Operations   T. Dahm
Internet-Draft   A. Ota
Intended status: Informational   Google Inc
Expires: June 5, 2019   D. Medway Gash
                                                    Cisco Systems, Inc.
                                                             D. Carrel
                                                          vIPtela, Inc.
                                                              L. Grant
                                                     December 2, 2018

                          The TACACS+ Protocol
                       draft-ietf-opsawg-tacacs-12

Abstract

   TACACS+ provides Device Administration for routers, network access
   servers and other networked computing devices via one or more
   centralized servers. This document describes the protocol that is
   used by TACACS+.

Status of This Memo

Copyright Notice

Table of Contents

1.   Introduction

   Terminal Access Controller Access-Control System Plus (TACACS+) was
   conceived initially as a general Authentication, Authorization and
   Accounting protocol. It is primarily used today for Device
   Administration: authenticating access to network devices, providing
   central authorization of operations, and audit of those operations.

   A wide range of TACACS+ clients and servers are already deployed in
   the field. The TACACS+ protocol they are based on is defined in a
   draft document that was originally intended for IETF publication.
   This document is known as 'The Draft' [TheDraft] .

   It is intended that all implementations which conform to this
   document will conform to 'The Draft'. However, attention is drawn to
   the following specific adjustments of the protocol specification from
   'The Draft':

      This document officially removes SENDPASS for security reasons.

      The normative description of Legacy features such as ARAP and
      outbound authentication has been removed, however, the required
      enumerations are kept.

      The Support for forwarding to an alternative daemon
      (TAC_PLUS_AUTHEN_STATUS_FOLLOW) has been deprecated.

   The TACACS+ protocol separates the functions of Authentication,
   Authorization and Accounting. It allows for arbitrary length and
   content authentication exchanges, to support future authentication

mechanisms. 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 server to respond to each
component of that request.

The separation of authentication, authorization and accounting was a
key element of the design of TACACS+ protocol. Essentially it makes
TACACS+ a suite of three protocols. This document will address each
one in separate sections. Although TACACS+ defines all three, an
implementation or configuration is not required to employ all three.
Separating the elements is useful for Device Administration use case,
specifically, for authorization of individual commands in a session.
Note that there is no provision made at the protocol level for
association of an authentication to each authorization request.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in RFC2119 [RFC2119]

3. Technical Definitions

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

Client

The client is any device, (often a Network Access Server) that
provides access services. The clients usually provide a character
mode front end and then allow the user to telnet or rlogin to another
host.

Server

The server receives TACACS+ protocol requests, and replies according
to its business model, in accordance with the flows defined in this
document.

Packet

All uses of the word packet in this document refer to TACACS+
protocol packets unless explicitly noted otherwise.

4. TACACS+ Connections and Sessions

4.1. Connection

   TACACS+ uses TCP for its transport. Server port 49 is allocated for
   TACACS+ traffic.

4.2. 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.

   An accounting and authorization session will consist of a single pair
   of packets (the request and its reply). An authentication session
   may involve an arbitrary number of packets being exchanged. The
   session is an operational concept that is maintained between the
   TACACS+ client and server. It does not necessarily correspond to a
   given user or user action.

4.3. Single Connection Mode

   Single Connection Mode is intended to improve performance by allowing
   a client to multiplex multiple session on a single TCP connection.

   The packet header contains the TAC_PLUS_SINGLE_CONNECT_FLAG used by
   the client and server to negotiate the use of Single Connect Mode.

   The client sets this flag, to indicate that it supports multiplexing
   TACACS+ sessions over a single TCP connection. The client MUST NOT
   send a second packet on a connection until single-connect status has
   been established.

   To indicate it will support Single Connection Mode, the server sets
   this flag in the first reply packet in response to the first request
   from a client. The server may set this flag even if the client does
   not set it, but the client may ignore the flag and close the
   connection after the session completes.

   The flag is only relevant for the first two packets on a connection,
   to allow the client and server to establish Single Connection Mode.
   No provision is made for changing Single Connection Mode after the
   first two packets: the client and server MUST ignore the flag after
   the second packet on a connection.

   If single Connection Mode has not been established in the first two
   packets of a TCP connection, then both the client and the server
   close the connection at the end of the first session.

The client negotiates Single Connection Mode to improve efficiency.
The server may refuse to allow Single Connection Mode for the client.
For example, it may not be appropriate to allocate a long-lasting TCP
connection to a specific client in some deployments. Even if the
server is configured to permit single Connection Mode for a specific
client, the server may close the connection. For example: a server
may be configured to time out a Single Connection Mode TCP Connection
after a specific period of inactivity to preserve its resources. The
client MUST accommodate such closures on a TCP session even after
Single Connection Mode has been established.

4.4. Session Completion

The REPLY packets defined for the packets types in the sections below
(Authentication, Authorization and Accounting) contain a status
field. The complete set of options for this field depend upon the
packet type, but all three REPLY packet types define values
representing PASS, ERROR and FAIL, which indicate the last packet of
a regular session (one which is not aborted).

The server responds with a PASS or a FAIL to indicate that the
processing of the request completed and the client can apply the
result (PASS or FAIL) to control the execution of the action which
prompted the request to be sent to the server.

The server responds with an ERROR to indicate that the processing of
the request did not complete. The client can not apply the result
and it MUST behave as if the server could not be connected to. For
example, the client tries alternative methods, if they are available,
such as sending the request to a backup server, or using local
configuration to determine whether the action which prompted the
request should be executed.

Refer to the section (Section 5.4.3) on Aborting Authentication
Sessions for details on handling additional status options.

When the session is complete, then the TCP connection should be
handled as follows, according to whether Single Connection Mode was
negotiated:

If Single Connection Mode was not negotiated, then the connection
should be closed

If Single Connection Mode was enabled, then the connection SHOULD be
left open (see section (Section 4.3)), but may still be closed after
a timeout period to preserve deployment resources

If Single Connection Mode was enabled, but an ERROR occurred due to
connection issues (such as an incorrect secret, see section
(Section 4.7)), then any further new sessions MUST NOT be accepted
on the connection. If there are any sessions that have already been
established then they MAY be completed. Once all active sessions are
completed then the connection MUST be closed.

It is recommended that client implementations provide robust schemes
for dealing with servers which cannot be connected to. Options
include providing a list of servers for redundancy, and an option for
a local fallback configuration if no servers can be reached. Details
will be implementation specific.

The client should manage connections and handle the case of a server
which establishes a connection, but does not respond. The exact
behavior is implementation specific. It is recommended that the
client should close the connection after a configurable timeout.

4.5. Treatment of Enumerated Protocol Values

This document describes various enumerated values in the packet
header and the headers for specific packet types. For example in the
Authentication start packet type, this document defines the action
field with three values TAC_PLUS_AUTHEN_LOGIN, TAC_PLUS_AUTHEN_CHPASS
and TAC_PLUS_AUTHEN_SENDAUTH.

If the server does not implement one of the defined options in a
packet that it receives, or it encounters an option that is not
listed in this document for a header field, then it should respond
with a ERROR and terminate the session. This will allow the client
to try a different option.

If an error occurs 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 MUST be
returned to indicate an error.

4.6. Text Encoding

All text fields in TACACS+ MUST be printable US-ASCII, excepting
special consideration given to user field and data fields used for
passwords.

To ensure interoperability of current deployments, the TACACS+ client
and server MUST handle user fields and those data fields used for
passwords as 8-bit octet strings. The deployment operator MUST
ensure that consistent character encoding is applied from the end

client to the server. The encoding SHOULD be UTF-8, and other
encodings outside printable US-ASCII SHOULD be deprecated.

4.7. Data Obfuscation

The body of packets may be obfuscated. The following sections
describe the obfuscation method that is supported in the protocol.
In 'The Draft' this process was actually referred to as Encryption,
but the algorithm would not meet modern standards, and so will not be
termed as encryption in this document.

The obfuscation mechanism relies on a secret key, a shared secret
value that is known to both the client and the server. The secret
keys MUST remain secret.

Server implementations MUST allow a unique secret key to be
associated with every client. It is a site-dependent decision as to
whether the use of separate keys is appropriate.

The flag field may be set as follows:

TAC_PLUS_UNENCRYPTED_FLAG = 0x0

In this case, the packet body is obfuscated by XOR-ing it byte-wise
with a pseudo-random pad.

ENCRYPTED {data} = data ^ pseudo_pad

The packet body can then be de-obfuscated by XOR-ing it byte-wise
with a pseudo random pad.

data = ENCRYPTED {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 RFC 1321 [RFC1321]
.

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 server.

The version number and session_id are used as extracted from the header

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}

When a server detects that the secret(s) it has configured for the device mismatch, it MUST return ERROR. For details of TCP connection handling on ERROR, refer to section (Section 4.4)

TAC_PLUS_UNENCRYPTED_FLAG == 0x1

In this case, the entire packet body is in cleartext. Obfuscation and de-obfuscation are null operations. This method should be avoided unless absolutely required for debug purposes, when tooling does not permit de-obfuscation.

If deployment is configured for obfuscating a connection then the request MUST be dropped if TAC_PLUS_UNENCRYPTED_FLAG is set to true.

After a packet body is de-obfuscated, the lengths of the component values in the packet are summed. If the sum is not identical to the cleartext datalength value from the header, the packet MUST be discarded, and an ERROR signaled. For details of TCP connection handling on ERROR, refer to section (Section 4.4)

Commonly such failures are seen when the keys are mismatched between the client and the TACACS+ server.

4.8. The TACACS+ Packet Header

All TACACS+ packets begin with the following 12-byte header. The header 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    |     flags     |
    +---------------+---------------+---------------+---------------+
    |                                                               |
    |                          session_id                           |
    +---------------+---------------+---------------+---------------+
    |                                                               |
    |                            length                             |
    +---------------+---------------+---------------+---------------+
```
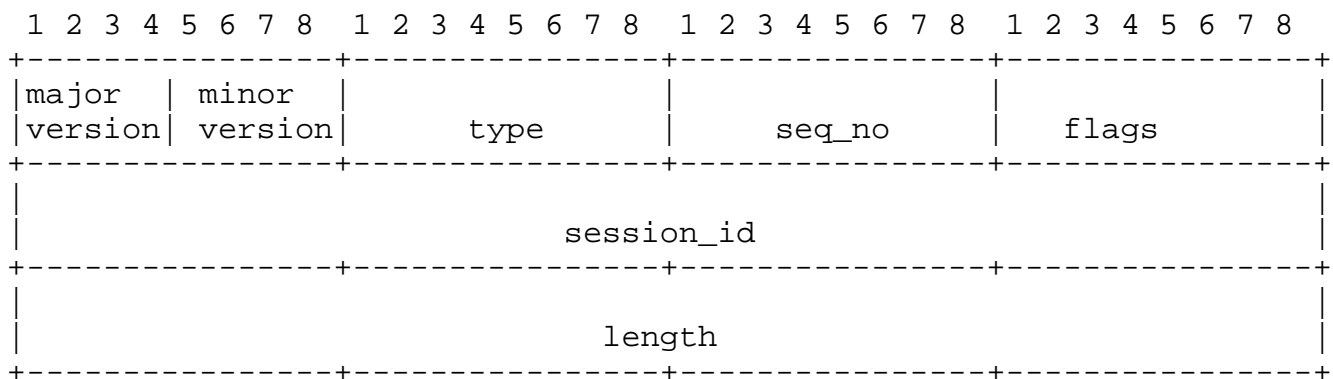
major_version

This is the major TACACS+ version number.

    TAC_PLUS_MAJOR_VER := 0xc

minor_version

The minor TACACS+ version number.

    TAC_PLUS_MINOR_VER_DEFAULT := 0x0

    TAC_PLUS_MINOR_VER_ONE := 0x1

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. The first 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+ servers
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 flag bit:

TAC_PLUS_UNENCRYPTED_FLAG := 0x01

This flag indicates that the sender did not obfuscate the body of the packet. The application of this flag will be covered in the security section (Section 10) .

This flag SHOULD be clear in all deployments. Modern network traffic tools support encrypted traffic when configured with the shared secret (see section below), so obfuscated mode can and SHOULD be used even during test.

The single-connection flag:

TAC_PLUS_SINGLE_CONNECT_FLAG := 0x04

This flag is used to allow a client and server to negotiate Single Connection Mode.

session_id

The Id for this TACACS+ session. This field does not change for the duration of the TACACS+ session. This number MUST be generated by a cryptographically strong random number generation method. Failure to do so will compromise security of the session. For more details refer to RFC 1750 [RFC1750]

length

The total length of the packet body (not including the header).

4.9. The TACACS+ Packet Body

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

   - To signal that any variable length data fields are unused, their length value is set to zero. Such fields MUST be ignored, and treated as if not present.

- the lengths of data and message fields in a packet are specified
by their corresponding length fields, (and are not null
terminated.)

- All length values are unsigned and in network byte order.

5. Authentication

Authentication is the action of determining who a user (or entity)
is. Authentication can take many forms. Traditional authentication
employs a name and a fixed password. However, fixed passwords are
vulnerable security, so many modern authentication mechanisms utilize
"one-time" passwords or a challenge-response query. TACACS+ is
designed to support all of these, and be flexible 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 himself or herself as someone who possesses the required
information (passwords, etc.) for those privileges.

5.1. 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  | authen_service|
+---------------+---------------+---------------+---------------+
|    user_len   |    port_len   |  rem_addr_len |    data_len   |
+---------------+---------------+---------------+---------------+
|    user ...
+---------------+---------------+---------------+---------------+
|    port ...
+---------------+---------------+---------------+---------------+
|    rem_addr ...
+---------------+---------------+---------------+---------------+
|    data...
+---------------+---------------+---------------+---------------+
```

Packet fields are as follows:

action

This indicates the authentication action. Legal values are listed
below.

        TAC_PLUS_AUTHEN_LOGIN := 0x01

        TAC_PLUS_AUTHEN_CHPASS := 0x02

        TAC_PLUS_AUTHEN_SENDAUTH := 0x04

    priv_lvl

    This indicates the privilege level that the user is authenticating
    as. Please refer to the Privilege Level section (Section 9) below.

    authen_type

    The type of authentication. 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 (deprecated)

        TAC_PLUS_AUTHEN_TYPE_MSCHAP := 0x05

        TAC_PLUS_AUTHEN_TYPE_MSCHAPV2 := 0x06

    authen_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 TAC_PLUS_AUTHEN_SVC_NONE option is intended for the authorization application of this field that indicates that no authentication was performed by the device.

The TAC_PLUS_AUTHEN_SVC_LOGIN option indicates regular login (as opposed to ENABLE) to a client device.

The TAC_PLUS_AUTHEN_SVC_ENABLE option identifies the ENABLE authen_service, which refers to a service requesting authentication in order to grant the user different privileges. This is comparable to the Unix "su(1)" command, which substitutes the current user's identity with another. An authen_service value of NONE is only to be used when none of the other authen_service values are appropriate. ENABLE may be requested independently, no requirements for previous authentications or authorizations are imposed by the protocol.

Other options are included for legacy/backwards compatibility.

user, user_len

The username is optional in this packet, depending upon the class of authentication. If it is absent, the client MUST set user_len to 0. If included, the user_len indicates the length of the user field, in bytes.

port, port_len

The printable US-ASCII name of the client port on which the authentication is taking place, and its length in bytes. 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). The port_len indicates the length of the port field, in bytes.

rem_addr, rem_addr_len

A printable US-ASCII string indicating the remote location from which the user has connected to the client. 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 is optional (since the information may not be available). The rem_addr_len indicates the length of the user field, in bytes.

data, data_len

This field is used to send data appropriate for the action and authen_type. It is described in more detail in the section Common Authentication flows (Section 5.4.2) . The data_len indicates the length of the data field, in bytes.

5.2. The Authentication REPLY Packet Body

The TACACS+ server sends only one type of authentication packet (a REPLY packet) to the client.

```
 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

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, server_msg_len

A message to be displayed to the user. This field is optional. The
printable US-ASCII charset MUST be used. The server_msg_len
indicates the length of the server_msg field, in bytes.

data, data_len

This field holds data that is a part of the authentication exchange
and is intended for the client, not the user. Examples of its use
are shown in the section Common Authentication flows (Section 5.4.2)
. The data_len indicates the length of the data field, in bytes.

5.3.   The Authentication CONTINUE Packet Body

This packet is sent from the client to the server following the
receipt of a REPLY 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
+----------------+----------------+----------------+----------------+
|          user_msg len           |            data_len             |
+----------------+----------------+----------------+----------------+
|     flags      |  user_msg ...
+----------------+----------------+----------------+----------------+
|     data ...
+----------------+
```
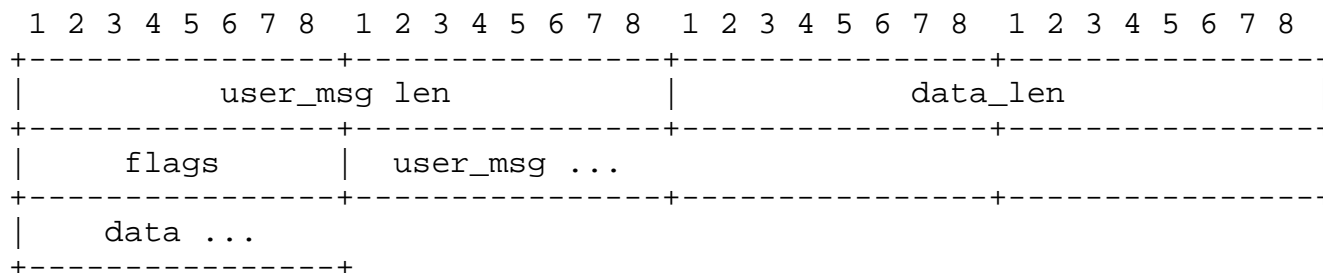
user_msg, user_msg_len

This field is the string that the user entered, or the client
provided on behalf of the user, in response to the server_msg from a
REPLY packet. The user_len indicates the length of the user field,
in bytes.

data, data_len

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. The data_len indicates the length of the data field, in
bytes.

flags

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

    TAC_PLUS_CONTINUE_FLAG_ABORT := 0x01

5.4. Description of Authentication Process

   The action, authen_type and authen_service fields (described above)
   combine to indicate what kind of authentication is to be performed.
   Every authentication START, REPLY and CONTINUE packet includes a data
   field. The use of this field is dependent upon the kind of the
   Authentication.

   This document defines a core set of authentication flows to be
   supported by TACACS+. Each authentication flow consists of a START
   packet. The server responds either with a request for more
   information (GETDATA, GETUSER or GETPASS) or a termination PASS,
   FAIL, ERROR or RESTART. The actions and meanings when the server
   sends a RESTART or ERROR 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 SHOULD provide
   server_msg content for 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.

   The client should interpret TAC_PLUS_AUTHEN_STATUS_GETUSER as a
   request for username and TAC_PLUS_AUTHEN_STATUS_GETPASS as a request
   for password. The TAC_PLUS_AUTHEN_STATUS_GETDATA is the generic
   request for more information to flexibly support future requirements.

   If the information being requested by the server form the client is
   sensitive, then the server should set the TAC_PLUS_REPLY_FLAG_NOECHO
   flag. When the client queries the user for the information, the
   response MUST NOT be echoed as it is entered.

   The data field is only used in the REPLY where explicitly defined
   below.

5.4.1. Version Behaviour

   The TACACS+ protocol is versioned to allow revisions while
   maintaining backwards compatibility. The version number is in every
   packet header. The changes between minor_version 0 and 1 apply only
   to the authentication process, and all deal with the way that CHAP
   and PAP authentications are handled. minor_version 1 may only be used
   for authentication kinds that explicitly call for it in the table
   below:

```
                   LOGIN      CHPASS     SENDAUTH
   ASCII           v0         v0         -
   PAP             v1         -          v1
   CHAP            v1         -          v1
   MS-CHAPv1/2     v1         -          v1
```

The '-' symbol represents that the option is not valid.

All authorisation and accounting and ASCII authentication use minor_version number of 0.

PAP, CHAP and MS-CHAP login use minor_version 1. The normal exchange is a single START packet from the client and a single REPLY from the server.

The removal of SENDPASS was prompted by security concerns, and is no longer considered part of the TACACS+ protocol.

5.4.2. Common Authentication Flows

This section describes common authentication flows. If the server does not implement an option, it MUST respond with TAC_PLUS_AUTHEN_STATUS_FAIL.

5.4.2.1. ASCII Login

```
     action = TAC_PLUS_AUTHEN_LOGIN
     authen_type = TAC_PLUS_AUTHEN_TYPE_ASCII
     minor_version = 0x0
```

This is a standard ASCII authentication. The START packet MAY contain the username. If the user does not include the username then the server MUST obtain it from the client with a CONTINUE TAC_PLUS_AUTHEN_STATUS_GETUSER. If the user does not provide a username then the server can send another TAC_PLUS_AUTHEN_STATUS_GETUSER request, but the server MUST limit the number of retries that are permitted, recommended limit is three attempts. When the server has the username, it will obtain the password using a continue with TAC_PLUS_AUTHEN_STATUS_GETPASS. ASCII login uses the user_msg field for both the username and password. The data fields in both the START and CONTINUE packets are not used for ASCII logins, any content MUST be ignored. The session is composed of a single START followed by zero or more pairs of REPLYs and CONTINUEs, followed by a final REPLY indicating PASS, FAIL or ERROR.

5.4.2.2. 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 RFC 1334 [RFC1334] . The REPLY
   from the server MUST be either a PASS, FAIL or ERROR.

5.4.2.3. 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 is 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 server calculates the PPP hash as
   defined in the PPP Authentication RFC RFC 1334 [RFC1334] and then
   compare that value with the response. The MD5 algorithm option is
   always used. The REPLY from the server MUST be a PASS, FAIL or
   ERROR.

   The selection of the challenge and its length are not an aspect of
   the TACACS+ protocol. However, it is strongly recommended that the
   client/endstation interaction is configured with a secure challenge.
   The TACACS+ server can help by rejecting authentications where the
   challenge is below a minimum length (Minimum recommended is 8 bytes).

5.4.2.4. MS-CHAP v1 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 server will use a combination of
MD4 and DES on the user's secret and the challenge, as defined in RFC
2433 [RFC2433] and then compare the resulting value with the
response. The REPLY from the server MUST be a PASS or FAIL.

For best practices, please refer to RFC 2433 [RFC2433] . The TACACS+
server MUST reject authentications where the challenge deviates from
8 bytes as defined in the RFC.

5.4.2.5. MS-CHAP v2 login

        action = TAC_PLUS_AUTHEN_LOGIN
        authen_type = TAC_PLUS_AUTHEN_TYPE_MSCHAPV2
        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 server will use the algorithm
specified RFC 2759 [RFC2759] on the user's secret and challenge and
then compare the resulting value with the response. The REPLY from
the server MUST be a PASS or FAIL.

For best practices for MS-CHAP v2, please refer to RFC2759 [RFC2759]
. The TACACS+ server MUST rejects authentications where the challenge
deviates from 16 bytes as defined in the RFC.

5.4.2.6. 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 user. The exchange MAY consist of multiple
   messages while the server collects the information it requires in
   order to allow changing the principal's privilege level. This
   exchange is very similar to an ASCII login (Section 5.4.2.1) .

   In order to readily distinguish enable requests from other types of
   request, the value of the authen_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.

5.4.2.7. ASCII change password request

```
        action = TAC_PLUS_AUTHEN_CHPASS
        authen_type = TAC_PLUS_AUTHEN_TYPE_ASCII
```

   This exchange consists of multiple messages while the server 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.

5.4.3. Aborting an Authentication 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. This information will
   be handled by the server according to the requirements of the
   deployment. The session is terminated, for more details about
   session termination, refer to section (Section 4.4)

   In cases of PASS, FAIL or ERROR, the server can insert a message into
   server_msg to be displayed to the user.

   The Draft 'The Draft' [TheDraft] defined a mechanism to direct
   authentication requests to an alternative server. This mechanism is
   regarded as insecure, is deprecated, and not covered here. The

client should treat TAC_PLUS_AUTHEN_STATUS_FOLLOW as
TAC_PLUS_AUTHEN_STATUS_FAIL

If the status equals TAC_PLUS_AUTHEN_STATUS_ERROR, then the host is
indicating that it is experiencing an unrecoverable error and the
authentication will 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 is restarted with a new START packet from the
client, with new session Id, and seq_no set to 1. This REPLY packet
indicates that the current authen_type value (as specified in the
START packet) is not acceptable for this session. The client may try
an alternative authen_type.

If a client does not implement TAC_PLUS_AUTHEN_STATUS_RESTART option,
then it MUST process the response as if the status was
TAC_PLUS_AUTHEN_STATUS_FAIL.

6. Authorization

In the TACACS+ Protocol, authorization is the action of determining
what a user is allowed to do. Generally authentication precedes
authorization, though it is not mandatory that a client use the same
service for authentication that it will use for authorization. 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 server to
determine, according to its configuration, if an unauthenticated user
is allowed the services in question.

Authorization does not merely provide yes or no answers, but it may
also customize the service for the particular user. A common use of
authorization is to provision a shell session when a user first logs
into a device to administer it. The TACACS+ server might respond to
the request by allowing the service, but placing a time restriction
on the login shell. For a list of common attributes used in
authorization, see the Authorization Attributes section (Section 8.2)
.

In the TACACS+ protocol an authorization is always a single pair of
messages: a REQUEST from the client followed by a REPLY from the
server.

The authorization REQUEST message contains a fixed set of fields that
indicate how the user was authenticated and a variable set of
arguments that describe the services and options for which
authorization is requested.

The REPLY contains a variable set of response arguments (attribute-
value pairs) that can restrict or modify the client's actions.

6.1. 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 by the client to
acquire the user information. As this information is not always
subject to verification, it is recommended that this field is
ignored.

    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 terminal line used to gain access.
  LOCAL is a client 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.

  priv_lvl

  This field is used in the same way as the priv_lvl field in
  authentication request and is described in the Privilege Level
  section (Section 9) below. It indicates the users current privilege
  level.

  authen_type

  This field coresponds to the authen_type field in the authentication
  section (Section 5) above. It indicates the type of authentication
  that was performed. If this information is not available, then the
  client will set authen_type to: TAC_PLUS_AUTHEN_TYPE_NOT_SET := 0x00.
  This value is valid only in authorization and accounting requests.

  authen_service

  This field is the same as the authen_service field in the
  authentication section (Section 5) above. It indicates the service
  through which the user authenticated.

  user, user_len

  This field contains the user's account name. The user_len MUST
  indicate the length of the user field, in bytes.

  port, port_len

  This field matches the port field in the authentication section
  (Section 5) above. The port_len indicates the length of the port
  field, in bytes.

rem_addr, rem_addr_len

This field matches the rem_addr field in the authentication section
(Section 5) above. The rem_addr_len indicates the length of the port
field, in bytes.

arg_cnt

The number of authorization arguments to follow

arg_1 ... arg_N, arg_1_len .... arg_N_len

The arguments are the primary elements of the authorization
interaction. In the request packet they describe the specifics of
the authorization that is being requested. Each argument is encoded
in the packet as a single arg filed (arg_1... arg_N) with a
corresponding length fields (which indicates the length of each
argument in bytes).

The authorization arguments in both the REQUEST and the REPLY are
attribute-value pairs. The attribute and the value are in a single
printable US-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.

It is not legal for an attribute name to contain either of the
separators. It is legal for attribute values to contain the
separators. This means that the arguments must be parsed until the
first separator is encountered, all characters in the argument, after
this separator, are interpreted as the argument value.

Optional arguments are ones that may be disregarded by either client
or server. Mandatory arguments require that the receiving side can
handle the attribute, that is: its implementation and configuration
includes the details of how to act on it. If the client receives a
mandatory argument that it cannot handle, it MUST consider the
authorization to have failed. It is legal to send an attribute-value
pair with a 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 minimum is two characters (one name-
value character and the separator)

Though the attributes allow extensibility, a common core set of
authorization attributes SHOULD be supported by clients and servers,
these are listed in the Authorization Attributes (Section 8.2)
section below.

6.2. The Authorization REPLY 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

      TAC_PLUS_AUTHOR_STATUS_FOLLOW := 0x21

   server_msg, server_msg_len

   This is a printable US-ASCII string that may be presented to the
   user. The server_msg_len indicates the length of the server_msg
   field, in bytes.

   data, data_len

   This is a printable US-ASCII string that may be presented on an
   administrative display, console or log. The decision to present this
   message is client specific. The data_len indicates the length of the
   data field, in bytes.

   arg_cnt

The number of authorization arguments to follow.

arg_1 ... arg_N, arg_1_len .... arg_N_len

The arguments describe the specifics of the authorization that is
being requested. For details of the content of the args, refer to:
Authorization Attributes (Section 8.2) section below. Each argument
is encoded in the packet as a single arg field (arg_1... arg_N) with
a corresponding length fields (which indicates the length of each
argument in bytes).

If the status equals TAC_PLUS_AUTHOR_STATUS_FAIL, then the requested
authorization MUST be denied.

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 MUST be applied according to the rules described
above.

If the status equals TAC_PLUS_AUTHOR_STATUS_PASS_REPL then the client
MUST use the authorization attribute-value pairs (if any) in the
response, instead of the authorization attribute-value pairs from the
request.

To approve the authorization with no modifications, the server sets
the status to TAC_PLUS_AUTHOR_STATUS_PASS_ADD and the arg_cnt to 0.

A status of TAC_PLUS_AUTHOR_STATUS_ERROR indicates an error occurred
on the server. For the differences between ERROR and FAIL, refer to
section Session Completion (Section 4.4) . None of the arg values
have any relevance if an ERROR is set, and must be ignored.

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.

7. Accounting

Accounting is typically the third action after authentication and
authorization. But again, neither authentication nor authorization
is 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 as an auditing tool for security services.
It may also be used to account for services used, such as in a
billing environment. 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
(Section 7) .

7.1. 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_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. They provide details for
the conditions on the client, and authentication context, so that
these details may be logged for accounting purposes.

See section 12 Accounting Attribute-value Pairs for the dictionary of
attributes relevant to accounting.

7.2. The Accounting REPLY Packet Body

The purpose of accounting is to record the action that has occurred
on the client. The server MUST reply with success only when the
accounting request has been recorded. If the server did not record
the accounting request then it MUST reply with ERROR.

```
 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

    TAC_PLUS_ACCT_STATUS_FOLLOW := 0x21

server_msg, server_msg_len

This is a printable US-ASCII string that may be presented to the
user. The server_msg_len indicates the length of the server_msg
field, in bytes.

data, data_len

This is a printable US-ASCII string that may be presented on an
administrative display, console or log. The decision to present this
message is client specific. The data_len indicates the length of the
data field, in bytes.

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.

TACACS+ accounting is intended to record various types of events on
clients, for example: login sessions, command entry, and others as

required by the client implementation. These events are collectively
referred to in 'The Draft' [TheDraft] as "tasks".

The TAC_PLUS_ACCT_FLAG_START flag indicates that this is a start
accounting message. Start messages will 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.

Summary of Accounting Packets

```
+----------+-------+-------+------------+------------------------+
| Watchdog | Stop  | Start | Flags & 0xE | Meaning                |
+----------+-------+-------+------------+------------------------+
|    0     |   0   |   0   |     0      | INVALID                |
|    0     |   0   |   1   |     2      | Start Accounting Record|
|    0     |   1   |   0   |     4      | Stop Accounting Record |
|    0     |   1   |   1   |     6      | INVALID                |
|    1     |   0   |   0   |     8      | Watchdog, no update    |
|    1     |   0   |   1   |     A      | Watchdog, with update  |
|    1     |   1   |   0   |     C      | INVALID                |
|    1     |   1   |   1   |     E      | INVALID                |
+----------+-------+-------+------------+------------------------+
```

The START and STOP flags are mutually exclusive.

The WATCHDOG flag is used by the client to communicate ongoing status
of a long-running task. Update records are sent at the client's
discretion. The frequency of the update depends upon the intended
application: A watchdog to provide progress indication will require
higher frequency than a daily keep-alive. When the WATCHDOG flag is
set along with the START flag, it indicates that the update record
provides additional or updated arguments from the original START
record. If the START flag is not set, then this indicates only that
task is still running, and no new information is provided (servers
MUST ignore any arguments). The STOP flag MUST NOT be set in
conjunction with the WATCHDOG flag.

The Server MUST respond with TAC_PLUS_ACCT_STATUS_ERROR if the client
requests an INVALID option.

8. Attribute-Value Pairs

TACACS+ is intended to be an extensible protocol. The attributes
used in Authorization and Accounting are not limited by this
document. Some attributes are defined below for common use cases,
clients MUST use these attributes when supporting the corresponding
use cases.

8.1. Value Encoding

All attribute values are encoded as printable US-ASCII strings. The following type representations SHOULD be followed

Numeric

All numeric values in an attribute-value string are provided as decimal printable US-ASCII numbers, unless otherwise stated.

Boolean

All boolean attributes are encoded as printable US-ASCII with values "true" or "false".

IP-Address

It is recommended that hosts be specified as a IP address so as to avoid any ambiguities. IPV4 address are specified as US-ASCII octet numerics separated by dots ('.'), IPV6 address text representation defined in RFC 4291 [RFC4291]

Date Time

Absolute date/times are specified in seconds since the epoch, 12:00am Jan 1 1970. The timezone MUST be UTC unless a timezone attribute is specified. Stardate is canonically inconsistent and so SHOULD NOT be used.

String

Many values have no specific type representation and so are interpreted as plain strings.

Empty Values

Attributes may be submitted with no value, in which case they consist of the name and the mandatory or optional separator. For example, the attribute "cmd" which has no value is transmitted as a string of four characters "cmd="

8.2. Authorization Attributes

service (String)

The primary service. Specifying a service attribute indicates that this is a request for authorization or accounting of that service.

For example: "shell", "tty-server", "connection", "system" and
"firewall". This attribute MUST always be included.

protocol (String)

the protocol field may be used to indicate a subset of a service.

cmd (String)

a shell (exec) command. This indicates the command name of the
command that is to be run. The "cmd" attribute MUST be specified if
service equals "shell".

Authorization of shell commands is a common use-case for the TACACS+
protocol. Command Authorization generally takes one of two forms:
session-based and command-based.

For session-based shell authorization, the "cmd" argument will have
an empty value. The client determines which commands are allowed in
a session according to the arguments present in the authorization.

In command-based authorization, the client requests that the server
determine whether a command is allowed by making an authorization
request for each command. The "cmd" argument will have the command
name as its value.

cmd-arg (String)

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 (Numeric)

printable US-ASCII number representing a connection access list.
Applicable only to session-based shell authorization.

inacl (String)

printable US-ASCII identifier for an interface input access list.

outacl (String)

printable US-ASCII identifier for an interface output access list.

addr (IP-Address)

a network address

addr-pool (String)

The identifier of an address pool from which the client can assign an address.

routing (Boolean)

Specifies whether routing information is to be propagated to, and accepted from this interface.

route (String)

Indicates an IPv4 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 is via the requesting peer.

timeout (Numeric)

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

idletime (Numeric)

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

autocmd (String)

an auto-command to run. Applicable only to session-based shell authorization.

noescape (Boolean)

Prevents user from using an escape character. Applicable only to session-based shell authorization.

nohangup (Boolean)

Boolean. Do not disconnect after an automatic command. Applicable only to session-based shell authorization.

priv-lvl (Numeric)

privilege level to be assigned. Please refer to the Privilege Level section (Section 9) below.

remote_user (String)

remote userid (authen_method must have the value
TAC_PLUS_AUTHEN_METH_RCMD). 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.

remote_host (String)

remote host (authen_method must have the value
TAC_PLUS_AUTHEN_METH_RCMD)

8.3. Accounting Attributes

The following attributes are defined for TACACS+ accounting only.
They MUST precede any attribute-value pairs that are defined in the
authorization section (Section 6) above.

task_id (String)

Start and stop records for the same event MUST have matching task_id
attribute values. The client MUST ensure that active task_ids are
not duplicated: a client MUST NOT reuse a task_id a start record
until it has sent a stop record for that task_id. Servers MUST not
make assumptions about the format of a task_id.

start_time (Date Time)

The time the action started (in seconds since the epoch.).

stop_time (Date Time)

The time the action stopped (in seconds since the epoch.)

elapsed_time (Numeric)

The elapsed time in seconds for the action.

timezone (String)

The timezone abbreviation for all timestamps included in this packet.
A database of timezones is maintained here: TZDB [TZDB]

event (String)

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 (String)

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

bytes (Numeric)

The number of bytes transferred by this action

bytes_in (Numeric)

The number of bytes transferred by this action from the endstation to
the client port

bytes_out (Numeric)

The number of bytes transferred by this action from the client to the
endstation port

paks (Numeric)

The number of packets transferred by this action.

paks_in (Numeric)

The number of input packets transferred by this action from the
endstation to the client port.

paks_out (Numeric)

The number of output packets transferred by this action from the
client port to the endstation.

err_msg (String)

A printable US-ASCII string describing the status of the action.

9. Privilege Levels

The TACACS+ Protocol supports flexible authorization schemes through
the extensible attributes.

One scheme is built into the protocol and has been extensively used
for Session-based shell authorization: Privilege Levels. Privilege
Levels are ordered values from 0 to 15 with each level being a
superset of the next lower value. Configuration and implementation

of the client will map actions (such as the permission to execute of
specific commands) to different privilege levels. 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

A Privilege level can be assigned to a shell (EXEC) session when it
starts (for example, TAC_PLUS_PRIV_LVL_USER). The client will permit
the actions associated with this level to be executed. This
privilege level is returned by the Server in a session-based shell
authorization (when "service" equals "shell" and "cmd" is empty).
When a user required to perform actions that are mapped to a higher
privilege level, then an ENABLE type reauthentication can be
initiated by the client. The client will insert the required
privilege level into the authentication header for enable
authentication request.

The use of Privilege levels to determine session-based access to
commands and resources is not mandatory for clients. Although the
privilege level scheme is widely supported, its lack of flexibility
in requiring a single monotonic hierarchy of permissions means that
other session-based command authorization schemes have evolved, and
so it is no longer mandatory for clients to use it. However, it is
still common enough that it SHOULD be supported by servers.

10. TACACS+ Security Considerations

The original TACACS+ Draft 'The Draft' [TheDraft] from 1998 did not
address all of the key security concerns which are considered when
designing modern standards. This section addresses known limitations
and concerns which will impact overall security of the protocol and
systems where this protocol is deployed to manage central
authentication, authorization or accounting for network device
administration.

Multiple implementations of the protocol described in the original
TACACS+ Draft 'The Draft' [TheDraft] have been deployed. As the
protocol was never standardized, current implementations may be
incompatible in non-obvious ways, giving rise to additional security
risks. This section does not claim to enumerate all possible
security vulnerabilities.

10.1. General Security of the Protocol

   TACACS+ protocol does not include a security mechanism that would
   meet modern-day requirements. Support for MD5-based crypto pad
   encryption fails to provide any kind of transport integrity, which
   presents at least the following risks:

      Accounting information may be modified by the man-in-the-middle
      attacker, making such logs unsuitable and untrustable for auditing
      purposes.

      Only the body of the request is obfuscated which leaves all header
      fields open to trivial modification by the man-in-the-middle
      attacker. For this reason, deployments SHOULD NOT use connections
      with TAC_PLUS_UNENCRYPTED_FLAG, as mentioned in the Best Practices
      section (Section 10.5) .

      Invalid or misleading values may be inserted by the man-in-the-
      middle attacker in various fields at known offsets to try and
      circumvent the authentication or authorization checks even inside
      the obfuscated body.

   While the protocol provides some measure of transport privacy, it is
   vulnerable to at least the following attacks:

      Brute force attacks exploiting increased efficiency of MD5 digest
      computation.

      Known plaintext attacks which may decrease the cost of brute force
      attack.

      Chosen plaintext attacks which may decrease the cost of a brute
      force attack.

      No forward secrecy.

   Even though, to the best knowledge of authors, this method of
   encryption wasn't rigorously tested, enough information is available
   that it is best referred to as "obfuscation" and not "encryption".

   For these reasons, users deploying TACACS+ protocol in their
   environments MUST limit access to known clients and MUST control the
   security of the entire transmission path. Attackers who can guess
   the key or otherwise break the obfuscation will gain unrestricted and
   undetected access to all TACACS+ traffic. Ensuring that a
   centralized AAA system like TACACS+ is deployed on a secured
   transport is essential to managing the security risk of such an
   attack.

The following parts of this section enumerate only the session-
specific risks which are in addition to general risk associated with
bare obfuscation and lack of integrity checking.

10.2. Security of Authentication Sessions

Authentication sessions SHOULD be used via a secure transport (see
Best Practices section (Section 10.5)) as the man-in-the-middle
attack may completely subvert them. Even CHAP, which may be
considered resistant to password interception, is unsafe as it does
not protect the username from a trivial man-in-the-middle attack.

This document deprecates the redirection mechanism using the
TAC_PLUS_AUTHEN_STATUS_FOLLOW option which was included in the
original draft. As part of this process, the secret key for a new
server was sent to the client. This public exchange of secret keys
means that once one session is broken, it may be possible to leverage
that key to attacking connections to other servers. This mechanism
SHOULD NOT be used in modern deployments. It MUST NOT be used
outside a secured deployment.

10.3. Security of Authorization Sessions

Authorization sessions SHOULD be used via a secure transport (see
Best Practices section (Section 10.5)) as it's trivial to execute a
successful man-in-the-middle attacks that changes well-known
plaintext in either requests or responses.

As an example, take the field "authen_method". It's not unusual in
actual deployments to authorize all commands received via the device
local serial port (a console port) as that one is usually considered
secure by virtue of the device located in a physically secure
location. If an administrator would configure the authorization
system to allow all commands entered by the user on a local console
to aid in troubleshooting, that would give all access to all commands
to any attacker that would be able to change the "authen_method" from
TAC_PLUS_AUTHEN_METH_TACACSPLUS to TAC_PLUS_AUTHEN_METH_LINE. In
this regard, the obfuscation provided by the protocol itself wouldn't
help much, because:

    Lack of integrity means that any byte in the payload may be
    changed without either side detecting the change.

    Known plaintext means that an attacker would know with certainty
    which octet is the target of the attack (in this case, 1st octet
    after the header).

In combination with known plaintext, the attacker can determine
with certainty the value of the crypto-pad octet used to obfuscate
the original octet.

10.4. Security of Accounting Sessions

Accounting sessions are not directly involved in authentication or
authorizing operations on the device. However, man-in-the-middle
attacker may do any of the following:

Replace accounting data with new valid or garbage which prevents
to provide distraction or hide information related to their
authentication and/or authorization attack attempts.

Try and poison accounting log with entries designed to make
systems behave in unintended ways (which includes TACACS+ server
and any other systems that would manage accounting entries).

In addition to these direct manipulations, different client
implementations pass different fidelity of accounting data. Some
vendors have been observed in the wild that pass sensitive data like
passwords, encryption keys and similar as part of the accounting log.
Due to lack of strong encryption with perfect forward secrecy, this
data may be revealed in future, leading to a security incident.

10.5. TACACS+ Best Practices

With respect to the observations about the security issues described
above, a network administrator MUST NOT rely on the obfuscation of
the TACACS+ protocol and TACACS+ MUST be deployed over networks which
ensure privacy and integrity of the communication. TACACS+ MUST be
used within a secure deployment. Failure to do so will impact
overall network security.

TACACS+ SHOULD be deployed over a network which is separated from
other traffic.

The following recommendations impose restrictions on how the protocol
is applied. These restrictions were not imposed in the original
draft. New implementations, and upgrades of current implementations,
MUST implement these recommendations.

10.5.1. Shared Secrets

TACACS+ servers and clients MUST treat shared secrets as sensitive
data to be managed securely, as would be expected for other sensitive
data such as identity credential information. TACACS+ servers MUST

not leak sensitive data. For example, TACACS+ servers should not
expose shared secrets in logs.

TACACS+ servers MUST allow a dedicated secret key to be defined for
each client.

TACACS+ servers SHOULD warn administrators if secret keys are not
unique per client.

TACACS+ server administrators SHOULD always define a secret for each
client.

TACACS+ servers and clients MUST support shared keys that are at
least 32 characters long.

TACACS+ servers MUST support policy to define minimum complexity for
shared keys.

TACACS+ clients SHOULD NOT allow servers to be configured without
shared secret key, or shared key that is less than 16 characters
long.

TACACS+ server administrators SHOULD configure secret keys of minimum
16 characters length.

TACACS+ server administrators SHOULD change secret keys at regular
intervals.

10.5.2. Connections and Obfuscation

TACACS+ servers MUST allow the definition of individual clients. The
servers MUST only accept network connection attempts from these
defined, known clients.

TACACS+ servers MUST reject connections with
TAC_PLUS_UNENCRYPTED_FLAG set, when there is a shared secret set on
the server for the client requesting the connection.

If an invalid shared secret is detected when processing packets for a
client, TACACS+ servers MUST NOT accept any new sessions on that
connection. TACACS+ servers MUST terminate the connection on
completion of any sessions that were previously established with a
valid shared secret on that connection.

TACACS+ clients MUST NOT set TAC_PLUS_UNENCRYPTED_FLAG when a secret
is defined. Clients MUST be implemented in a way that requires
explicit configuration to enable the use of
TAC_PLUS_UNENCRYPTED_FLAG.

When a TACACS+ client receives responses from servers where:

   the response packet was received from the server configured with
   shared key, but the packet jas TAC_PLUS_UNENCRYPTED_FLAG set.

   the response packet was received from the server configured not to
   use obfuscation, but the packet has TAC_PLUS_UNENCRYPTED_FLAG not
   set.

then the TACACS+ client MUST close TCP session, and process the
response in the same way that a TAC_PLUS_AUTHEN_STATUS_FAIL
(authentication sessions) or TAC_PLUS_AUTHOR_STATUS_FAIL
(authorization sessions) was received.

10.5.3. Authentication

To help TACACS+ administraots select the stronger authentication
options, TACACS+ servers MUST allow the administrator to configure
the server to only accept challenge/response options for
authentication (TAC_PLUS_AUTHEN_TYPE_CHAP or
TAC_PLUS_AUTHEN_TYPE_MSCHAP or TAC_PLUS_AUTHEN_TYPE_MSCHAPV2 for
authen_type).

TACACS+ server administrators SHOULD enable the option mentioned in
the previous paragraph. TACACS+ Server deployments SHOULD ONLY
enable other options (such as TAC_PLUS_AUTHEN_TYPE_ASCII or
TAC_PLUS_AUTHEN_TYPE_PAP) when unavoidable due to requirements of
identity/password systems.

TACACS+ server administrators SHOULD NOT allow the same credentials
to be applied in challenge-based (TAC_PLUS_AUTHEN_TYPE_CHAP or
TAC_PLUS_AUTHEN_TYPE_MSCHAP or TAC_PLUS_AUTHEN_TYPE_MSCHAPV2) and non
challenge-based authen_type options as the insecurity of the latter
will compromise the security of the former.

TAC_PLUS_AUTHEN_SENDAUTH and TAC_PLUS_AUTHEN_SENDPASS options
mentioned in the original draft SHOULD not be used, due to their
security implications. TACACS+ servers SHOULD NOT implement them.
If they must be implemented, the servers MUST default to the options
being disabled and MUST warn the administrator that these options are
not secure.

10.5.4. Authorization

The authorization and accounting features are intended to provide
extensibility and flexibility. There is a base dictionary defined in
this document, but it may be extended in deployments by using new
attribute names. The cost of the flexibility is that administrators

and implementors MUST ensure that the attribute and value pairs
shared between the clients and servers have consistent
interpretation.

TACACS+ clients that receive an unrecognised mandatory attribute MUST
evaluate server response as if they received
TAC_PLUS_AUTHOR_STATUS_FAIL.

10.5.5. Redirection Mechanism

The original draft described a redirection mechanism
(TAC_PLUS_AUTHEN_STATUS_FOLLOW). This feature is difficult to
secure. The option to send secret keys in the server list is
particularly insecure, as it can reveal client shared secrets.

TACACS+ servers SHOULD deprecate the redirection mechanism.

If the redirection mechanism is implemented then TACACS+ servers MUST
disable it by default, and MUST warn TACACS+ server administrators
that it must only be enabled within a secure deployment due to the
risks of revealing shared secrets.

TACACS+ clients SHOULD deprecate this feature by treating
TAC_PLUS_AUTHEN_STATUS_FOLLOW as TAC_PLUS_AUTHEN_STATUS_FAIL.

11. Acknowledgements

The authors would like to thank the following reviewers whose
comments and contributions made considerable improvements to the
document: Alan DeKok, Alexander Clouter, Chris Janicki, Tom Petch,
Robert Drake, among many others.

The authors would particularly like to thank Alan DeKok, who provided
significant insights and recommendations on all aspects of the
document and the protocol. Alan DeKok has dedicated considerable
time and effort to help improve the document, identifying weaknesses
and providing remediation.

The authors would also like to thanks the support from the OPSAWG
Chairs and advisors.

12. References

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321,
           April 1992.

   [RFC1334] Lloyd, B. and W. Simpson, "PPP Authentication Protocols",
              RFC 1334, DOI 10.17487/RFC1334, October 1992,
              <http://www.rfc-editor.org/info/rfc1334>.

   [RFC1750] Eastlake 3rd, D., Crocker, S., and J. Schiller,
              "Randomness Recommendations for Security", RFC 1750,
              DOI 10.17487/RFC1750, December 1994,
              <http://www.rfc-editor.org/info/rfc1750>.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <https://www.rfc-editor.org/info/rfc2119>.

   [RFC2433] Zorn, G. and S. Cobb, "Microsoft PPP CHAP Extensions",
              RFC 2433, DOI 10.17487/RFC2433, October 1998,
              <http://www.rfc-editor.org/info/rfc2433>.

   [RFC2759] Zorn, G., "Microsoft PPP CHAP Extensions, Version 2",
              RFC 2759, DOI 10.17487/RFC2759, January 2000,
              <http://www.rfc-editor.org/info/rfc2759>.

   [RFC4291] Hinden, R. and S. Deering, "IP Version 6 Addressing
              Architecture", RFC 4291, DOI 10.17487/RFC4291, February
              2006, <https://www.rfc-editor.org/info/rfc4291>.

   [TheDraft]
              Carrel, D. and L. Grant, "The TACACS+ Protocol Version
              1.78", June 1997,
              <https://tools.ietf.org/html/draft-grant-tacacs-02>.

   [TZDB]   Eggert, P. and A. Olson, "Sources for Time Zone and
              Daylight Saving Time Data", 1987,
              <https://www.iana.org/time-zones>.

Authors' Addresses

   Thorsten Dahm
   Google Inc
   1600 Amphitheatre Parkway
   Mountain View, CA 94043
   US

   EMail: thorstendlux@google.com

Andrej Ota
Google Inc
1600 Amphitheatre Parkway
Mountain View, CA   94043
US

EMail: andrej@ota.si

Douglas C. Medway Gash
Cisco Systems, Inc.
170 West Tasman Dr.
San Jose, CA   95134
US

EMail: dcmgash@cisco.com

David Carrel
vIPtela, Inc.
1732 North First St.
San Jose, CA   95112
US

EMail: dcarrel@viptela.com

Lol Grant

EMail: lol.grant@gmail.com