Internet Engineering Task Force (IETF)
Request for Comments: 8907
Category: Informational
ISSN: 2070-1721

T. Dahm
A. Ota
Google Inc.
D.C. Medway Gash
Cisco Systems, Inc.
D. Carrel
IPsec Research
L. Grant

September 2020

The Terminal Access Controller Access-Control System Plus (TACACS+) Protocol

Abstract

This document describes the Terminal Access Controller Access-Control System Plus (TACACS+) protocol, which is widely deployed today to provide Device Administration for routers, network access servers, and other networked computing devices via one or more centralized servers.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are candidates for any level of Internet Standard; see Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc8907.

Table of Contents

1. Introduction

This document describes the Terminal Access Controller Access-Control System Plus (TACACS+) protocol. It was conceived initially as a general Authentication, Authorization, and Accounting (AAA) protocol. It is widely deployed today but is mainly confined for a specific subset of AAA called Device Administration, which includes authenticating access to network devices, providing central authorization of operations, and auditing of those operations.

A wide range of TACACS+ clients and servers is already deployed in the field. The TACACS+ protocol they are based on is defined in a document that was originally intended for IETF publication, but was never standardized. The document is known as "The Draft" [THE-DRAFT].

This Draft was a product of its time and did not address all of the key security concerns that are considered when designing modern standards. Therefore, deployment must be executed with care. These concerns are addressed in Section 10.

The primary intent of this informational document is to clarify the subset of "The Draft", which is common to implementations supporting Device Administration. It is intended that all implementations that conform to this document will conform to "The Draft". However, it is not intended that all implementations that conform to "The Draft" will conform to this document. The following features from "The Draft" have been removed:

* This document officially removes SENDPASS for security reasons.

* The normative description of legacy features such as the Apple Remote Access Protocol (ARAP) and outbound authentication has been removed.

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

The TACACS+ protocol allows for arbitrary length and content authentication exchanges to support alternative 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 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 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 deployment is not required to employ all three. Separating the elements is useful for the Device Administration use case, specifically, for authorization and accounting of individual commands in a session. Note that there is no provision made at the protocol level to associate authentication requests with authorization requests.

2. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Technical Definitions

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

3.1. Client

The client is any device that initiates TACACS+ protocol requests to mediate access, mainly for the Device Administration use case.

3.2. Server

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

3.3. Packet

All uses of the word packet in this document refer to TACACS+ protocol data units unless explicitly noted otherwise. The informal term "packet" has become an established part of the definition.

3.4. Connection

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

3.5. 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.

3.6. 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 an 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.

3.7. Treatment of Text Strings

The TACACS+ protocol makes extensive use of text strings. "The Draft" intended that these strings would be treated as byte arrays where each byte would represent a US-ASCII character.

More recently, server implementations have been extended to interwork with external identity services, and so a more nuanced approach is needed. Usernames MUST be encoded and handled using the UsernameCasePreserved Profile specified in [RFC8265]. The security considerations in Section 8 of [RFC8265] apply.

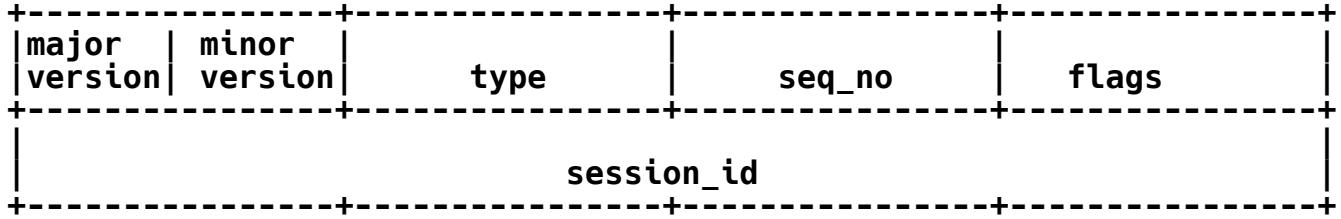
Where specifically mentioned, data fields contain arrays of arbitrary bytes as required for protocol processing. These are not intended to be made visible through user interface to users.

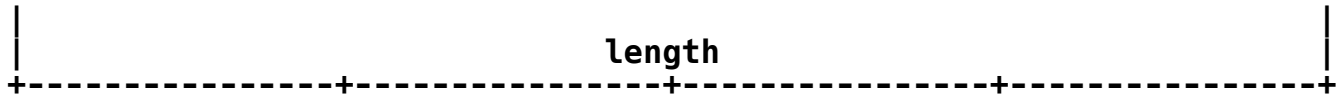
All other text fields in TACACS+ MUST be treated as printable byte arrays of US-ASCII as defined by [RFC0020]. The term "printable" used here means the fields MUST exclude the "Control Characters" defined in Section 5.2 of [RFC0020].

4. TACACS+ Packets and Sessions

4.1. The TACACS+ Packet Header

All TACACS+ packets begin with the following 12-byte header. The header describes the remainder of the packet:





The following general rules apply to all TACACS+ packet types:

* To signal that any variable-length data fields are unused, the corresponding length values are 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 field (and are not null terminated).

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

major_version

This is the major TACACS+ version number.

TAC_PLUS_MAJOR_VER := 0xc

minor_version

This is the minor TACACS+ version number.

TAC_PLUS_MINOR_VER_DEFAULT := 0x0

TAC_PLUS_MINOR_VER_ONE := 0x1

type

This is the packet type.

Options 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. TACACS+ 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. This option MUST NOT be used in production. The application of this flag will be covered in "Security Considerations" (Section 10).

This flag SHOULD be clear in all deployments. Modern network traffic tools support encrypted traffic when configured with the shared secret (see "Shared Secrets" (Section 10.5.1)), 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" (Section 4.3).

All other bits MUST be ignored when reading, and SHOULD be set to zero when writing.

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 [RFC4086].

length

The total length of the packet body (not including the header). Implementations MUST allow control over maximum packet sizes accepted by TACACS+ Servers. The recommended maximum packet size is 2^(16).

4.2. 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.

4.3. Single Connection Mode

Single Connection Mode is intended to improve performance where there is a lot of traffic between a client and a server by allowing the client to multiplex multiple sessions 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 Connection 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 MUST 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.

The TCP connection underlying the Single Connection Mode will close eventually either because of the timeout from the server or from an intermediate link. If a session is in progress when the client detects disconnect, then the client should handle it as described in "Session Completion" (Section 4.4). If a session is not in progress, then the client will need to detect this and restart the Single Connection Mode when it initiates the next session.

4.4. Session Completion

The REPLY packets defined for the packet 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 that is not aborted).

The server responds with a PASS or a FAIL to indicate that the processing of the request completed and that the client can apply the result (PASS or FAIL) to control the execution of the action that 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 cannot 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 that prompted the request should be executed.

Refer to "Aborting an Authentication Session" (Section 5.4.3) for details on handling additional status options.

When the session is complete, 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 "Single Connection Mode" (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 4.5)), 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 that 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 that establishes a connection but does not respond. The exact behavior is implementation specific. It is recommended that the client close the connection after a configurable timeout.

4.5. 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 each client. It is a site-dependent decision as to whether or not the use of separate keys is appropriate.

The flag field MUST be configured with TAC_PLUS_UNENCRYPTED_FLAG set to 0 so that the packet body is obfuscated by XORing it bytewise with

a pseudo-random pad:

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

The packet body can then be de-obfuscated by XORing it bytewise 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 [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 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 secrets it has configured for the device do not match, it MUST return ERROR. For details of TCP connection handling on ERROR, refer to "Session Completion" (Section 4.4).

```
TAC_PLUS_UNENCRYPTED_FLAG == 0x1
```

This option is deprecated and MUST NOT be used in production. In this case, the entire packet body is in cleartext. A 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 "Session Completion" (Section 4.4).

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

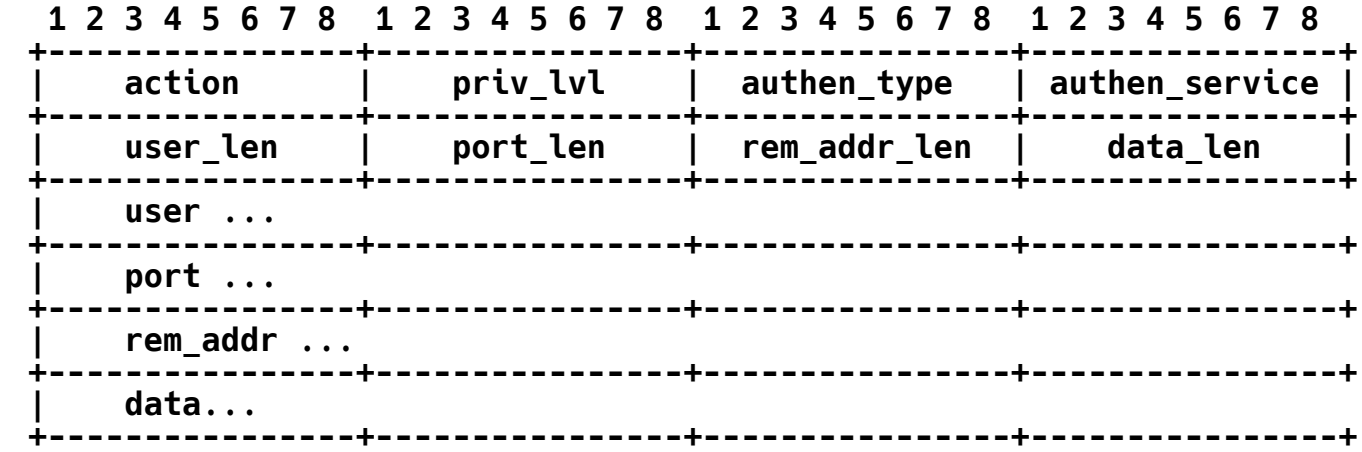
5. Authentication

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" (Section 6)). 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



Packet fields are as follows:

action

This indicates the authentication action.

Valid 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 "Privilege Levels" (Section 9).

authen_type

The type of authentication. Please see "Common Authentication Flows" (Section 5.4.2).

Valid values are:

TAC_PLUS_AUTHEN_TYPE_ASCII := 0x01

TAC_PLUS_AUTHEN_TYPE_PAP := 0x02

TAC_PLUS_AUTHEN_TYPE_CHAP := 0x03

TAC_PLUS_AUTHEN_TYPE_MSCHAP := 0x05

TAC_PLUS_AUTHEN_TYPE_MSCHAPV2 := 0x06

authen_service

This is the service that is requesting the authentication.

Valid 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_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 name of the client port on which the authentication is taking place. The value of this field is free-format text and is client specific. Examples of this argument include "tty10" to denote the tenth tty line, and "async10" to denote the tenth async interface. The client documentation SHOULD define the values and their meanings for this field. For details of text encoding, see "Treatment of Text Strings" (Section 3.7). The port_len indicates the length of the port field, in bytes.

rem_addr, rem_addr_len

A string indicating the remote location from which the user has connected to the client. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

When TACACS+ was used for dial-up services, this value contained the caller ID.

When TACACS+ is used for Device Administration, the user is normally connected via a network, and in this case, the value is intended to hold a network address, IPv4 or IPv6. For IPv6 address text representation defined, please see [RFC5952].

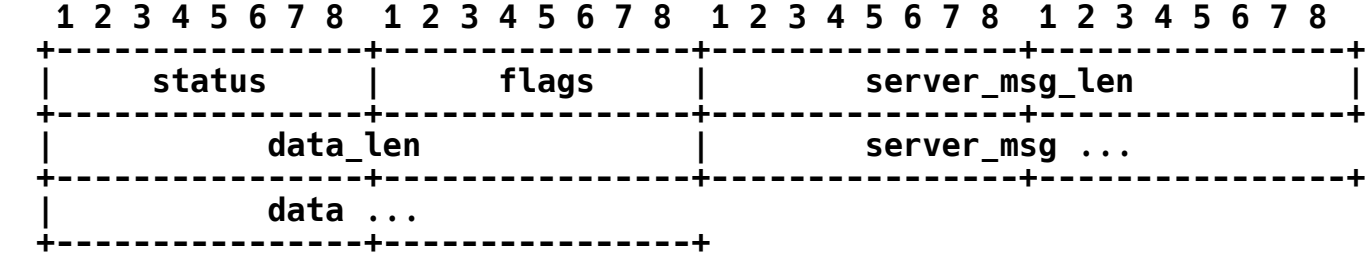
This field is optional (since the information may not be available). The rem_addr_len indicates the length of the user field, in bytes.

data, data_len

The data field is used to send data appropriate for the action and authen_type. It is described in more detail in "Common Authentication Flows" (Section 5.4.2). The data_len field 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.



status

The current status of the authentication.

Valid 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

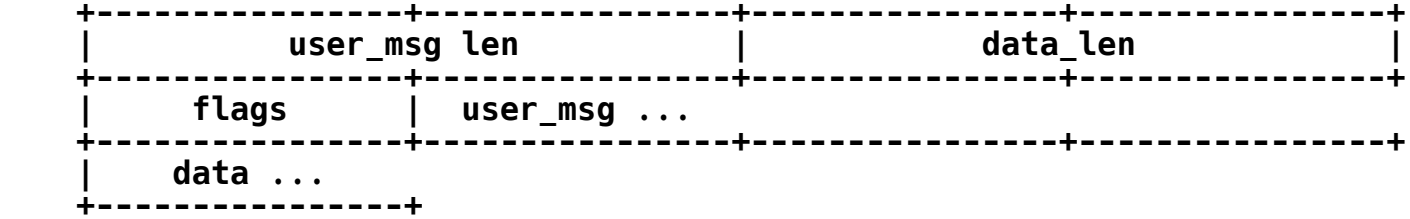
A message to be displayed to the user. This field is optional. The server_msg_len indicates the length of the server_msg field, in bytes. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

data, data_len

A field that holds data that is a part of the authentication exchange and is intended for client processing, not the user. It is not a printable text encoding. Examples of its use are shown in "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.



user_msg, user_msg_len

A field that 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. It is not a printable text encoding. 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 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, 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 a username and TAC_PLUS_AUTHEN_STATUS_GETPASS as a request for a 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 from 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 reflected in the user interface as it is entered.

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

5.4.1. Version Behavior

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 Challenge Handshake Authentication Protocol (CHAP) and Password Authentication Protocol (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

Table 1: TACACS+ Protocol Versioning

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

All authorization and accounting and ASCII authentication use minor_version 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; the 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 [RFC1334] (Obsolete). 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 PPP Authentication [RFC1334] and then compares 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 be 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 [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 [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 [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]. The TACACS+ server MUST reject 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 a text explaining the reason for the abort. This text will be handled by the server according to the requirements of the deployment. For details of text encoding, see "Treatment of Text Strings" (Section 3.7). For more details about session termination, refer to "Session Completion" (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] defined a mechanism to direct authentication requests to an alternative server. This mechanism is regarded as insecure, is deprecated, and is 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 a 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 the 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.

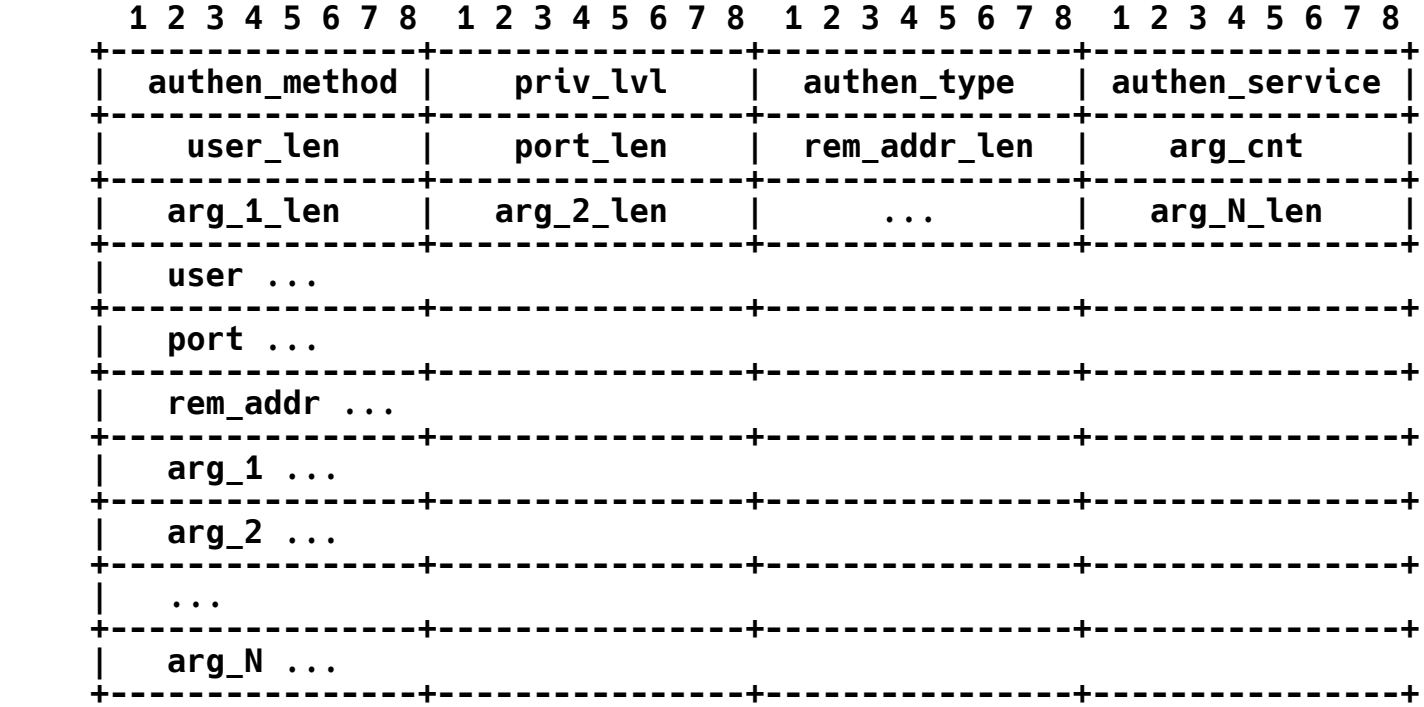
Authorization does not merely provide yes or no answers, but it may also customize the service for the particular user. A common use of authorization is to provision a shell session when a user first logs into a device to administer it. The TACACS+ server might respond to the request by allowing the service, but placing a time restriction on the login shell. For a list of common arguments used in authorization, see "Authorization Arguments" (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 (argument-value pairs) that can restrict or modify the client's actions.

6.1. The Authorization REQUEST Packet Body



authen_method

This field allows the client to indicate the authentication method used to acquire 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

As this information is not always subject to verification, it MUST NOT be used in policy evaluation. 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. RADIUS is the RADIUS authentication protocol. RCMD refers to authentication provided via the R-command protocols from Berkeley Unix. KRB5 [RFC4120] and KRB4 [KRB4] are Kerberos versions 5 and 4.

As mentioned above, this field is used by the client to indicate how it performed the authentication. One of the options (TAC_PLUS_AUTHEN_METH_TACACSPLUS := 0x06) is TACACS+ itself, and so the detail of how the client performed this option is given in "Authentication" (Section 5). For all other options, such as KRB and RADIUS, the TACACS+ protocol did not play any part in the authentication phase; as those interactions were not conducted using the TACACS+ protocol, they will not be documented here. For implementers of clients who need details of the other protocols, please refer to the respective Kerberos [RFC4120] and RADIUS [RFC3579] RFCs.

priv_lvl

This field is used in the same way as the priv_lvl field in authentication request and is described in "Privilege Levels" (Section 9). It indicates the user's current privilege level.

authen_type

This field corresponds to the authen_type field in "Authentication" (Section 5). 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 "Authentication" (Section 5). 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 "Authentication" (Section 5). 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 "Authentication" (Section 5). The rem_addr_len indicates the length of the port field, in bytes.

arg_cnt

This represents the number of authorization arguments to follow.

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

These 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 field (arg_1... arg_N) with a corresponding length field (which indicates the length of each argument in bytes).

The authorization arguments in both the REQUEST and the REPLY are argument-value pairs. The argument and the value are in a single string and are separated by either a "=" (0X3D) or a "*" (0X2A). The equals sign indicates a mandatory argument. The asterisk indicates an optional one. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

An argument name MUST NOT contain either of the separators. An argument value MAY 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.

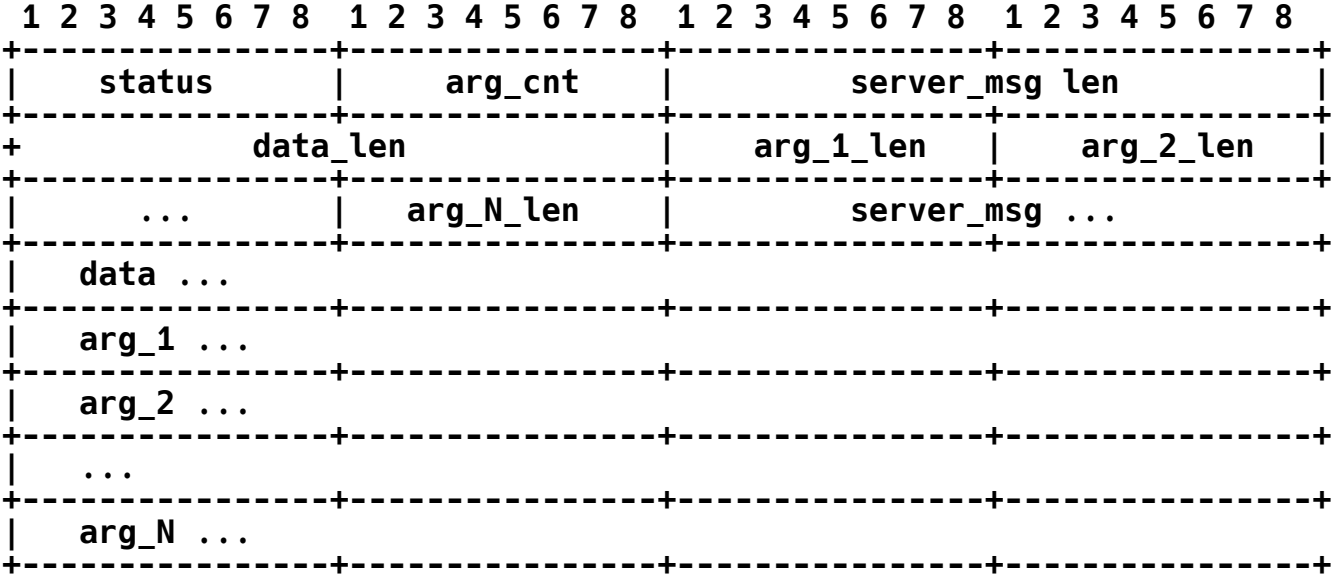
Optional arguments are ones that may be disregarded by either

client or server. Mandatory arguments require that the receiving side can handle the argument, 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. The value part of an argument-value pair may be empty, that is, the length of the value may be zero.

Argument-value strings are not NULL terminated; rather, their length value indicates their end. The maximum length of an argument-value string is 255 characters. The minimum is two characters (one name-value character and the separator).

Though the arguments allow extensibility, a common core set of authorization arguments SHOULD be supported by clients and servers; these are listed in "Authorization Arguments" (Section 8.2).

6.2. The Authorization REPLY Packet Body



status

This field indicates the authorization status.

TAC_PLUS_AUTHOR_STATUS_PASS_ADD := 0x01

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.

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.

TAC_PLUS_AUTHOR_STATUS_PASS_REPL := 0x02

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

TAC_PLUS_AUTHOR_STATUS_FAIL := 0x10

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

TAC_PLUS_AUTHOR_STATUS_ERROR := 0x11

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

TAC_PLUS_AUTHOR_STATUS_FOLLOW := 0x21

When the status equals TAC_PLUS_AUTHOR_STATUS_FOLLOW, 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.

server_msg, server_msg_len

This is a string that may be presented to the user. The server_msg_len indicates the length of the server_msg field, in bytes. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

data, data_len

This is a 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. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

arg_cnt

This represents the number of authorization arguments to follow.

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

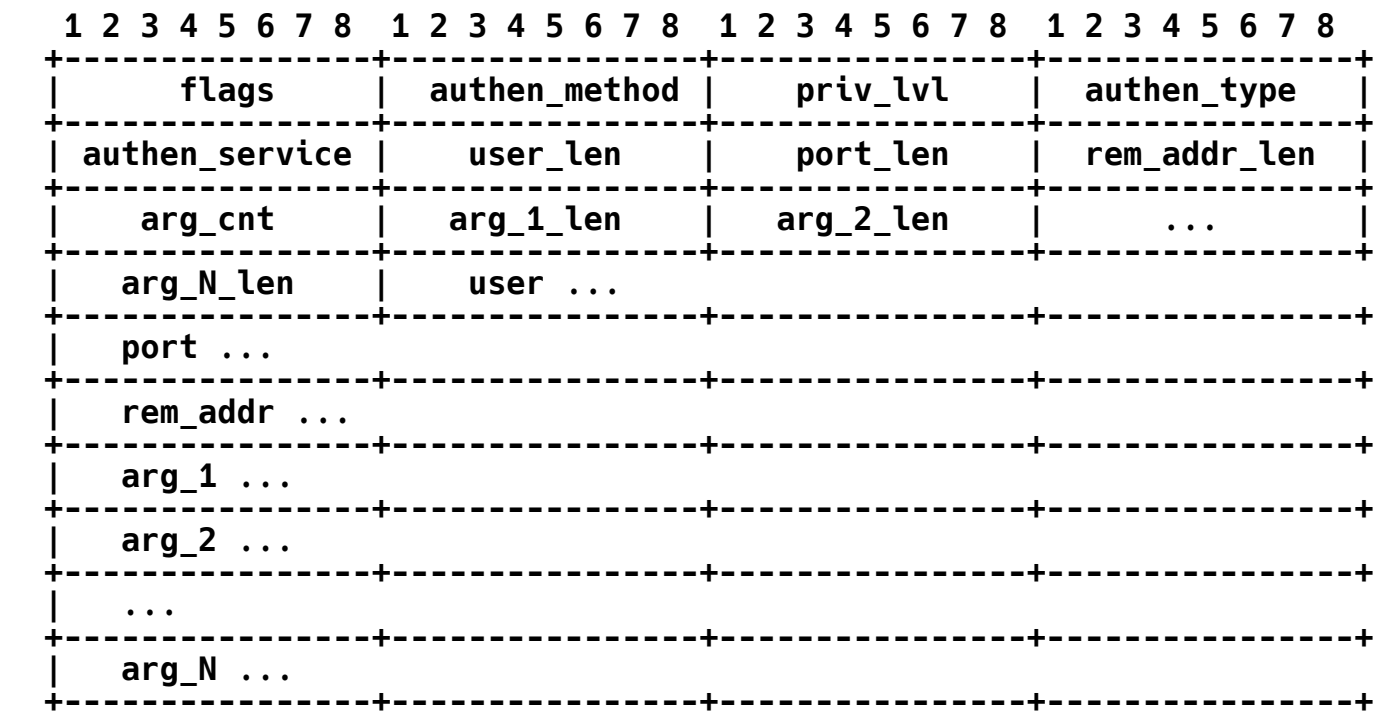
The arguments describe the specifics of the authorization that is being requested. For details of the content of the args, refer to "Authorization Arguments" (Section 8.2). Each argument is encoded in the packet as a single arg field (arg_1... arg_N) with a corresponding length field (which indicates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

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, and 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 arguments is defined in "Accounting Arguments" (Section 8.3).

7.1. The Account REQUEST Packet Body



flags

This holds bitmapped flags.

Valid values are:

TAC_PLUS_ACCT_FLAG_START := 0x02

TAC_PLUS_ACCT_FLAG_STOP := 0x04

TAC_PLUS_ACCT_FLAG_WATCHDOG := 0x08

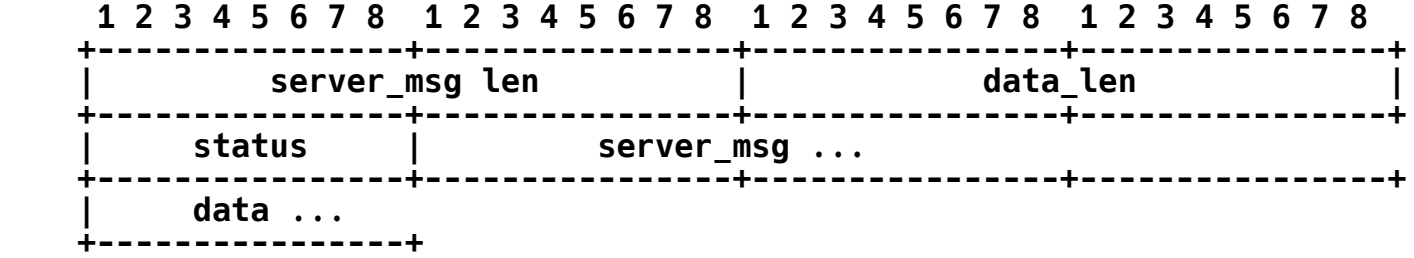
All other fields are defined in "Authentication" (Section 5) and "Authorization" (Section 6) and have the same semantics. They provide details for the conditions on the client, and authentication

context, so that these details may be logged for accounting purposes.

See "Accounting Arguments" (Section 8.3) for the dictionary of arguments 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.



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

When the status equals TAC_PLUS_ACCT_STATUS_FOLLOW, the actions to be taken and the contents of the data field are identical to the TAC_PLUS_AUTHEN_STATUS_FOLLOW status for authentication.

server_msg, server_msg_len

This is a string that may be presented to the user. The server_msg_len indicates the length of the server_msg field, in bytes. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

data, data_len

This is a 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. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

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" [THE-DRAFT] 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.

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

Table 2: Summary of Accounting Packets

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. Argument-Value Pairs

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

8.1. Value Encoding

All argument values are encoded as strings. For details of text encoding, see "Treatment of Text Strings" (Section 3.7). The following type representations SHOULD be followed.

Numeric

All numeric values in an argument-value string are provided as decimal numbers, unless otherwise stated. All arguments include a length field, and TACACS+ implementations MUST verify that they can accommodate the lengths of numeric arguments before attempting to process them. If the length cannot be accommodated, then the argument MUST be regarded as not handled and the logic in "Authorization" (Section 6.1) regarding the processing of arguments MUST be applied.

Boolean

All Boolean arguments are encoded with values "true" or "false".

IP-Address

It is recommended that hosts be specified as an IP address so as to avoid any ambiguities. For details of text encoding, see "Treatment of Text Strings" (Section 3.7). IPv4 addresses are specified as octet numerics separated by dots ('.'). IPv6 address text representation is defined in [RFC5952].

Date Time

Absolute date/times are specified in seconds since the epoch, 12:00am, January 1, 1970. The time zone MUST be UTC unless a time zone argument is specified.

String

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

Empty Values

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

8.2. Authorization Arguments

service (String)

The primary service. Specifying a service argument indicates that this is a request for authorization or accounting of that service. For example: "shell", "tty-server", "connection", "system" and "firewall"; others may be chosen for the required application. This argument MUST always be included.

protocol (String)

A field that 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" argument 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 or 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 arguments may be specified, and they are order dependent.

acl (Numeric)

A number representing a connection access list. Applicable only to session-based shell authorization. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

inacl (String)

The identifier (name) of an interface input access list. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

outacl (String)

The identifier (name) of an interface output access list. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

addr (IP-Address)

A network address.

addr-pool (String)

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

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 the 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)

The privilege level to be assigned. Please refer to "Privilege Levels" (Section 9).

8.3. Accounting Arguments

The following arguments are defined for TACACS+ accounting only. They MUST precede any argument-value pairs that are defined in "Authorization" (Section 6).

task_id (String)

Start and stop records for the same event MUST have matching task_id argument values. The client MUST ensure that active task_ids are not duplicated; a client MUST NOT reuse a task_id in 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 time zone abbreviation for all timestamps included in this packet. A database of time zones is maintained in [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 argument. 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 string describing the status of the action. For details of text encoding, see "Treatment of Text Strings" (Section 3.7).

Where the TACACS+ deployment is used to support the Device Administration use case, it is often required to log all commands entered into client devices. To support this mode of operation, TACACS+ client devices MUST be configured to send an accounting start packet for every command entered, irrespective of how the commands were authorized. These "Command Accounting" packets MUST include the

"service" and "cmd" arguments, and if needed, the "cmd-arg" arguments detailed in Section 8.2.

9. Privilege Levels

The TACACS+ protocol supports flexible authorization schemes through the extensible arguments.

The privilege levels scheme is built into the protocol and has been extensively used as an option for Session-based shell authorization. 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 specific commands) to different privilege levels. The allocation of commands to privilege levels is highly dependent upon the deployment. Common allocations are as follows:

TAC_PLUS_PRIV_LVL_MIN := 0x00. The level normally allocated to an unauthenticated session.

TAC_PLUS_PRIV_LVL_USER := 0x01. The level normally allocated to a regular authenticated session.

TAC_PLUS_PRIV_LVL_ROOT := 0x0f. The level normally allocated to a session authenticated by a highly privileged user to allow commands with significant system impact.

TAC_PLUS_PRIV_LVL_MAX := 0x0f. The highest privilege level.

A privilege level can be assigned to a shell (exec) session when it starts. 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 is required to perform actions that are mapped to a higher privilege level, 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 requests.

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. However, it is still common enough that it SHOULD be supported by servers.

10. Security Considerations

"The Draft" [THE-DRAFT] from 1998 did not address all of the key security concerns that are considered when designing modern standards. This section addresses known limitations and concerns that 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 Draft" [THE-DRAFT] 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

The TACACS+ protocol does not include a security mechanism that would meet modern-day requirements. These security mechanisms would be best referred to as "obfuscation" and not "encryption", since they provide no meaningful integrity, privacy, or replay protection. An attacker with access to the data stream should be assumed to be able to read and modify all TACACS+ packets. Without mitigation, a range of risks such as the following are possible:

* Accounting information may be modified by the man-in-the-middle attacker, making such logs unsuitable and not trustable for auditing purposes.

* 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 that may decrease the cost of brute-force attacks.

* Chosen plaintext attacks that may decrease the cost of a brute-force attacks.

* No forward secrecy.

Even though, to the best knowledge of the 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 the 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 that 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 "TACACS+ Best Practices" (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 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 MUST 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 "TACACS+ Best Practices" (Section 10.5)) as it's trivial to execute a successful man-in-the-middle attack 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:

* A 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, first 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 SHOULD be used via a secure transport (see "TACACS+ Best Practices" (Section 10.5)). Although Accounting sessions are not directly involved in authentication or authorizing operations on the device, man-in-the-middle attackers may do any of the following:

* Replace accounting data with new valid values or garbage that can

confuse auditors or hide information related to their authentication and/or authorization attack attempts.

* Try and poison an accounting log with entries designed to make systems behave in unintended ways (these systems could be TACACS+ servers and any other systems that would manage accounting entries).

In addition to these direct manipulations, different client implementations pass a different fidelity of accounting data. Some vendors have been observed in the wild that pass sensitive data like passwords, encryption keys, and the like as part of the accounting log. Due to a lack of strong encryption with perfect forward secrecy, this data may be revealed in the 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. TACACS+ MUST be used within a secure deployment; TACACS+ MUST be deployed over networks that ensure privacy and integrity of the communication and MUST be deployed over a network that is separated from other traffic. Failure to do so will impact overall network security.

The following recommendations impose restrictions on how the protocol is applied. These restrictions were not imposed in "The Draft". New implementations, and upgrades of current implementations, MUST implement these recommendations. Vendors SHOULD provide mechanisms to assist the administrator to achieve these best practices.

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 MUST NOT expose shared secrets in logs.

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

* TACACS+ server management systems MUST provide a mechanism to track secret key lifetimes and notify administrators to update them periodically. TACACS+ server administrators SHOULD change secret keys at regular intervals.

* 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 a shared secret key or shared key that is less than 16 characters long.

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

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 that have TAC_PLUS_UNENCRYPTED_FLAG set. There MUST always be 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. Clients MUST be implemented in a way that requires explicit configuration to enable the use of TAC_PLUS_UNENCRYPTED_FLAG. This option MUST NOT be used when the client is in production.

When a TACACS+ client receives responses from servers where:

* the response packet was received from the server configured with a shared key, but the packet has TAC_PLUS_UNENCRYPTED_FLAG set, and

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

the TACACS+ client MUST close the 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+ administrators select 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 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 argument names. The cost of the flexibility is that administrators and implementers MUST ensure that the argument and value pairs shared between the clients and servers have consistent interpretation.

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

10.5.5. Redirection Mechanism

"The 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 MUST 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. IANA Considerations

This document has no IANA actions.

12. References

12.1. Normative References

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://www.rfc-editor.org/info/rfc20>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

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

[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, <https://www.rfc-editor.org/info/rfc2433>.

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

[RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, DOI 10.17487/RFC3579, September 2003, <https://www.rfc-editor.org/info/rfc3579>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8265] Saint-Andre, P. and A. Melnikov, "Preparation, Enforcement, and Comparison of Internationalized Strings Representing Usernames and Passwords", RFC 8265,

DOI 10.17487/RFC8265, October 2017,
<https://www.rfc-editor.org/info/rfc8265>.

12.2. Informative References

[KRB4] Miller, S., Neuman, C., Schiller, J., and J. Saltzer, "Section E.2.1: Kerberos Authentication and Authorization System", MIT Project Athena, Cambridge, Massachusetts, December 1987.

[THE-DRAFT] Carrel, D. and L. Grant, "The TACACS+ Protocol Version 1.78", Work in Progress, Internet-Draft, draft-grant-tacacs-02, January 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>.

Acknowledgements

The authors would like to thank the following reviewers whose comments and contributions made considerable improvements to this document: Alan DeKok, Alexander Clouter, Chris Janicki, Tom Petch, Robert Drake, and John Heasley, 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 thank the support from the OPSAWG Chairs and advisors, especially Joe Clarke.

Authors' Addresses

Thorsten Dahm
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: thorstendlux@google.com

Andrej Ota
Google Inc.
1600 Amphitheatre Parkway
Mountain View, CA 94043
United States of America

Email: andrej@ota.si

Douglas C. Medway Gash
Cisco Systems, Inc.
170 West Tasman Dr.
San Jose, CA 95134
United States of America

Email: dcmgash@cisco.com

David Carrel
IPsec Research

Email: carrel@ipsec.org

Lol Grant

Email: lol.grant@gmail.com