Operations
Internet-Draft
Intended status: Standards Track
Expires: January 4, 2016

T. Dahm
A. Ota
Google Inc
D. Medway Gash
Cisco Systems, Inc.
D. Carrel

L. Grant
July 3, 2015

The TACACS+ Protocol
draft-dahm-opsawg-tacacs-01.txt

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+.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 4, 2016.

Copyright Notice

Table of Contents

1. Introduction

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 10, 2015.

A wide range of TACACS+ clients and servers are already deployed in the field, based upon ''The Draft''. This specification is essentially a refactoring of the draft with some minor additions. The definitions in the draft should map onto this document, such that any implementations based on the draft will be compliant with this document. Chief changes between the documents:

- This document introduces a TLS encryption scheme

- This document supports MS-CHAPv2

- This document officially removes SENDPASS for security reasons. This option was still documented as 'deprecated' in ''The Draft''

- This document deprecates description of legacy features such as ARAP and outbound authentication. The required enumerations are kept, but related normative description is removed.

The TACACS+ protocol is the latest generation of TACACS. It separates the functions of Authentication, Authorization and Accounting. It allows for arbitrary length and content authentication exchanges, which will support 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 server to respond to each component of that request.

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

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 himself or herself 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+ server 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 (Section 5) .

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 6) .

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. A client may also support protocol based access services.

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.

3. TACACS+ Connections and Sessions

3.1. Connection

TACACS+ uses TCP for its transport. The server 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.

The encryption described inside the protocol below is kept to provide backwards compatibility, however it does not provide sufficiently robust security. For this reason, the connection can be upgraded to a secured tunnel using START TLS.

3.2. 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.

3.3. Single Connect Mode

The packet header (see below) contains a flag to allow sessions to be multiplexed on a connection.

If a client sets this flag, this indicates 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.

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

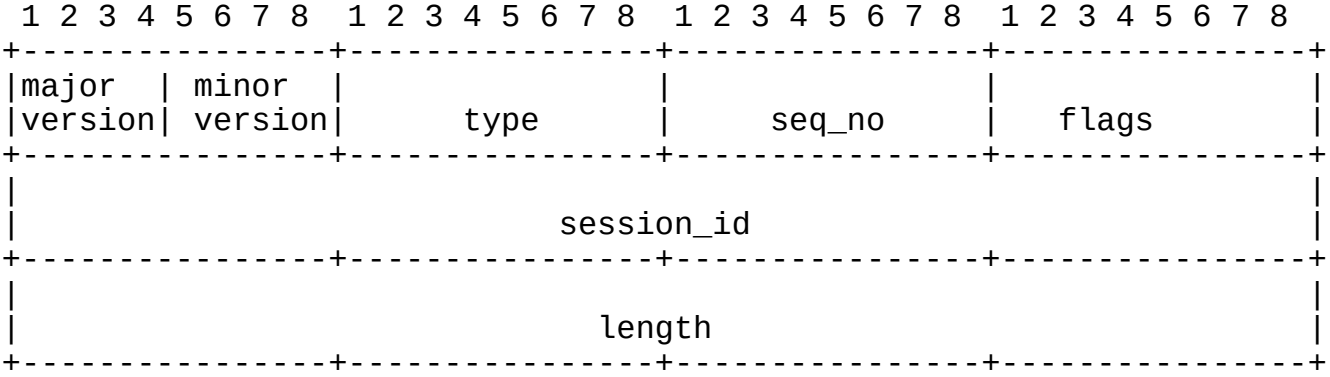
The flag is only relevant for the first two packets on a connection, to allow the client and server to establish single connection mode.

The flag MUST be ignored after these two packets since the single-connect status of a connection, once established, must not be changed. The connection must instead be closed and a new connection opened, if required.

When single-connect status is established, multiple sessions MUST be allowed simultaneously and/or consecutively on a single TCP connection. If single-connect status has not been established in the first two packets of a TCP connection, then the connection must be closed at the end of the first session.

3.4. 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.

TAC_PLUS_MINOR_VER_DEFAULT := 0x0

TAC_PLUS_MINOR_VER_ONE := 0x1

type

This is the packet type. Legal values are:

TAC_PLUS_START_TLS := 0x00 (Upgrade Connection to TLS)

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 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 unencrypted flag bit says whether encryption is being used on the body of the 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

This flag is used to allow a client and server to agree whether multiple sessions may be multiplexed onto a single connection.

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. RFC 1750 [RFC1750])

length

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

3.5. 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:

- 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 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.

3.6. Encryption

3.6.1. Legacy Body Encryption

The body of packets may be encrypted. The following sections describe the legacy encryption mechanism that is supported to enable backwards compatibility with "The Draft".

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 server. This document does not discuss the management and storage of those keys. It is an implementation detail of the server and client, as to whether they will maintain only one key, or a different key for each client or server 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 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 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. If the unencrypted flag is not set, and the packet is not encrypted, it must be dropped.

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 client 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 MUST be returned to indicate an error.

3.6.2. Start TLS Transport Encryption

TACACS+ supports a mechanism to upgrade the plaintext connection to TLS. This is performed using the packet type TAC_PLUS_START_TLS. Packet Type 0x00

Flag field MUST have TAC_PLUS_SINGLE_CONNECT_FLAG set for packet type 0x00 for client request and server response.

Flag field MUST have TAC_PLUS_UNENCRYPTED_FLAG set for packet type 0x00 for client request and server response.

Only client CAN send packet type 0x00 and it MUST send it as a first packet in session.

After negotiating TLS session, client MUST NOT send any more packets of type 0x00 while TLS session is established. If client needs to renegotiate secure transport, it MUST close the current connection and open a new connection.

After negotiating TLS session client SHOULD set TAC_PLUS_UNENCRYPTED_FLAG as there would be no benefit for additional encryption of body content.

3.6.3. TLS Handshake

Client sends complete ClientHello packet in the first packet of the session as a packet body. Packet type is TAC_PLUS_START_TLS.

If server supports STARTTLS, it SHALL respond with ServerHello. There is no further TACACS+ header encapsulation needed for TLS handshake to proceed.

If server does not support STARTTLS, it MUST respond with the same header as received, but with length set to 0 and seq_no incremented by one.

If TLS handshake succeeds, client CAN use the tunnel as a method of secure transport.

If TLS handshake fails, client MUST close the connection.

Server CAN be configured to only allow STARTTLS protected sessions. In this case, it MUST reject all client requests which are not of type TAC_PLUS_STARTTLS and are not received over already established

connection. It rejects them by sending back the same header as received, but with length set to 0 and seq_no incremented by one.

3.6.4. TLS Cypher Requirements

TLS Protocol Version

TACACS+ STARTTLS MUST implement at least TLS version 1.2. TACACS+ STARTTLS MAY implement higher TLS versions.

Mandatory Cipher Suites

TLS 1.2 RFC 5246 [RFC5246] allows specifying application profiles prescribing which cipher suites to implement for interoperability purposes. To maintain simplicity of current TACACS+ configuration using preshared secrets, the server implementation MUST implement:

TLS_DHE_PSK_WITH_3DES_EDE_CBC_SHA

TLS_DHE_PSK_WITH_AES_128_CBC_SHA

TLS_DHE_PSK_WITH_AES_256_CBC_SHA

Client MUST implement at least one of cipher suites which are implemented on the server, and it MAY implement all of them.

Both clients and servers MAY implement other cipher suites, but their interoperability is not guaranteed and their implementation is outside of scope of this document.

PSK Identity Requirements.

Because determining a correct PSK value on the server side is a computationally intensive operation requiring multiple round trips, a mechanism for hitless key change must be defined. During TLS handshake, a client MUST use PSK identity as defined in RFC 4279 [RFC4279] to signal to server which PSK value to use. If server does not recognize PSK identity it MUST respond with decrypt_error alert and MUST NOT respond with unknown_psk_identity. Process to change preshared keys on server and client is then:

1. Add new key with new PSK identity on the server.

2. Add new key with new PSK identity on the client.

3. Remove old key with old PSK identity from the client.

4. Remove old key with old PSK identity from the server.

Note: PSK identity is transmitted in clear text and must not contain information which could aid an attacker who can eavesdrop on the connection.

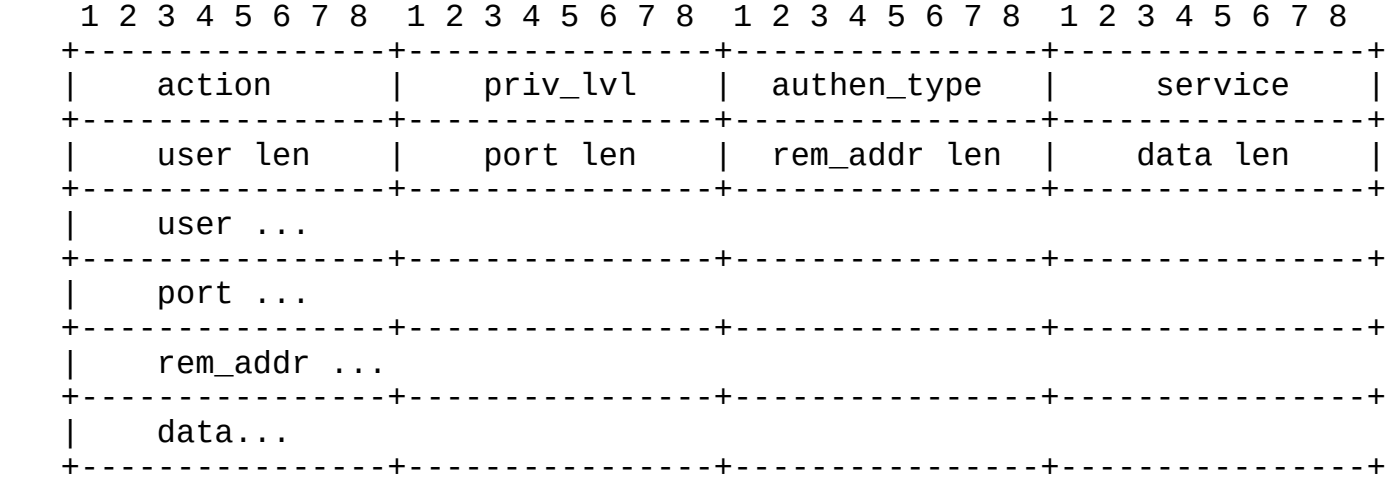
3.6.5. Security Considerations

Transport encryption SHOULD be used in deployments when both the clients and servers support it. Servers that support Transport encryption MAY be configured to allow Legacy Body Encryption when Transport encryption is not supported by the client.

It is NOT recommended to deploy TACACS+ without Transport or Legacy Body encryption, other than for test environments.

4. Authentication

4.1. 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_SENDAUTH := 0x04

priv_lvl

This indicates the privilege level that the user is authenticating as. Please refer to the Privilege Level section (Section 8) below.

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 (deprecated)

TAC_PLUS_AUTHEN_TYPE_MSCHAP := 0x05

TAC_PLUS_AUTHEN_TYPE_MSCHAPV2 := 0x06

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 encoded in [UTF-8]. It is optional in this packet, depending upon the class of authentication.

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

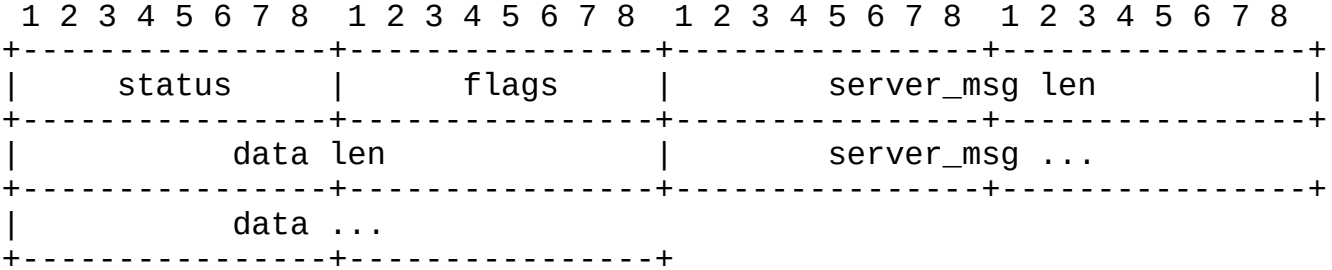
An ASCII string this is a "best effort" description of the remote location from which the user has connected to the client. It is intended to hold a network address if the user is connected via a network, a caller ID is the user is connected via ISDN or a POTS, or any other remote location information that is available. This field is optional (since the information may not be available).

data

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

4.2. The Authentication REPLY Packet Body

The TACACS+ server 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

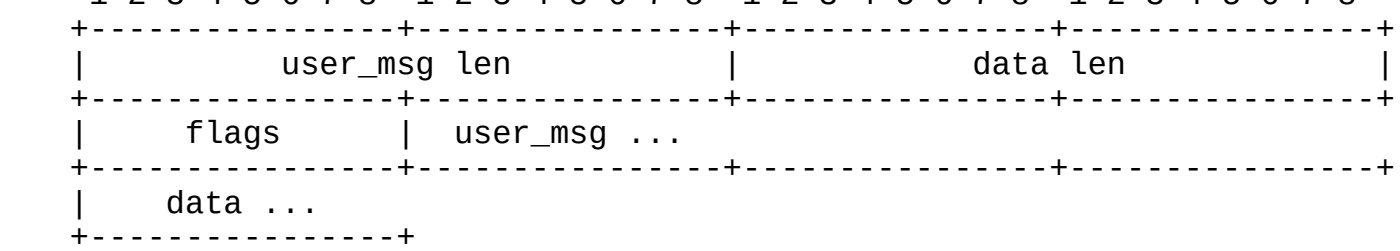
A message to be displayed to the user. This field is optional. If it exists, it is intended to be presented to the user. US-ASCII charset must be used.

data

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

4.3. The Authentication CONTINUE Packet Body

This packet is sent from the client to the server following the receipt of a REPLY packet.



user_msg

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.

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
```

4.4. Description of Authentication Process

Authentications are classified by the action, authen_type and service fields in the START packet of the authentication Session. The user, priv_lvl, service, port and rem_addr in the START packet are all provided to help identify the conditions on the client.

The information necessary to transact the authentication is passed in the data field of every START, REPLY and CONTINUE packet. The usage of this field varies according to the classification of the authentication, and is described below. For all REPLY packets, the server_msg may contain a message to be displayed to the user.

A set of standard authentication classifications is defined in this document. 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 or FAIL). The actions and meanings when the server 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.

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. TAC_PLUS_AUTHEN_STATUS_GETDATA is the generic request for more information. If the TAC_PLUS_REPLY_FLAG_NOECHO flag is set in the REPLY, then the user response must not be echoed as it is entered. The data field is only used in the REPLY where explicitly defined below.

4.4.1. 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 classes that explicitly call for it in the table below:

	LOGIN	CHPASS	SENDAUTH
ASCII	v0	v0	NA
PAP	v1	NA	v1
CHAP	v1	NA	v1
MS-CHAP	v1	NA	v1

When a server 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.

In minor_version 0, CHAP and outbound PAP authentications were performed by the client sending a SENDPASS packet to the server. The SENDPASS requested a copy of the user’s plaintext password so that the client could complete the authentication. The CHAP hashing was performed on the client. 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 and inbound PAP use LOGIN to perform inbound authentication and the exchanges use the data field so that the client only sends a single START packet and expects to receive a PASS or FAIL. SENDPASS has been deprecated and SENDAUTH introduced, so that the client 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. CHAP, MS-CHAP and PAP authentications) should use the new minor_version number of 1. All other requests (i.e. all authorisation and accounting and ASCII authentication) MUST continue to use the same minor_version number of 0. The removal of SENDPASS

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

4.4.2. Common Authentication Flows

This section describes the authentication flows that should be supported.

Inbound 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, 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).

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

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 server will run MD5 over the id, the user's secret and the challenge, as defined in the PPP Authentication RFC RFC 1334 [RFC1334] and then compare that value with the response. The REPLY from the server MUST be a PASS or FAIL.

Inbound 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.

Inbound MS-CHAP v2 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 the algorithm specified RFC RFC2759 [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.

Outbound PAP request (Backward compatibility, not for new designs)

```
action = TAC_PLUS_AUTHEN_SENDAUTH
authen_type = TAC_PLUS_AUTHEN_TYPE_PAP
minor_version = 0x1
```

This is used when the client 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 client, 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 client. Use of this is discouraged, but supported for complete interoperability with the PPP protocol.

Outbound CHAP request (Backward compatibility, not for new designs)

```
action = TAC_PLUS_AUTHEN_SENDAUTH
authen_type = TAC_PLUS_AUTHEN_TYPE_CHAP
minor_version = 0x1
```

This is used when the client 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 server will run MD5 over the id, the user’s secret and the challenge, as defined in the PPP Authentication RFC RFC 1334 [RFC1334] .

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

Outbound MS-CHAP request (Backward compatibility, not for new designs)

```
action = TAC_PLUS_AUTHEN_SENDAUTH
authen_type = TAC_PLUS_AUTHEN_TYPE_MSCHAP
minor_version = 0x1
```

This is used when the client 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 server will use MD4 and DES to process the user's secret and the challenge, as defined in RFC 2433 [RFC2433] .

The REPLY from the server 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.

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 server 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.

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.

4.4.3. 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. 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+ server requests that authentication should be performed with an alternate server. The data field MUST contain ASCII text describing one or more servers. A server 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 Carriage Return (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.

5. 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.

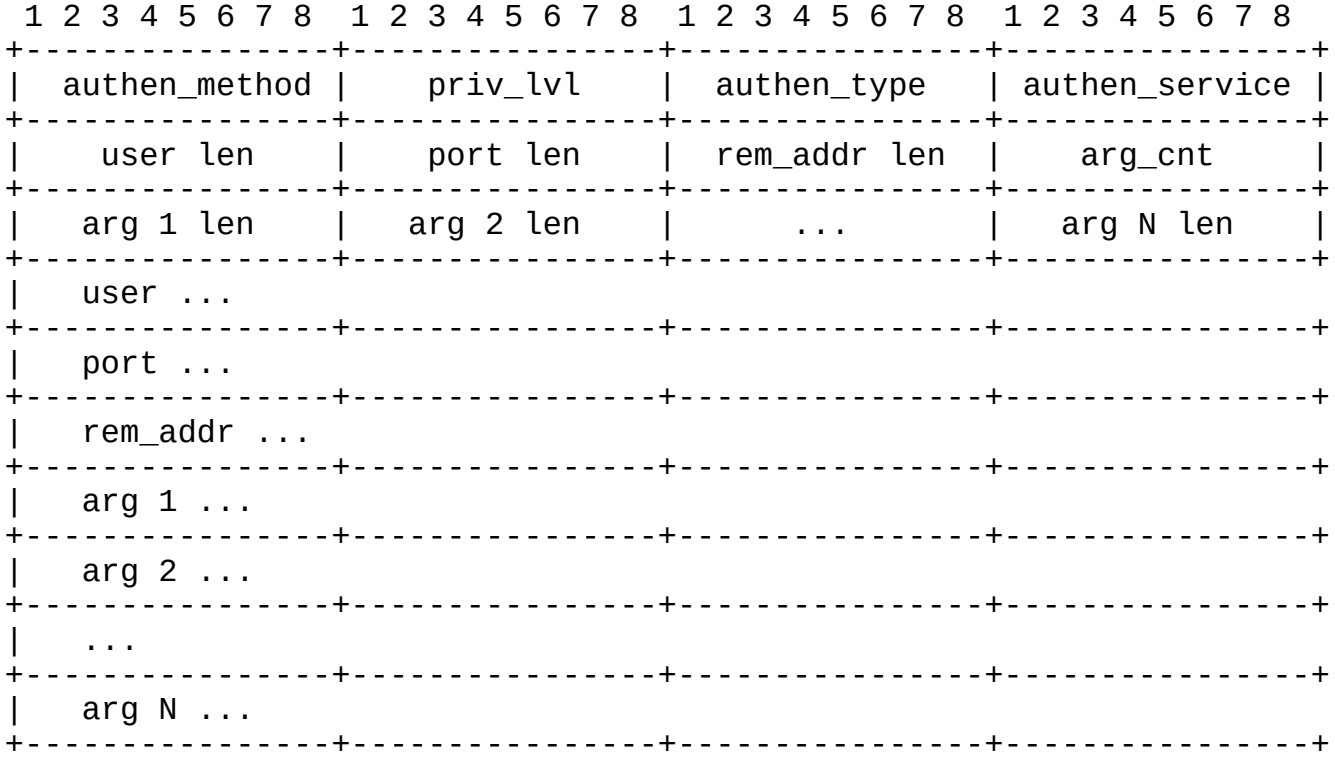
The authorization REQUEST message contains a fixed set of fields that describe the authenticity of the user or process, and a variable set of arguments that describe the services and options for which authorization is requested.

The RESPONSE contains a variable set of response arguments (attribute-value pairs) that 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.

5.1. 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 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. (One should be aware of the security limitations to R-command authentication.)

priv_lvl

This field matches the priv_lvl field in authentication request and is described in the Privilege Level section (Section 8) below. It indicates the users current privilege level.

authen_type

This field matches the authen_type field in the authentication section (Section 4) above. It indicates the type of authentication that was performed.

authen_service

This field matches the service field in the authentication section (Section 4) 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 (Section 4) above.

rem_addr

This field matches the rem_addr field in the authentication section (Section 4) above.

arg_cnt

The number of authorization arguments to follow

arg

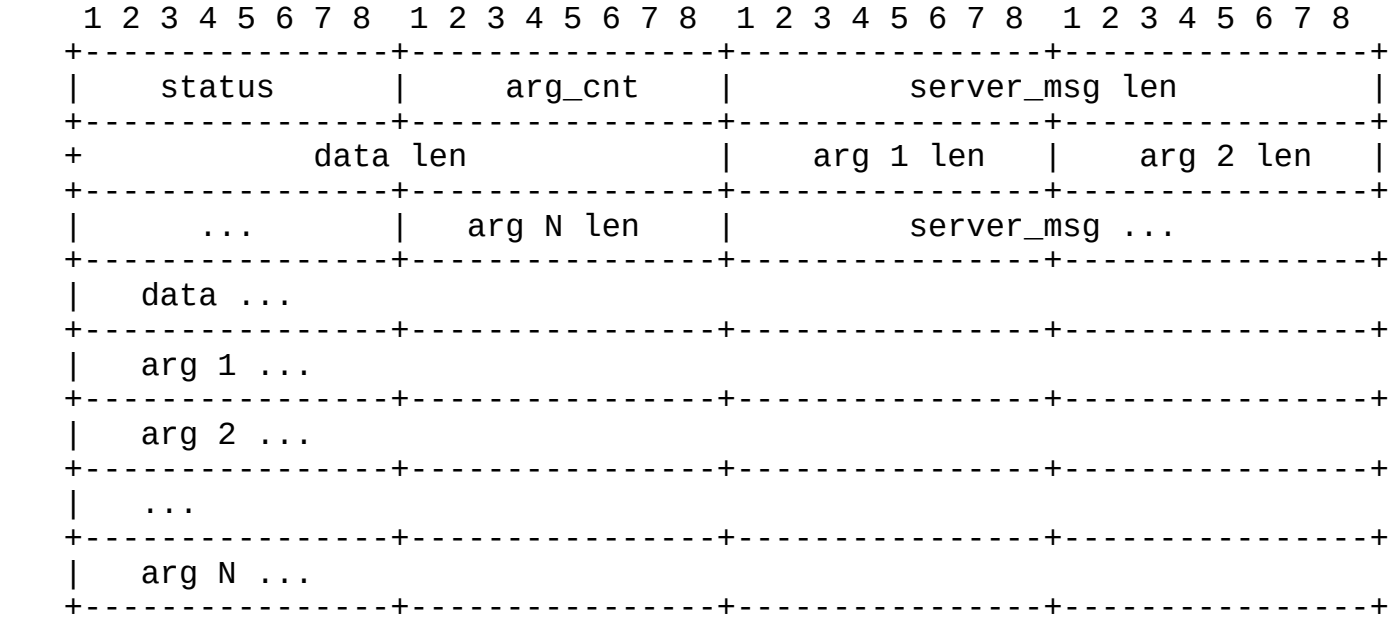
An attribute-value pair that describes the command to be performed. (see below)

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

Optional arguments are ones that may be disregarded by either client or server. Mandatory arguments require that the receiving side understands the attribute and will act on it. If the client receives a mandatory argument that it cannot oblige or does not understand, it MUST consider the authorization to have failed. It is legal to send an attribute-value pair with a NULL (zero length) value.

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.

5.2. 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.

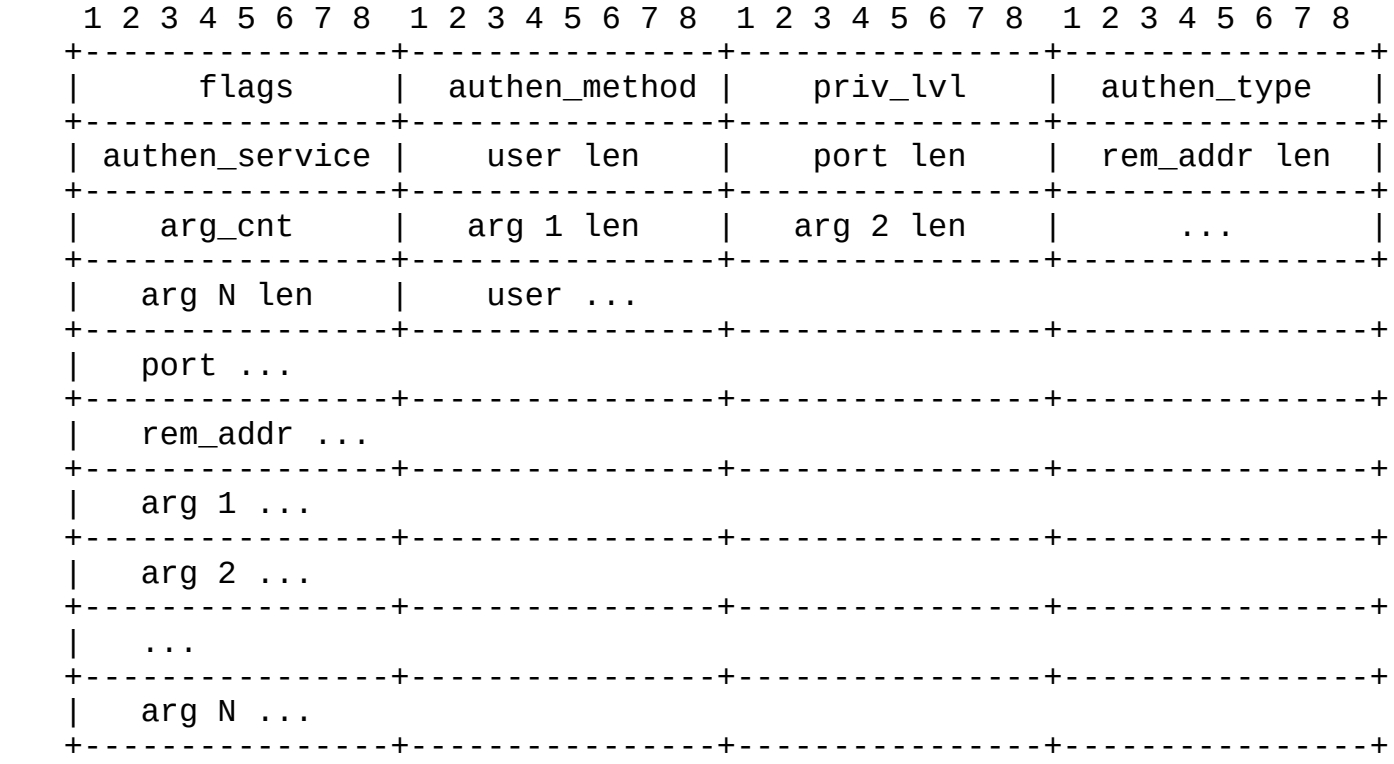
If the intended action is to approve the authorization with no modifications, then the status should be set to TAC_PLUS_AUTHOR_STATUS_PASS_ADD and the arg_cnt should be set to 0.

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

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, and must be ignored.

6. Accounting

6.1. The Account REQUEST Packet Body



flags

This holds bitmapped flags.

TAC_PLUS_ACCT_FLAG_START := 0x02

TAC_PLUS_ACCT_FLAG_STOP := 0x04

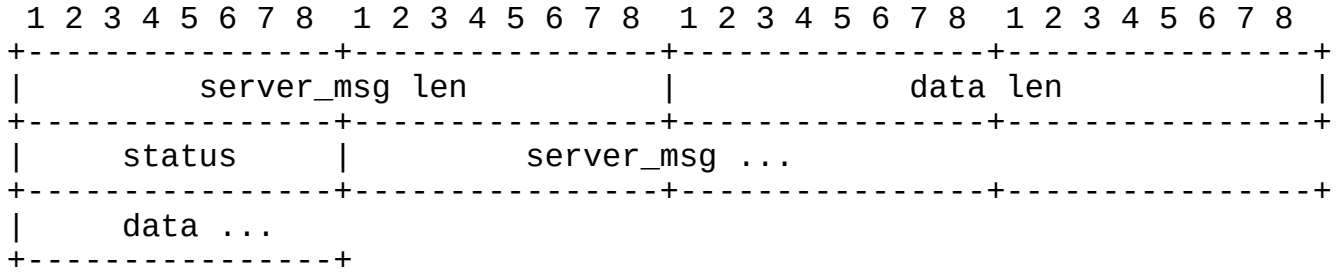
TAC_PLUS_ACCT_FLAG_WATCHDOG := 0x08

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

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

6.2. The Accounting REPLY Packet Body

The response to an accounting message is used to indicate that the accounting function on the server has completed. The server should reply with success only when the record has been committed to the required level of security, relieving the burden on the client from ensuring any better form of accounting is required.



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 server 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.

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. 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.

7. Attribute-Value Pairs

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

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

All boolean attributes are encoded with values "true" or "false".

It is recommended that hosts be specified as a numeric address so as to avoid any ambiguities.

Absolute times should be specified in seconds since the epoch, 12:00am Jan 1 1970. The timezone MUST be UTC unless a timezone attribute is specified.

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=".

7.1. Authorization Attributes

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", "shell", "tty-server", "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 client 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. Please refer to the Privilege Level section (Section 8) below.

remote_user

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

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 dialstring. 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.

7.2. Accounting Attributes

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 (Section 5) above.

task_id

Start and stop records for the same event MUST have matching task_id attribute values. The client must not reuse a specific task_id in a start record until it has sent a stop record for that task_id.

start_time

The time the action started ().

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.

8. Privilege Levels

The TACACS+ Protocol supports flexible authorization schemes through the extensible attributes. One scheme is built in to the protocol: Privilege Levels. 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. 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
```

If a client uses a different privilege level scheme, then it must map the privilege level to scheme above.

Privilege Levels are applied in two ways in the TACACS+ protocol:

- As an argument in authorization EXEC phase (when service=shell and cmd=NULL), where it is primarily used to set the initial privilege level for the EXEC session.

- In the packet headers for Authentication, Authorization and Accounting. The privilege level in the header is primarily significant in the Authentication phase for enable authentication where a different privilege level is required.

The use of Privilege levels to determine session-based access to commands and resources is not mandatory for clients, but it is in common use so SHOULD be supported by servers.

9. References

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

[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>.

[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>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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: aota@google.com

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

Phone: +44 0208 8244508
EMail: dcmgash@cisco.com

David Carrel

Lol Grant