Training: Understanding SIP Packets

Created by Pedro Geraldo on [2019-08-26](190826-understand-sip-packets.docx)

Table of Contents

[Introduction 1](#_Toc20147834)

[Requirements 1](#_Toc20147835)

[SIP Packets 1](#_Toc20147836)

[Basic Concepts 1](#_Toc20147837)

[User Agents 1](#_Toc20147838)

[Proxy Servers 1](#_Toc20147839)

[What are SIP Messages 1](#_Toc20147840)

[SIP Request 1](#_Toc20147841)

[SIP Responses 1](#_Toc20147842)

[Validation 1](#_Toc20147843)

[References 1](#_Toc20147844)

[End of document 1](#_Toc20147845)

# Introduction

In this training we will talk about the different SIP packets and how to detect them.

## Requirements

Training: [SSH connection](https://docs.switzernet.com/3/support/101014-training-connection-sip/)

Training: [Simple Unix Bash commands](https://docs.switzernet.com/3/support/people/elen-virabyan/100906-simple-unix-bash-commands/)

Training: [Ngrep: watch telnet packets](https://docs.switzernet.com/3/support/110224-ngrep-telnet/)

Training: [Ngrep: watch ftp packets](https://docs.switzernet.com/3/support/110225-ngrep-ftp/)

Training: [Understanding ngrep](https://docs.switzernet.com/3/public/190903-training-understanding-ngrep/)

# SIP Packets

## Basic Concepts

### User Agents

A normal device capable of listening and emitting SIP packets. A device can act as both a user agent client and user agent server for SIP transactions:

* User Agent Server (UAS): A user agent server is a logical entity that generates a response to a SIP request.
* User Agent Client (UAC): A user agent client is a logical entity that creates a new request, and then uses the client transaction state machinery to send it. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction.

### Proxy Servers

An intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity "closer" to the targeted user. Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it. A proxy server can have stateless or stateful properties

* Stateless Proxy: A logical entity that does not maintain the client or server transaction state machines defined in this specification when it processes requests. A stateless proxy forwards every request it receives downstream and every response it receives upstream.
* Stateful Proxy: A logical entity that maintains the client and server transaction state machines defined by this specification during the processing of a request, also known as a transaction stateful proxy. A (transaction) stateful proxy is not the same as a call stateful proxy.

## What are SIP Messages

A SIP message is either a request from a client to a server, or a response from a server to a client.

Both **Request** and **Response** messages use the basic format of RFC 2822 Internet Message Format, even though the syntax differs in character set and syntax specifics (i.e. SIP allows header fields that would not be valid RFC 2822 header fields, for example.)

Both types of messages consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and an optional message-body.

generic-message = start-line

\*message-header

CRLF

[ message-body ]

start-line = Request-Line / Status-Line

The start-line, each message-header line, and the empty line MUST be terminated by a carriage-return line-feed sequence (CRLF). Note that the empty line MUST be present even if the message-body is not.

Except for the above difference in character sets, much of SIP’s message and header field syntax is identical to HTTP/1.1. But KEEP IN MIND, SIP is not an extension of HTTP.

### SIP Request

SIP requests are distinguished by having a Request-Line for a start-line. A Request-Line contains a method name, a Request-URI, and the protocol version separated by a single space (SP) character.

The Request-Line ends with CRLF. No carriage-return (CR) or line-feed (LF) are allowed except in the end-of-line CRLF sequence. No linear whitespace (LWS) is allowed in any of the elements.

Request-Line = **Method** SP **Request-URI** SP **SIP-Version** CRLF

Method

This specification defines six methods: REGISTER for registering contact information, INVITE, ACK, and CANCEL for setting up sessions, BYE for terminating sessions, and OPTIONS for querying servers about their capabilities. SIP extensions, documented in standards track RFCs, may define additional methods.

For now, there are fourteen SIP Request methods of which the first six are the most basic request / method types:

* **INVITE** = Establishes a session.
* **ACK** = Confirms an INVITE request.
* **BYE** = Ends a session.
* **CANCEL** = Cancels establishing of a session.
* **REGISTER** = Communicates user location (host name, IP).
* **OPTIONS** = Communicates information about the capabilities of the calling and receiving SIP phones.
* **PRACK =** Provisional Acknowledgement.
* **SUBSCRIBE =** Subscribes for Notification from the notifier.
* **NOTIFY =** Notifies the subscriber of a new event.
* **PUBLISH =** Publishes an event to the Server.
* **INFO =** Sends mid session information.
* **REFER =** Asks the recipient to issue call transfer.
* **MESSAGE =** Transports Instant Messages.
* **UPDATE =** Modifies the state of a session.

#### Request-URI

The Request-URI is a SIP or SIPS URI as described in Section 19.1 or a general URI. It indicates the user or service to which this request is being addressed. The Request-URI MUST NOT contain unescaped spaces or control characters and MUST NOT be enclosed in "<>".

SIP elements MAY support Request-URIs with schemes other than "sip" and "sips", for example the "tel" URI scheme of RFC 2806. SIP elements MAY translate non-SIP URIs using any mechanism at their disposal, resulting in SIP URI, SIPS URI, or some other scheme.

#### SIP-Version

Both request and response messages include the version of SIP in use regarding version ordering, compliance requirements, and upgrading of version numbers. To be compliant with this specification, applications sending SIP messages MUST include a SIP-Version of "SIP/2.0". The SIP-Version string is case-insensitive, but implementations MUST send upper-case. Unlike HTTP/1.1, SIP treats the version number as a literal string. In practice, this should make no difference.

### SIP Responses

SIP responses are distinguished from requests by having a Status-Line as their start-line. A Status-Line consists of the protocol version followed by a numeric Status-Code and its associated textual phrase, with each element separated by a single SP character.

No CR or LF is allowed except in the final CRLF sequence.

Status-Line = **SIP-Version** SP **Status-Code** SP **Reason-Phrase** CRLF

The Status-Code is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata, whereas the Reason-Phrase is intended for the human user. A client is not required to examine or display the Reason-Phrase.

While this specification suggests specific wording for the reason phrase, implementations MAY choose other text, for example, in the language indicated in the Accept-Language header field of the request.

The first digit of the Status-Code defines the class of response. The last two digits do not have any categorization role. For this reason, any response with a status code between 100 and 199 is referred to as a "1xx response", any response with a status code between 200 and 299 as a "2xx response", and so on. SIP/2.0 allows six values for the first digit:

* **1xx** = Informational responses, such as 180 (ringing).
* **2xx** = Success responses.
* **3xx** = Redirection responses.
* **4XX** = Request failures.
* **5xx** = Server errors.
* **6xx** = Global failures.

# Validation

For validation, first ask permission to use one of our SIP Proxy servers (Kamailio Server), and then create a document with a header and footer according to the [guidelines](https://docs.switzernet.com/3/company/181003-how-to-work-with-documentations/) and place screenshots with the results of the tasks in a dedicated chapter named “Validation”:

1. In your Ubuntu console, apply a timestamp followed by an echo of your name and the ssh to the server;
2. Execute a ngrep of the first 5 SIP packets that pass;
3. Identify on each packet if it is a SIP Request or a SIP Response, and which one it is.

# References

<http://www.unappel.ch/2/support/100722-training-employees/i/>

<https://www.ietf.org/rfc/rfc3261.txt.pdf>

# End of document

\* \* \*

© 4z.com