the other service values are appropriate.

user

The username. It is optional in this packet.

port

The ASCII name of the client port on which the authentication is taking place. The value of this field is client specific. (For example, Cisco uses "tty10" to denote the tenth tty line and "Async10" to denote the tenth async interface).

rem_addr

An ASCII string that describes the user's remote location. This field is optional (since the information may not be available). It is intended to hold a network address if the user is connected via a

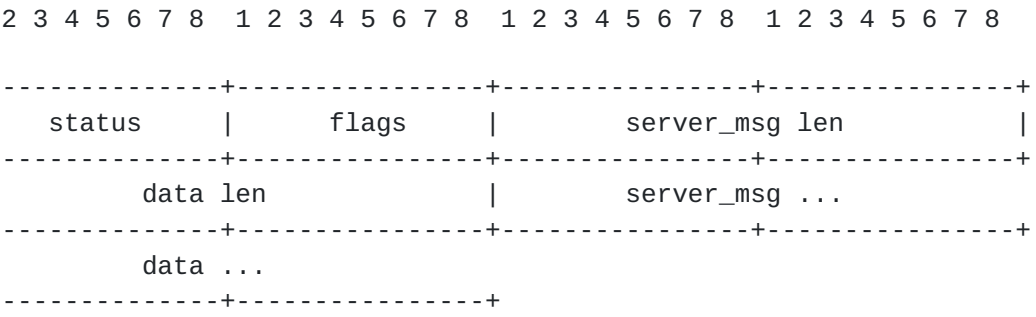
network, a caller ID is the user is connected via ISDN or a POTS, or any other remote location information that is available. This field value is client specified.

data

This field is used to send data appropriate for the action and authen_type. It is described in more detail below.

7. The authentication REPLY packet body

The TACACS+ daemon sends only one type of authentication packet (a REPLY packet) to the client. The REPLY packet body looks as follows:



status

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

Bitmapped flags that modify the action to be taken. The following values are defined:

TAC_PLUS_REPLY_FLAG_NOECHO := 0x01

server_msg

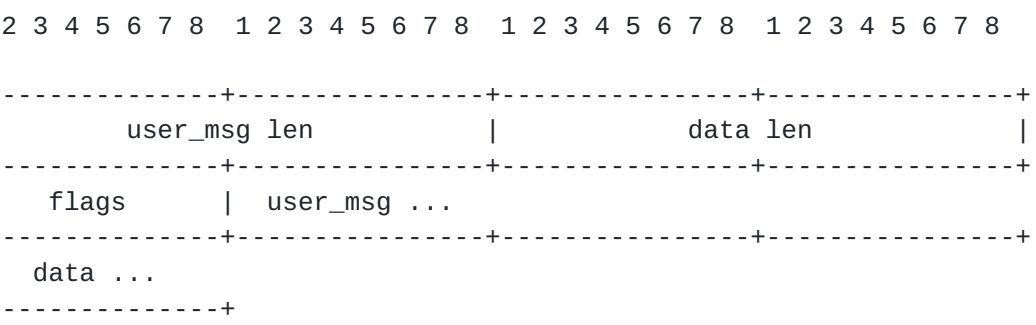
A message to be displayed to the user. This field is optional. If it exists, it is intended to be presented to the user.

data

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

8. The authentication CONTINUE packet body

This packet is sent from the NAS to the daemon following the receipt of a REPLY packet.



user_msg

This field is the string that the user entered, or the NAS provided on behalf of the user, in response to the server_msg from a REPLY packet.

data

This field carries information that is specific to the action and the authen_type for this session. Valid uses of this field are described below.

flags

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

TAC_PLUS_CONTINUE_FLAG_ABORT := 0x01

9. The authentication process

The flavor of the authentication is determined by the action and the authen_type fields in the START packet. First we should discuss some general fields that apply to all flavors of authentication exchanges. The user and data fields in the START packet are defined below for each flavor.

The priv_lvl, service, port and rem_addr in the START packet are all provided to help identify the conditions on the NAS. In the CONTINUE packet, the user_msg and data fields are defined below for each flavor. For all REPLY packets, the server_msg may contain a message to be displayed to the user, but the data field usage varies and is described below.

The descriptions below first describe "normal" authentication where, in response to a START packet, the daemon either sends a request for more information (GETDATA, GETUSER or GETPASS) or a termination (PASS or FAIL). The actions and meanings when the daemon sends a RESTART, ERROR or FOLLOW are common and are described further below.

When the REPLY status equals TAC_PLUS_AUTHEN_STATUS_GETDATA, TAC_PLUS_AUTHEN_STATUS_GETUSER or TAC_PLUS_AUTHEN_STATUS_GETPASS, then authentication continues and the server_msg may be used by the client to prompt the user for more information. The client MUST then return a CONTINUE packet containing the requested information in the user_msg field.

TAC_PLUS_AUTHEN_STATUS_GETDATA is the generic request for more information. 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, that the user response represents a password.

If the TAC_PLUS_REPLY_FLAG_NOECHO flag is set in the REPLY, then the user response should not be echoed as it is entered. The data field is only used in the REPLY where explicitly defined below.

9.0.1. Enable Requests

```
action      = TAC_PLUS_AUTHEN_LOGIN

priv_lvl    = implementation dependent

authen_type = not used

service     = TAC_PLUS_AUTHEN_SVC_ENABLE
```

This is an ENABLE request, used to change the current running privilege level of a principal. The exchange MAY consist of multiple messages while the daemon collects the information it requires in order to allow changing the principal's privilege level. This exchange is very similar to an Inbound ASCII login (which see).

In order to readily distinguish enable requests from other types of request, the value of the service field MUST be set to TAC_PLUS_AUTHEN_SVC_ENABLE when requesting an ENABLE. It MUST NOT be set to this value when requesting any other operation.

9.0.2. Inbound ASCII Login

```
        action      =   TAC_PLUS_AUTHEN_LOGIN

        authen_type =   TAC_PLUS_AUTHEN_TYPE_ASCII
```

This is a standard ASCII authentication. The START packet may contain the username, but need not do so. The data fields in both the START and CONTINUE packets are not used for ASCII logins. There is a single START followed by zero or more pairs of REPLYs and CONTINUEs, followed by a terminating REPLY (PASS or FAIL).

9.0.3. Inbound PAP Login

```
        action        =   TAC_PLUS_AUTHEN_LOGIN

        authen_type   =   TAC_PLUS_AUTHEN_TYPE_PAP
```

```
minor_version =   0x1
```

The entire exchange MUST consist of a single START packet and a single REPLY. The START packet MUST contain a username and the data field MUST contain the PAP ASCII password. A PAP authentication only consists of a username and password [4]. The REPLY from the daemon MUST be either a PASS or FAIL.

9.0.4. Inbound CHAP login

```
action        =   TAC_PLUS_AUTHEN_LOGIN

authen_type   =   TAC_PLUS_AUTHEN_TYPE_CHAP

minor_version =   0x1
```

The entire exchange MUST consist of a single START packet and a single REPLY. The START packet MUST contain the username in the user field and the data field will be a concatenation of the PPP id, the challenge and the response.

The length of the challenge value can be determined from the length of the data field minus the length of the id (always 1 octet) and the length of the response field (always 16 octets).

To perform the authentication, the daemon will run MD5 over the id, the user's secret and the challenge, as defined in the PPP Authentication RFC [4] and then compare that value with the response. The REPLY from the daemon MUST be a PASS or FAIL.

9.0.5. Inbound MS-CHAP login

```
action        =   TAC_PLUS_AUTHEN_LOGIN

authen_type   =   TAC_PLUS_AUTHEN_TYPE_MSCHAP

minor_version =   0x1
```

The entire exchange MUST consist of a single START packet and a single REPLY. The START packet MUST contain the username in the user field and the data field will be a concatenation of the PPP id, the MS-CHAP challenge and the MS-CHAP response.

The length of the challenge value can be determined from the length of the data field minus the length of the id (always 1 octet) and the length of the response field (always 49 octets).

To perform the authentication, the daemon will use a combination of MD4 and DES on the user's secret and the challenge, as defined in [7] and then compare the resulting value with the response. The REPLY from the daemon MUST be a PASS or FAIL.

9.0.6. Outbound MS-CHAP request

```
action        = TAC_PLUS_AUTHEN_SENDAUTH

authen_type   = TAC_PLUS_AUTHEN_TYPE_MSCHAP

minor_version = 0x1
```

This is used when the NAS needs to provide MS-CHAP authentication credentials to the remote PPP peer. The entire exchange MUST consist of a single START packet and a single REPLY. The START packet MUST contain the username in the user field and the data field will be a concatenation of the PPP id and the challenge.

The length of the challenge value can be determined from the length of the data field minus the length of the id (always 1 octet). The daemon will use MD4 and DES to process the user's secret and the challenge, as defined in [7].

The REPLY from the daemon MUST be a PASS or FAIL. If the status is PASS, then the data field MUST contain the 49-octet output, in which 24 octets are MD4 output for the Microsoft LAN Manager compatible challenge response, 24 octets are MD4 output for the Microsoft Windows NT compatible challenge response and 1 octet is the flag to determine which part of the response packet should be utilized.

9.0.7. Inbound ARAP login

```
action        =   TAC_PLUS_AUTHEN_LOGIN

authen_type   =   TAC_PLUS_AUTHEN_TYPE_ARAP

minor_version =   0x1
```

The entire exchange MUST consist of a single START packet and a single REPLY. The START packet MUST contain the username in the user field and the data field will be a concatenation of the NAS's challenge to the remote peer (8 octets) the remote peer's challenge to the NAS (8 octets) and the remote peer's response to the NAS's challenge (8 octets).

The daemon must run DES encryption over both the challenges using the user's secret as the DES key, as described in the ARAP specification [5]. For a successful authentication, the encrypted NAS challenge MUST be identical to the peer's response. The REPLY from the daemon MUST be a PASS or FAIL. The encrypted peer challenge (8 octets) is returned in the data field of a REPLY packet if the status is set to PASS.

9.0.8. Outbound PAP request

```
action        =   TAC_PLUS_AUTHEN_SENDAUTH

authen_type   =   TAC_PLUS_AUTHEN_TYPE_PAP

minor_version =   0x1
```

This is used when the NAS needs to provide PAP authentication credentials to the remote PPP peer. The entire exchange MUST consist of a single START packet and a single REPLY. The START packet contains a username in the user field. A REPLY with status set to PASS MUST contain a cleartext password in the data field. Caution is urged when using this. By sending a cleartext password to the NAS, that password will then be passed to the remote PPP peer. It should be ensured that the provided password can never be used to authenticate back to the NAS. Use of this is discouraged, but supported for complete interoperability with the PPP protocol.

9.0.9. Outbound CHAP request

```
action        = TAC_PLUS_AUTHEN_SENDAUTH

authen_type   = TAC_PLUS_AUTHEN_TYPE_CHAP

minor_version = 0x1
```

This is used when the NAS needs to provide CHAP authentication

credentials to the remote PPP peer. The entire exchange MUST consist of a single START packet and a single REPLY. The START packet MUST contain the username in the user field and the data field will be a concatenation of the PPP id and the challenge.

The length of the challenge value can be determined from the length of the data field minus the length of the id (always 1 octet). The daemon will run MD5 over the id, the user's secret and the challenge, as defined in the PPP Authentication RFC [4].

The REPLY from the daemon MUST be a PASS or FAIL. If the status is PASS, then the data field MUST contain the 16 octet MD5 output

9.0.10. Outbound ASCII and ARAP request

```
action      =   TAC_PLUS_AUTHEN_SENDAUTH

authen_type =   TAC_PLUS_AUTHEN_TYPE_ASCII

authen_type =   TAC_PLUS_AUTHEN_TYPE_ARAP
```

This is an error. This action is not supported for ASCII logins and in not needed for ARAP since ARAP authentication is already a two way protocol.

9.0.11. ASCII change password request

```
action      =   TAC_PLUS_AUTHEN_CHPASS

authen_type =   TAC_PLUS_AUTHEN_TYPE_ASCII
```

This exchange consists of multiple messages while the daemon collects the information it requires in order to change the user's password. It is very similar to an ASCII login. The status value TAC_PLUS_AUTHEN_STATUS_GETPASS MUST only be used when requesting the "new" password. It MAY be sent multiple times. When requesting the "old" password, the status value MUST be set to TAC_PLUS_AUTHEN_STATUS_GETDATA.

9.0.12. PPP change password request

```
action      =   TAC_PLUS_AUTHEN_CHPASS

authen_type =   TAC_PLUS_AUTHEN_TYPE_PAP

authen_type =   TAC_PLUS_AUTHEN_TYPE_CHAP
```

This is never valid. The PPP protocol does not support password changing.

9.0.13. ARAP change password request

```
action      =   TAC_PLUS_AUTHEN_CHPASS

authen_type =   TAC_PLUS_AUTHEN_TYPE_ARAP
```

This exchange consists of a single START packet and a single REPLY. The START packet MUST contain the username and the data field contains both the old and the new passwords encrypted (**FORMAT NOT KNOWN AT THIS TIME **). The reply is a PASS or FAIL and the data field is unused.

10. Aborting a session

The client may prematurely terminate a 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.

There are three other possible return status values that can be used in a REPLY packet. These can be sent regardless of the action or authen_type. Each of these indicates that the TACACS+ authentication session should be terminated. In each case, the server_msg may contain a message to be displayed to the user.

When the status equals TAC_PLUS_AUTHEN_STATUS_FOLLOW the packet indicates that the TACACS+ daemon requests that authentication should be performed with an alternate daemon. The data field MUST contain ASCII text describing one or more daemons. A daemon description appears like this:

```
[@<protocol>@]<host>[@<key>]
```

The protocol and key are optional. The protocol can describe an

alternate way of performing the authentication, other than TACACS+. If the protocol is not present, then TACACS+ is assumed.

Protocols are ASCII numbers corresponding to the methods listed in the authen_method field of authorization packets (defined below). The host is specified as either a fully qualified domain name, or an ASCII numeric IP address specified as octets separated by dots (`.').

If a key is supplied, the client MAY use the key in order to authenticate to that host. If more than one host is specified, they MUST be separated by an ASCII <CR> (0x0D).

Use of the hosts in a TAC_PLUS_AUTHEN_STATUS_FOLLOW packet is at the discretion of the TACACS+ client. It may choose to use any one, all or none of these hosts. If it chooses to use none, then it MUST treat the authentication as if the return status was TAC_PLUS_AUTHEN_STATUS_FAIL.

While the order of hosts in this packet indicates a preference, but the client is not obliged to use that ordering.

If the status equals TAC_PLUS_AUTHEN_STATUS_ERROR, then the host is indicating that it is experiencing an unrecoverable error and the authentication should proceed as if that host could not be contacted. The data field may contain a message to be printed on an administrative console or log.

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 REPLY 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 REPLY 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. This also requires future PPP authentication extensions which have not yet been passed through the IETF. If a client chooses not to accept the TAC_PLUS_AUTHEN_STATUS_RESTART packet, then it should be TREATED as if the status was TAC_PLUS_AUTHEN_STATUS_FAIL.

11. Authorization

TACACS+ authorization is an extensible way of providing remote

authorization services. An authorization session is defined as a single pair of messages, a REQUEST followed by a RESPONSE.

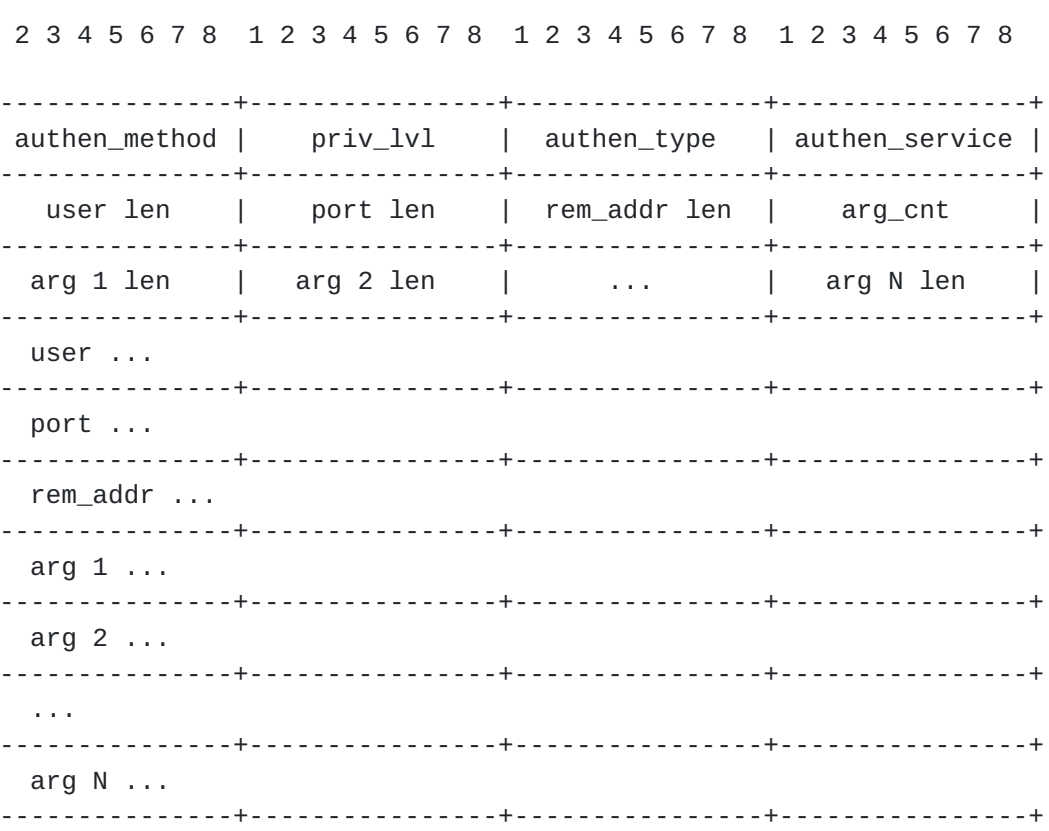
The authorization REQUEST message contains a fixed set of fields that describe the authenticity of the user or process, and a variable set of arguments that describes the services and options for which authorization is requested.

The RESPONSE contains a variable set of response arguments (attribute-value pairs) which can restrict or modify the clients actions.

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 by the recipient.

A mandatory argument MUST be both understood and used. This allows for extending the attribute list while providing secure backwards compatibility.

11.1. The authorization REQUEST packet body



authen_method

This indicates the authentication method used by the client 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_GUEST      := 0x08

   TAC_PLUS_AUTHEN_METH_RADIUS     := 0x10

   TAC_PLUS_AUTHEN_METH_KRB4       := 0x11

   TAC_PLUS_AUTHEN_METH_RCMD       := 0x20
```

KRB5 and KRB4 are kerberos version 5 and 4. LINE refers to a fixed password associated with the line used to gain access. LOCAL is a NAS local user database. ENABLE is a command that authenticates in order to grant new privileges. TACACSPLUS is, of course, TACACS+. GUEST is an unqualified guest authentication, such as an ARAP guest login. RADIUS is the Radius authentication protocol. RCMD refers to authentication provided via the R-command protocols from Berkeley Unix. (One should be aware of the security limitations to R-command authentication.)

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 rem_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 daemon. 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.

Attribute-value strings are not NULL terminated, rather their length value indicates their end. The maximum length of an attribute-value string is 255 characters. The following attributes are defined:

12. Table 1: Attribute-value Pairs

service

The primary service. Specifying a service attribute indicates that this is a request for authorization or accounting of that service. Current values are "slip", "ppp", "arap", "shell", "tty-daemon", "connection", "system" and "firewall". This attribute MUST always be included.

protocol

a protocol that is a subset of a service. An example would be any PPP NCP. Currently known values are "lcp", "ip", "ipx", "atalk", "vines", "lat", "xremote", "tn3270", "telnet", "rlogin", "pad", "vpdn", "ftp", "http", "deccp", "osicp" and "unknown".

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 identifier for an interface input access list.

outacl

ASCII identifier for an interface output access list.

zonelist

A numeric zonelist value. (Applicable to AppleTalk only).

addr

a network address

addr-pool

The identifier of an address pool from which the NAS should assign an address.

routing

A boolean. Specifies whether routing information is to be propagated 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 a

<routing_addr> is not specified, the resulting route should be via the requesting peer.

timeout

an absolute timer for the connection (in minutes). A value of zero indicates no timeout.

idletime

an idle-timeout for the connection (in minutes). A value of zero indicates no timeout.

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 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 have the value TAC_PLUS_AUTHEN_METH_RCMD)

remote_host

remote host (authen_method must have the value TAC_PLUS_AUTHEN_METH_RCMD)

callback-dialstring

Indicates that callback should be done. Value is NULL, or a dial-string. A NULL value indicates that the service MAY choose to get the dialstring through other means.

callback-line

The line number to use for a callback.

callback-rotary

The rotary number to use for a callback.

nocallback-verify

Do not require authentication after callback.

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

value of NULL means an attribute with a zero length string for its value i.e. cmd=NULL is actually transmitted as the string of 4 characters "cmd=".

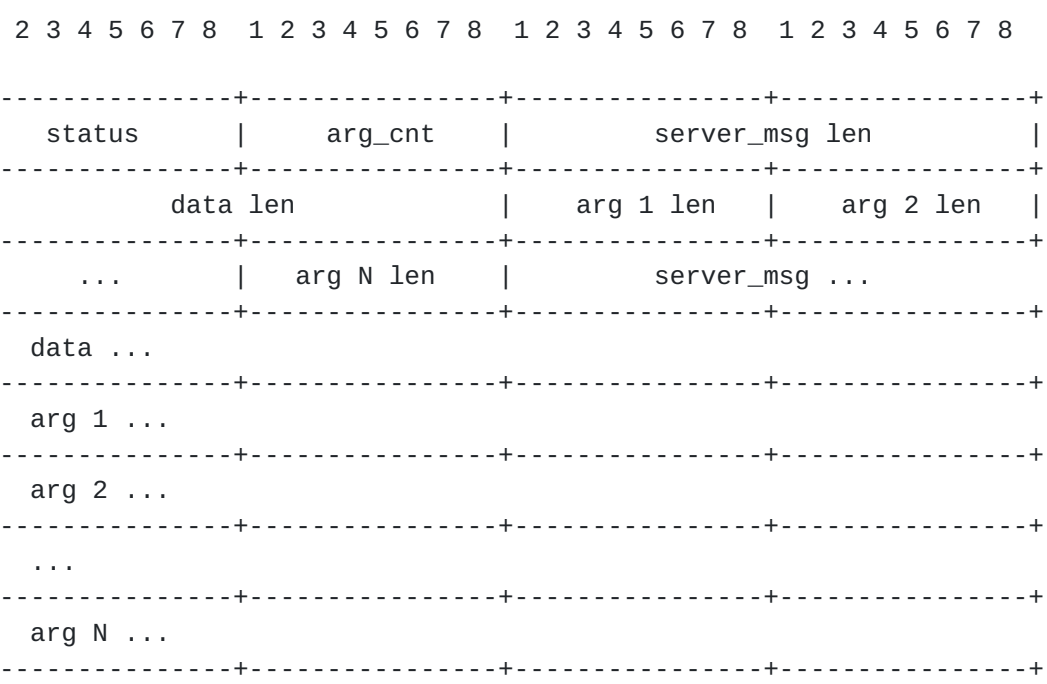
If a host is specified in a cmd-arg or addr, it is recommended that it be specified as a numeric address so as to avoid any ambiguities.

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 request that a privilege level be set for the user.

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 determined by the values of the service or protocol attributes. Not all are necessarily implemented for any given network protocol.

12.1. The authorization RESPONSE packet body



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

TAC_PLUS_AUTHOR_STATUS_FOLLOW := 0x21

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.

arg_cnt

The number of authorization arguments to follow.

arg

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

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 be 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.

A status of TAC_PLUS_AUTHOR_STATUS_ERROR indicates an error occurred on the daemon.

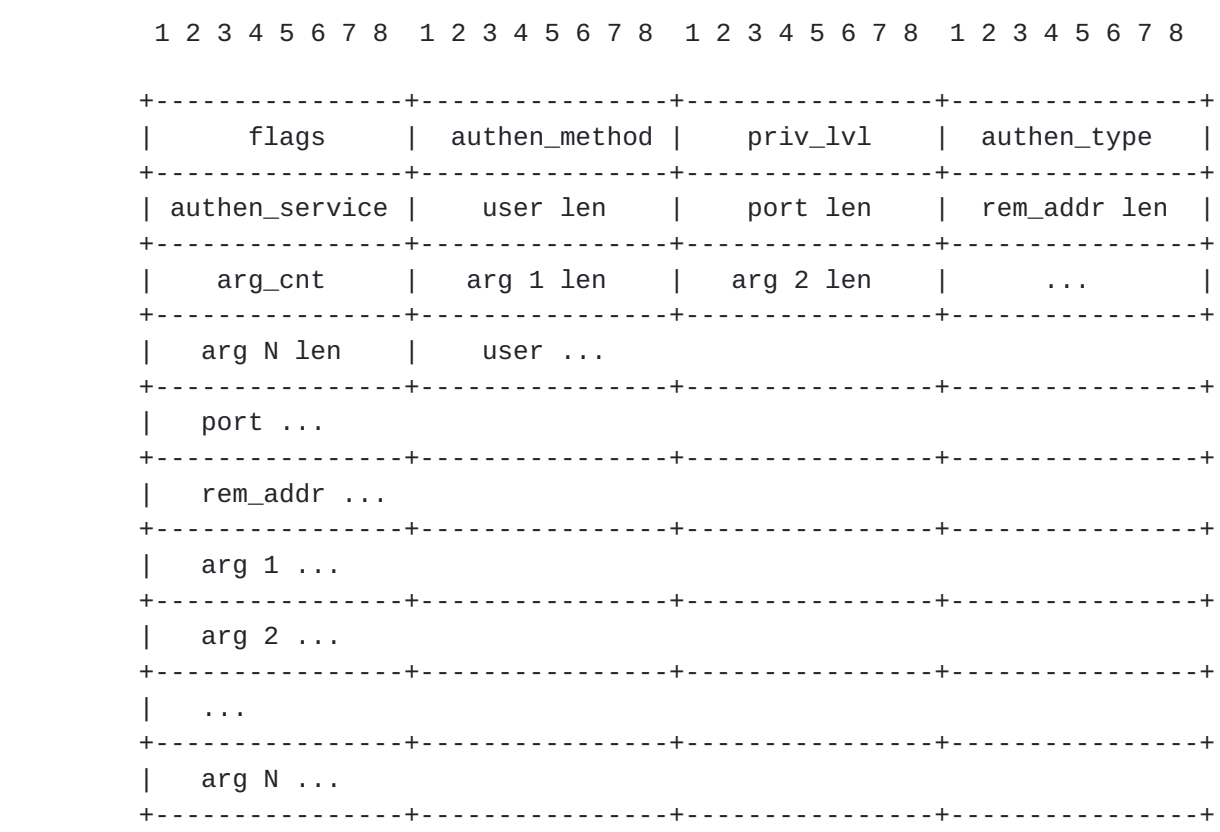
When the status equals TAC_PLUS_AUTHOR_STATUS_FOLLOW, then the arg_cnt MUST be 0. In that case, the actions to be taken and the contents of the data field are identical to the TAC_PLUS_AUTHEN_STATUS_FOLLOW status for Authentication.

None of the arg values have any relevance if an ERROR is set.

13. Accounting

TACACS+ accounting is very similar to authorization. The packet format is also 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.

13.1. The account REQUEST packet body



flags

This holds bitmapped flags.

TAC_PLUS_ACCT_FLAG_MORE := 0x01 (deprecated)

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.

Table 2: Accounting Attribute-value Pairs

task_id

Start and stop records for the same event MUST have matching (unique) task_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 keep real time.

timezone

The timezone abbreviation for all timestamps included in this packet.

event

Used only when "service=system". Current values are "net_acct", "cmd_acct", "conn_acct", "shell_acct" "sys_acct" and "clock_change". These indicate system level changes. The flags field SHOULD indicate whether the service started or stopped.

reason

Accompanies an event attribute. It describes why the event occurred.

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

paks

The number of packets transferred by this action.

paks_in

The number of input packets transferred by this action.

paks_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.

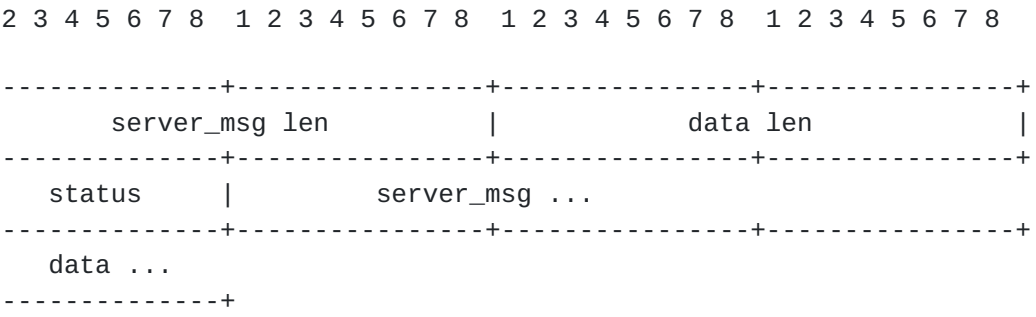
err_msg

An ascii string describing the status of the action.

NOTE: All numeric values in an attribute-value string are provided as decimal ASCII numbers.

13.2. The accounting REPLY packet body

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



status

This is the return status. Values are:
TAC_PLUS_ACCT_STATUS_SUCCESS := 0x01

TAC_PLUS_ACCT_STATUS_ERROR := 0x02

TAC_PLUS_ACCT_STATUS_FOLLOW := 0x21

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.

When the status equals TAC_PLUS_ACCT_STATUS_FOLLOW, then the actions to be taken and the contents of the data field are identical to the

TAC_PLUS_AUTHEN_STATUS_FOLLOW status for Authentication.

The daemon MUST terminate the session after sending a REPLY.

The TAC_PLUS_ACCT_FLAG_START flag indicates that this is a start accounting message. Start messages should only be sent once when a task is started. The TAC_PLUS_ACCT_FLAG_STOP indicates that this is a stop record and that the task 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 task is still running.

The START and STOP flags are mutually exclusive. When the WATCHDOG flag is set along with the START flag, it indicates that the update record is a duplicate of the original START record. If the START flag is not set, then this indicates a minimal record indicating only that task is still running. The STOP flag MUST NOT be set in conjunction with the WATCHDOG flag.

14. Compatibility between Minor Versions 0 and 1

Whenever a TACACS+ daemon receives a packet with a minor_version that it does not support, it should return an ERROR status with the minor_version set to the supported value closest to the requested value.

The changes between minor_version 0 and 1 all deal with the way that CHAP, ARAP and PAP authentications are handled.

In minor_version 0, CHAP, ARAP and outbound PAP authentications were performed by the NAS sending a SENDPASS packet to the daemon. The SENDPASS requested a copy of the user's plaintext password so that the NAS could complete the authentication. The CHAP hashing and ARAP encryption were all performed on the NAS. Inbound PAP performed a normal LOGIN, sending the username in the START packet and then waiting for a GETPASS and sending the password in a CONTINUE packet.

In minor_version 1, CHAP, ARAP and inbound PAP use LOGIN to perform inbound authentication and the exchanges use the data field so that the NAS only sends a single START packet and expects to receive a PASS or FAIL. SENDPASS has been deprecated and SENDAUTH introduced, so that the NAS can request authentication credentials for authenticating to a remote peer. SENDAUTH is only used for PPP when performing outbound authentication.

NOTE: Only those requests which have changed from their minor_version 0 implementation (i.e. ARAP, CHAP and PAP) should use the new

minor_version number of 1. All other requests (whose implementation has not changed) MUST continue to use the same minor_version number of 0 that they have always used.

If a daemon or NAS implementation desires to provide support for minor_number 0 TACACS+ hosts, it MUST pay attention to the minor_version in the TACACS+ header (as it should anyway) and be prepared to support the SENDPASS operation.

The removal of SENDPASS was prompted by security concerns, and implementors should think very carefully about how they wish to provide this service. On a NAS, the minor_version 0 compatibility can be layered such that higher layers only need to understand the minor_version 1 methodology, with the compatibility layer translating requests appropriately when contacting an older daemon.

On a TACACS+ server, when detecting minor_number 0, the daemon should allow for PAP authentications that do not send the password in the data field, but instead expect to read the PAP password from a subsequent CONTINUE packet.

If the daemon supports SENDPASS, then it should be prepared to handle such requests for CHAP and ARAP and even PAP, when outbound authentication takes place.

15. Notes to Implementors

For those interested in integrating one-time password support into TACACS+ daemons, there are some subtleties to consider. TACACS+ is designed to make this straightforward, but two cases require some extra care.

One-time password support with ARAP and PPP's CHAP authentication protocols is NOT straightforward, but there are work arounds. The problem lies in the nature of ARAP and CHAP authentication. Both employ 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 can rarely 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, and a fixed password can be used 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 `*'

(asterisk) character. After some deliberation, it was decided that it was the least likely character to be found in a username.

16. References

[1] D. Carrel, L. Grant, "The TACACS+ API Definition"

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

[3] R. Rivest, RFC 1321, "The MD5 Message-Digest Algorithm", April 1992.

[4] B. Lloyd, W. Simpson, RFC 1334, "PPP Authentication Protocols", October 1992.

[5] Apple Computer Corp. AppleTalk Remote Access Protocol (ARAP) Version 2.0 External Reference Specification. Preliminary document (no date available).

[6] D. Eastlake, S. Crocker, J. Schiller, RFC 1750, "Randomness Recommendations for Security", December 1994.

[7] S. Cobb, Microsoft PPP CHAP Extensions, Network working group Informational memo, Revision 1.2, March 1995.

17. Revision History

v1.75: first IETF submission

v1.76: fix encrypted flag text

v1.77: Add service 5 for Protocol translation

v1.78: Add ms chap description

Table of Contents