SIPvestigator: A CLI Based SIP Session Diagnostics Utility

Örn Arnarson

Tölvunarfræði
Háskóli Íslands
2018

SIPvestigator: A CLI Based SIP Session Diagnostics Utility

Örn Arnarson

6 eininga ritgerð sem er hluti af
Baccalaureus Scientiarum gráðu í Tölvunarfræði

Leiðbeinendur
Jón Ingi Einarsson
Hjálmtýr Hafsteinsson

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Verkfræði- og náttúruvísindasvið
Háskóli Íslands
Reykjavík, maí 2018

SIPvestigator: A CLI Based SIP Session Diagnostics Utility
SIPvestigator
6 eininga ritgerð sem er hluti af *Baccalaureus Scientiarum* gráðu í Tölvunarfræði

Höfundarréttur © 2018 Örn Arnarson
Öll réttindi áskilin

Iðnaðarverkfræði-, vélaverkfræði- og tölvunarfræðideild
Verkfræði- og náttúruvísindasvið
Háskóli Íslands
Hjarðarhaga 6
107 Reykjavík

Sími: 525 4000

Skráningarupplýsingar:
Örn Arnarson, 2018, *SIPvestigator: A CLI Based SIP Session Diagnostics Utility*, BS ritgerð, Iðnaðarverkfræði-, vélaverkfræði- og tölvunardeild, Háskóli Íslands, 21 bls.

Prentun: Háskólaprent
Reykjavík, maí 2018

Abstract

SIPvestigator is a utility to view and analyze SIP communications on the command line. It is written in the Python programming language. SIP (Session Initiation Protocol) is a protocol that is used to initiate a phone call or a video call between two or more endpoints. SIPvestigator loads data containing SIP communications and groups messages into sessions (or "conversations") between parties that can be searched or filtered for certain conditions. The primary goal of the utility is to make analysis of problematic communications easier by offering filtration of the data (which can be quite voluminous) based on its contents, directly on the hardware collecting the data, provided that the hardware is capable of running Python.

This paper describes the design and implementation of SIPvestigator and the problems sought to solve with the utility.

Útdráttur

SIPvestigator er tól til að rýna í og greina SIP samskipti á skipanalínu. SIPvestigator er skrifaður í Python forritunarmálinu. SIP (e. Session Initiation Protocol) er samskiptastaðall sem er m.a. notaður í að koma á símtali eða myndsímtali milli tveggja eða fleiri aðila. SIPvestigator les inn gögn með SIP samskiptum og flokkar skeyti niður í „samtöl" á milli aðila sem hægt er að leita í eða sía með ákveðnum skilyrðum. Helsta markmið tólsins er að auðvelda greiningu vandamála sem komið geta upp með því að bjóða upp á síun gagna (sem geta orðið heilmikil) út frá innihaldi gagnanna, beint á þeim vélbúnaði þar sem gögnunum er safnað, að því gefnu að vélbúnaðurinn geti keyrt Python.

Í þessari ritgerð er farið yfir helstu þætti í hönnun og útfærslu SIPvestigator ásamt því að lýsa þeim vandamálum sem leitast er við að leysa með tólinu.

Contents

List of Figures ..... **vi**

Abbreviations ..... **vii**

1 Introduction **1**

2 The SIPvestigator Utility **3**
2.1 Supported Datatypes 3
2.1.1 Loading Data from a Plain-Text File 3
2.1.2 Loading Data from a PCAP File 3
2.1.3 Live Capture 3
2.2 Data Organization 4
2.3 CLI Interface 5
2.3.1 Loading Data 6
2.3.2 Capturing Data 6
2.3.3 Creating Filters 6
2.3.4 Showing Conversations 8
2.3.5 Showing Conversation Message Overview 8
2.3.6 Showing Conversation Message Contents 9
2.4 Common Problems and Practical Examples 11
2.4.1 NAT Timeout or Missing Port-Forward 11
2.4.2 NAT Timeout Example 12
2.4.3 Improper Caller-ID Configuration 14
2.4.4 Improper Caller-ID Configuration Example 15
2.4.5 Missing or Wrong Diversion Headers 16
2.4.6 Wrong Diversion Header Example 17

3 Conclusion **19**

References **21**

List of Figures

Figure 1 - Basic Call Flow 4
Figure 2 - Example SIP Invite 5
Figure 3 - Help Menu 5
Figure 4 - Help Menu for Show Command 6
Figure 5 - Loading Data 6
Figure 6 - Capturing Data 6
Figure 7 - Filtering Data Using ‘and’ Condition 7
Figure 8 - Filtering Data Using ‘or’ Condition 7
Figure 9 - Filter Help Menu 7
Figure 10 - Filter Help Menu for Add Command 8
Figure 11 - Showing Matching Conversations in Order of Arrival 8
Figure 12 - Showing Matching Conversations in Alphabetical Order of Call-ID 8
Figure 13 - Specific Conversation Overview 9
Figure 14 - Show Conversation Messages 10
Figure 15 - Show Specific Conversation Message 10
Figure 16 - NAT Timeout 12
Figure 17 - Loading, Filtering and Glancing at NAT Timeout Data 13
Figure 18 - NAT Timeout Conversation Overview 13
Figure 19 - NAT Timeout SIP Invite 14
Figure 20 - Loading and Filtering Improper Caller-ID Data 15
Figure 21 - Improper Caller-ID Conversation Overview 15
Figure 22 - Improper Caller-ID Conversation Details 16
Figure 23 - Improper Caller-ID INVITE message (message clipped before SDP) 16
Figure 24 - Loading and Filtering Wrong Diversion Header 17
Figure 25 - Filtering Wrong Diversion Header with Has Diversion and Dstnum 17
Figure 26 - Wrong Diversion Header Conversation Details 18
Figure 27 - Wrong Diversion Header INVITE message 18

Abbreviations

ALG: Application Layer Gateway
B2BUA: Back-to-back User Agent
CLI: Command-line Interface
HTTP: Hypertext Transfer Protocol
IETF: Internet Engineering Task Force
IP: Internet Protocol
LTE: Long-Term Evolution
NAT: Network Address Translation
PBX: Private Branch Exchange
PSTN: Public Switched Telephone Network
PCAP: Packet Capture File
RTP: Real-Time Transport Protocol
SDP: Session Description Protocol
SIP: Session Initiation Protocol
SRTP: Secure Real-Time Transport Protocol
TCP: Transmission Control Protocol
TXT: Text File
UDP: User Datagram Protocol
VoIP: Voice over IP
VoLTE: Voice over LTE

1 Introduction

SIP is a widely used protocol for VoIP telephony at the time of writing. Since the late 2000s its use has become widespread, originally starting mostly in the corporate realm on company PBX's. Telecommunications companies are increasingly favoring interconnecting with other telecommunications companies via SIP trunking[1] instead of using traditional circuits due to the lower costs of running SIP trunks. This is especially true for international interconnections, where long-distance leased lines are exceedingly expensive, but SIP trunks can utilize pre-existing infrastructure used for the Internet.

The residential sector has not seen much VoIP adoption until recently, as residences already had existing fixed-line services and the cost decrease of switching to VoIP was marginal for the end user. Fixed-line usage has rapidly decreased in the last few years with users adopting cellular telephony instead[2]. Exceptions to this are indirect VoIP use, where users make calls that get converted to VoIP along the way due to PSTN (circuit switched) network and VoIP network interconnects (and perhaps converted back to traditional fixed-line services on the other end), and users wanting to have national "fixed-line" services with flexible locations. This permits users to take their fixed-line number with them overseas, for example.

In recent years and months as of this writing, though, the private sector has seen a rapid adoption of VoIP technology that relies on the SIP protocol through the adoption of VoLTE on 4G cellular networks. It is likely that within a few years' time, the majority of cellular calls will be Voice over LTE (or LTEs succeeding technologies).

The SIP protocol[3] itself, mostly designed and standardized by the IETF, is modeled after the HTTP protocol. It is a request and response-based protocol; a client will send a request to a server and receive some sort of response. The number of messages (requests and responses) for a single phone call can number from 7 to multiple 10s of messages, which means that whoever might be trying to debug a problem might quickly lose oversight of the data.

The SIP protocol is just a control protocol. The media for the phone calls (the speech) is not part of the SIP protocol. Another protocol, called RTP[4] (or SRTP, if the media is encrypted) is used, while the SIP protocol sets up the clients for media transmission by specifying which IP addresses, ports, codecs, etc. will be used in a negotiation between the two clients. The transmission of the media is then an entirely separate process, though under the control of SIP.

---

[1] SIP trunking has a rather vague meaning, but generally it means an interconnect between two parties, often with an upper limit to the number of concurrent calls. It is usually trusted from a protocol perspective, in that it requires no protocol-based authentication, but rather relies on IP addresses or VPN tunnels etc.

[2] The number of active fixed-line services in Iceland decreased by about 40.000 lines from the year 2008 to 2016, while mobile subscriptions increased by about 125.000 subscriptions in the same period. VoIP services increased by 50.000 subscriptions in the same period. [1][2]

The SIPvestigator utility is designed with the service provider in mind, where data captures are usually done in some central system that has 100s or 1000s of SIP requests per second, with the aim of giving the user a clear overview of the data and providing searchability via filters that make sense to the user. Its benefit to the user is that data can be analyzed directly on the server responsible for capturing the data, instead of transferring the data out from the server and loading it up in an analysis tool like Wireshark[3]. Further, SIPvestigator is application aware, and can provide easy filtering of things like calling-telephone number, called-telephone number, the presence of certain headers or their contents, which can be difficult to filter out with a tool like Wireshark.

SIPvestigator can be downloaded from the project's GitHub[4] page.

[3] Wireshark is a widely used cross-platform graphical network protocol analyzer, which lets users do deep inspection of network communications on hundreds of protocols.
[4] https://github.com/orn1983/sipvestigator

2 The SIPvestigator Utility

2.1 Supported Datatypes

The SIPvestigator utility supports importing data via three methods:

1. Loading data from a plain-text file with a .txt suffix
2. Loading data from a PCAP file with a .pcap suffix
3. Capturing data directly by sniffing networking traffic

2.1.1 Loading Data from a Plain-Text File

The program expects the plain-text file to contain something that looks like the output one might get from the tcpdump utility, when tcpdump is set to dump data directly to the terminal. The reason for this is that often when doing debugging, one might start with running tcpdump directly, not expecting a large amount of data, but then having far more data than expected. Dumping this data to a file would then allow one to subsequently load the data into SIPvestigator for parsing and filtering. Further, many phone systems provide their debugging data in exactly this data format, which makes it possible to analyze said data with SIPvestigator.

2.1.2 Loading Data from a PCAP File

SIPvestigator can also load PCAP files directly. These files are usually produced by programs like tcpdump with the –w option added, or Wireshark. Other utilities often support exporting PCAP files as well. SIPvestigator uses the python module dpkt[5] to load PCAP files. Unfortunately, dpkt uses a lot of memory when loading PCAP files, so loading PCAP files larger than 50 MB is not recommended.

2.1.3 Live Capture

There is a third method of loading data into the program, and that is by having the program capture live network packets as they are seen on the wire. In this case, another PCAP library called scapy[6] is used, and thus nothing needs to be saved to file in order to do some analysis. This method is rather rudimentary and is a blocking method with no feedback about captured packets until the capture is stopped.

2.2 Data Organization

As mentioned in the introduction, the SIP protocol is modeled after the HTTP protocol. Almost universally, the UDP protocol is chosen for delivery over TCP due to its light overhead and the nature of voice communications. This means that for any given phone call, you will have several separate UDP packets being sent back and forth, but they all belong to the same phone call.

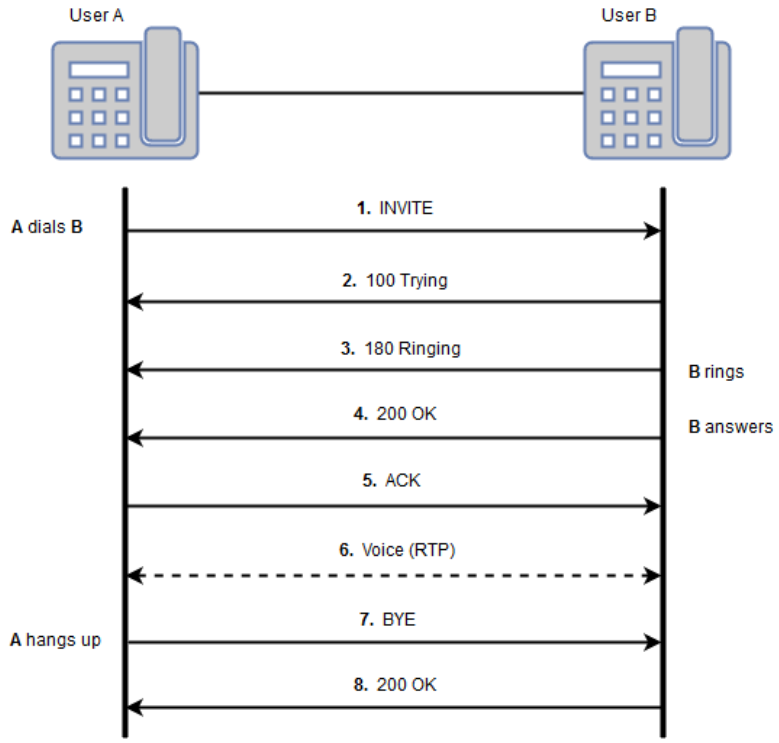

Figure 1 - Basic Call Flow

Often when capturing SIP data for analysis, one will have some data that is not pertinent to the debugging process, such as other phone calls or SIP events that were going on at the same time. You might also have, depending on the capture method, a lot of RTP media, and even other unrelated packets, which makes analysis even more cumbersome unless you have some way of filtering out the non-pertinent data. Sometimes, analyzing the RTP media along with the SIP messages can be key to solving a problem, and if this is the case, another tool, such as Wireshark, would need to be used.

When SIPvestigator parses and processes the SIP data, regardless of the source of the data, it only cares about SIP messages, and will discard all other information. The SIP messages are grouped together into dialogs by a message header present in all messages called Call-ID, which is a (partially) randomly generated unique identifier generated by the client sending the initial request (see Figure 2). SIPvestigator refers these dialogs (or sessions) as *conversations*, though the former two are more commonly used in standard practice.

```
INVITE sip:b@10.20.30.40 SIP/2.0
Via: SIP/2.0/UDP 10.11.12.13:5060;branch=Zhfm23KipPqX
Max-Forwards: 70
To: User B <sip:b@10.20.30.40>
From: User A <sip:a@10.11.12.13>;tag=7290019273
Call-ID: 4ab0020c58f474810282c8ab410862c2b@10.11.12.13:5060
CSeq: 1 INVITE
Contact: <sip:a@10.11.12.13:5060>
Content-Type: application/sdp
Content-Length: 152

[SDP OMITTED]
```

Figure 2 - Example SIP Invite

Since SIP messages can appear out of order (due to the nature of UDP), and multiple conversations are taking place at once, SIPvestigator will blindly go over the whole capture file, and just add each SIP message to its corresponding conversation, based on the Call-ID. The conversations themselves are then stored in the order in which they arrived (or rather the order of the first message seen of each conversation), and can be accessed numerically via the order of arrival or via the unique Call-ID.

2.3 CLI Interface

The SIPvestigator utility was designed to be used with a command-line interface, or CLI, akin to those used in commercial networking equipment, such as Juniper or Cisco. It is somewhat tree structured, in that each command has a set of subcommands which may have further subcommands. It has built-in help and tab-completion.

There are effectively two modes of operation, or contexts; general operation and filter configuration.

In general operation, one can load data from one or more files, clear data, start a packet capture, get help, show data relating to SIP conversations that have been loaded or captured, exit the program or enter the other mode of operation; filter configuration.

One can see a help menu by typing out the command *'help'*.

```
SipInv: help

Available commands (type help <topic> for help on topic):

capture
clear                   Clear the SIP conversation data
exit                    Exit the program
filter                  Enter SIP filter configuration for filtering conversations
help                    List available commands with 'help' or detailed help with 'help <topic>'.
load                    Load SIP data from file.
show                    Show information about configuration and data
```

Figure 3 - Help Menu

As shown in Figure 3, detailed help on a topic can be seen by issuing the command *'help <topic>'*. For example, in Figure 4, one can see the help menu for the *show* command.

```
SipInv: help show
Show configuration or data. Type help <topic> for further information.
Valid parameters are:

conversation       <call-id|index> [message <index>|messages]     Shows conversation details
conversations      [sorted]                                       Shows list of conversations, optionally sorted by order received
filter                                                            Shows the active conversation filter
```

Figure 4 - Help Menu for Show Command

2.3.1 Loading Data

Data can either be loaded by supplying a filename when starting the program, or via the use of the command *load* followed by a path to a file. Multiple files can be loaded in the same session, and this will simply add to the number of messages/conversations available.

```
$ python sipvestigator.py
SipInv: load txts/testdata-pt.txt
Processing file txts/testdata-pt.txt...
1083 conversations (6496 SIP messages) extracted
```

Figure 5 - Loading Data

2.3.2 Capturing Data

Data can also be captured directly from within SIPvestigator. This is still rather rudimentary, with a capture blocking further user input and analysis until the capture is complete. The capture can be stopped by using the *break* key sequence, and at the end of the capture, the number of captured packets will be shown on screen.

A capture can be started by issuing the command *'capture start'*. The capture is stopped by issuing the *break* (Control-C) key sequence.

```
SipInv: capture start
Starting packet capture
^C
6 conversations (20 SIP messages) extracted
```

Figure 6 - Capturing Data

2.3.3 Creating Filters

In the filter configuration mode of operation, or context, one can add and delete filters, show the active filters, select between filtering modes, reset (remove) all filters and exit the filter mode. Once a filter has been added, an asterisk (*) will appear behind the SipInv prompt to indicate that a filter is currently active.

To create filters, the *filter* configuration mode must be entered by issuing the command *'filter'*.

There are two filtering modes; intersection-based filtering and union-based filtering. This is colloquially better known as *and/or* filtering. The filtering mode is selected by setting the *condition* filter to either *and* or *or*. If it helps, one could think of the *and* filtering mode as a mode that narrows search results down to fewer and fewer results, while the *or* mode will expand the search to include more and more results.

```
SipInv: filter
SipInv[filter]: set condition and
1083 conversations
SipInv[filter]*: add srcnum 4151509
8 conversations
SipInv[filter]*: add dstnum 8969669
2 conversations
SipInv[filter]*: exit
SipInv*:
```

Figure 7 - Filtering Data Using 'and' Condition

The filtering mode can be set to intersection filter by issuing the command *'set condition and'* from the filter context. Similarly, it can be set to set to a union filter by issuing the command *'set condition or'*. The default value for condition is *or*. At present, it is not possible to mix and match and/or conditions.

```
SipInv: filter
SipInv[filter]: set condition or
0 conversations
SipInv[filter]*: add srcnum 8969669
2 conversations
SipInv[filter]*: add dstnum 8969669
4 conversations
SipInv[filter]*: add has diversion
59 conversations
SipInv[filter]*: exit
SipInv*:
```

Figure 8 - Filtering Data Using 'or' Condition

The filter context has its own help menu, which can be invoked by issuing the command *'help'*.

```
SipInv[filter]: help

Available commands (type help <topic> for help on topic):

add                     Add a filter condition
del                     Remove a filter condition
exit                    Exit filter configuration
help                    List available commands with 'help' or detailed help with 'help <topic>'.
reset                   Remove all filter conditions
set                     Set a filter condition
show
```

Figure 9 - Filter Help Menu

Each command within that help menu also has its own help submenu, which can be invoked by issuing the command *'help <topic>'*, which will generally have some examples of output as well.

```
SipInv[filter]: help add

Add a filter to select a subset of conversations matching filter. Valid arguments are:

                                                    Select conversations where...
diversion          <telnum> [telnum(s)]             <telnum> is in one of the Diversion headers
dstip              <ip> [ip(s)]                     <ip> is the destination address
dstnum             <telnum> [telnum(s)]             <telnum> is the destination number
has                <header>                         <header> is present. Use 'help add has' for details
ip                 <ip> [ip(s)]                     <ip> is either the source or destination ip address
pai                <telnum>                         <pai> is in one of the P-Asserted-Identity headers
ppi                <telnum>                         <ppi> is in one of the P-Preferred-Identity headers
srcip              <ip> [ip(s)]                     <ip> is the destionation address
srcnum             <telnum> [telnum(s)]             <telnum> is the source number
telnum             <telnum> [telnum(s)]             <telnum> is either the source or destionation number

Examples:
                   SipInv: add srcnum 4151500
                   5 conversations

                   SipInv: add srcnum 4151500 4151502
                   7 conversations

                   SipInv: add has diversion
                   87 conversations
```

Figure 10 - Filter Help Menu for Add Command

2.3.4 Showing Conversations

Once the data has been filtered, a list of the matching conversations with a summary of the first SIP method seen and between what parties can be shown. This is done by issuing the command *'show conversations'*. They can be displayed either in order of arrival by adding the keyword *sorted*, i.e. *'show conversations sorted'*, or ordered alphabetically by their Call-IDs by omitting the keyword.

```
SipInv*: show conversations sorted
[0]   15:49:30.333525: INVITE 4151509->8969669
[1]   15:49:30.377296: INVITE 4151509->8969669
SipInv*:
```

Figure 11 - Showing Matching Conversations in Order of Arrival

```
SipInv*: show conversations
4a40a41a3636462357f6f6f25b700844@10.20.0.22:5060: 15:49:30.333525: INVITE 4151509->8969669
91c06e8e-c572-1235-c0ba-00215e2c8c90: 15:49:30.377296: INVITE 4151509->8969669
SipInv*:
```

Figure 12 - Showing Matching Conversations in Alphabetical Order of Call-ID

In this instance, there seem to be two legs of the same conversation; i.e. the capture seems to have been made at a SIP proxy/B2BUA, and we are seeing both the inbound call-leg as received by the proxy and the outbound call-leg as sent-out by the proxy/B2BUA.

2.3.5 Showing Conversation Message Overview

From there, we can choose a conversation (e.g. conversation #0) and see a summary of all the SIP methods (or messages) exchanged in the conversation, with timestamps and information about which party sent it by issuing the command *'show conversation 0'*.

```
SipInv*: show conversation 0
===  Conversation Call-ID: 4a40a41a3636462357f6f6f25b700844@10.20.0.22:5060  ===
[0]  15:49:30.333525      INVITE                                        (4151509->8969669)
[1]  15:49:30.334001      100 Trying                                    (4151509->8969669)
[2]  15:49:30.852919      183 Session Progress                          (4151509->8969669)
[3]  15:49:44.236059      200 OK                                        (4151509->8969669)
[4]  15:49:44.236692      ACK                                           (4151509->8969669)
[5]  15:50:03.479602      BYE                                           (8969669->4151509)
[6]  15:50:03.480833      200 OK                                        (8969669->4151509)
SipInv*:
```

Figure 13 - Specific Conversation Overview

2.3.6 Showing Conversation Message Contents

At this point, we can either choose to see all the messages of the conversation sequentially, with the start of each message clearly marked out, or we can select which individual message to see.

To see the contents of all the messages, we simply request all the messages by issuing the command '*show conversation 0 messages'* for the above example. Note that Figure 14 has been cropped to limit size and does not include the entire conversation.

```
SipInv*: show conversation 0 messages
=======  Message #0  =======
15:49:30.333525 IP 10.20.0.22.5060 > 10.11.12.13.5060: SIP, length: 1041
INVITE sip:8969669@10.11.12.13:5060 SIP/2.0
Via: SIP/2.0/UDP 10.20.0.22:5060;branch=z9hG4bK375a8cbf
Max-Forwards: 70
From: <sip:4151509@10.20.0.22>;tag=as25c47886
To: <sip:8969669@10.11.12.13:5060>
Contact: <sip:4151509@10.20.0.22:5060>
Call-ID: 4a40a41a3636462357f6f6f25b700844@10.20.0.22:5060
CSeq: 102 INVITE
User-Agent: Asterisk PBX 13.1.0~dfsg-1.1ubuntu4.1
Date: Tue, 06 Jun 2017 15:49:30 GMT
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
Supported: replaces, timer
Content-Type: application/sdp
Content-Length: 459

v=0
o=root 1718872612 1718872612 IN IP4 10.20.0.22
s=Asterisk PBX 13.1.0~dfsg-1.1ubuntu4.1
c=IN IP4 10.20.0.22
b=CT:384
t=0 0
m=audio 57692 RTP/AVP 9 8 0 101
a=rtpmap:9 G722/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=maxptime:150
a=sendrecv
m=video 54246 RTP/AVP 31 34 98 99
a=rtpmap:31 H261/90000
a=rtpmap:34 H263/90000
a=rtpmap:98 h263-1998/90000
a=rtpmap:99 H264/90000
a=sendrecv

=======  Message #1  =======
15:49:30.334001 IP 10.11.12.13.5060 > 10.20.0.22.5060: SIP, length: 299
SIP/2.0 100 Trying
```

Figure 14 - Show Conversation Messages

Similarly, we can request a single message (e.g. #5) from the conversation by issuing the command *'show conversation 0 message 5'*

```
SipInv*: show conversation 0 message 5
15:50:03.479602 IP 10.11.12.13.5060 > 10.20.0.22.5060: SIP, length: 631
BYE sip:4151509@10.20.0.22:5060 SIP/2.0
Via: SIP/2.0/UDP 10.11.12.13;rport;branch=z9hG4bKg3QQ3gBBS61NF
Max-Forwards: 70
From: <sip:8969669@10.11.12.13:5060>;tag=aKa50Svj8SSmr
To: <sip:4151509@10.20.0.22>;tag=as25c47886
Call-ID: 4a40a41a3636462357f6f6f25b700844@10.20.0.22:5060
CSeq: 108046037 BYE
Contact: <sip:8969669@10.11.12.13:5060;transport=udp>
User-Agent: FreeSWITCH-mod_sofia/1.2.12
Allow: INVITE, ACK, BYE, CANCEL, OPTIONS, MESSAGE, INFO, UPDATE, REGISTER, REFER, NOTIFY, PUBLISH, SUBSCRIBE
Supported: timer, precondition, path, replaces
Reason: Q.850;cause=16;text="Normal call clearing"
Content-Length: 0
```

Figure 15 - Show Specific Conversation Message

2.4 Common Problems and Practical Examples

Now that we have gone through the features, it might be helpful to show some practical examples of how the tool is used to aid with debugging.

There are, in the author's experience, three problems that make up the majority of VoIP support requests from customers, in order of prevalence[5];

1. NAT or firewall issues leading to retransmitted SIP requests that get no response
2. Improper Caller-ID configuration during initial setup of SIP trunks resulting in a call-rejection, either for inbound or outbound calls
3. Missing or wrong *Diversion* headers during call forwards of SIP trunks resulting in a call-rejection

Less common problems include more subtle NAT problems that are a bit harder to diagnose, usually due to a faulty ALG[6] implementation, SIP requests coming in from the wrong source IP address or destination port, codec mismatches, and many more.

As previously mentioned; SIPvestigator's purpose is to aid debugging by making it easier to diagnose problems by reducing noise through simple and intuitive filters that make sense to the user in an IP telephony context, such as phone numbers, IP addresses, diversion headers, and so on. We will now look at each of the three problems with an example use case.

2.4.1 NAT Timeout or Missing Port-Forward

A common problem for SIP endpoints is that they are sitting behind a NAT or a firewall that does not pass received messages on to the endpoint. In these cases, a SIP message is sent to the address of the endpoint, which is known either via a SIP registration from the SIP endpoint to its SIP registration server (or registrar), or via a known, fixed IP address and port combination. The latter is often called a SIP trunk.

In the case of a SIP registration, the SIP endpoint will send a special SIP REGISTER message to its SIP registration server which will start an authentication process. If the SIP endpoint is sitting behind a NAT or a firewall, this process will leave open a port in the router/firewall's NAT table, through which the endpoint will be reachable. Usually, though, this NAT session in the NAT table will live for a relatively short time (anywhere from 30 seconds to a few minutes). Once the entry has expired, the SIP endpoint is no longer reachable through this IP/port combination. It will only become reachable again once it re-registers with the registrar or dispatches a SIP message. The interval between registrations is usually configurable with common values of 30-60 minutes.

---

[5] Naturally, the prevalence of the types of problems encountered depends on the types of services being sold and customer base of the service provider.

[6] An *Application Layer Gateway* is a router implementation that aims to be protocol/application aware and mitigate problems presented by NAT. Often, these implementations are poorly implemented and cause more harm than good. SIP gateways have gotten pretty good at detecting and mitigating NAT problems themselves.

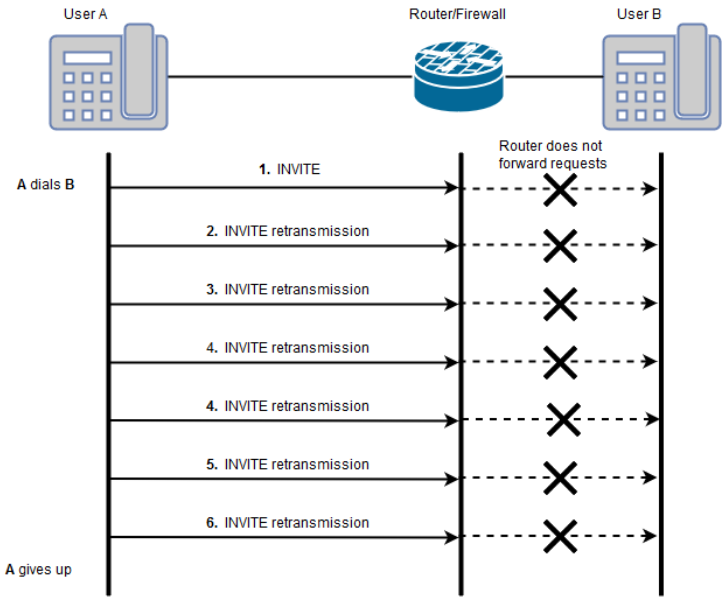

Figure 16 - NAT Timeout

A common solution to the problem is to lower the re-registration period to something like 60-120 seconds. This will force the SIP endpoint to frequently re-register, thus maintaining the router's/firewall's NAT table entry. Another solution is to leave the registration period as-is and send a so-called NAT keepalive message. This is generally a rather meaningless SIP message like OPTIONS, initially intended to probe a SIP endpoint for its capabilities. While the endpoint usually does not care about the response to the OPTIONS query in this context, the intent of keeping the NAT table entry alive has been fulfilled.

In the case of SIP trunks, if a SIP endpoint is sitting behind a NAT/firewall, there are two methods available to make it reachable. The first one is via the NAT keepalive SIP OPTIONS message mentioned in the previous paragraph, and the second one is via a NAT port-forwarding rule on the router/firewall. The latter is generally used for SIP trunks, should they reside behind a NAT, though it is generally preferred to eliminate NAT altogether with SIP trunks by using additional IP addresses or private networks.

2.4.2 NAT Timeout Example

The general problem with a NAT timeout is that outbound calls from the SIP endpoint are working, while inbound calls are (sometimes or always) not working. The best filter for finding such a problem is thus generally via the destination number.

We start off by loading SIPvestigator with the captured output and setting up a filter for the known destination number dialed.

```
$ python sipvestigator.py txts/timeout.txt
Processing file txts/timeout.txt...
45 conversations (354 SIP messages) extracted
SipInv: filter
SipInv[filter]: add dstnum 5660066
1 conversations
SipInv[filter]*: exit
SipInv*: show conversations sorted
[0]   16:41:37.979995: INVITE 4151502->5660066
SipInv*:
```

Figure 17 - Loading, Filtering and Glancing at NAT Timeout Data

This reduces the number of conversations from 45 to exactly 1 conversation, which has already reduced the scope of the problem greatly. We can optionally take a make sure this is the correct call by verifying the source number and timestamp by issuing *'show conversations sorted*.

This seems like the right call, so we can look at the message exchange by issuing *'show conversation 0'*.

```
SipInv*: show conversation 0
===  Conversation Call-ID: 35e857d642ef6280292e2d0e4cac4f13@10.10.10.2:5060  ===
[0]  16:41:37.979995      INVITE                                        (4151502->5660066)
[1]  16:41:37.979998      INVITE                                        (4151502->5660066)
[2]  16:41:38.480357      INVITE                                        (4151502->5660066)
[3]  16:41:38.480360      INVITE                                        (4151502->5660066)
[4]  16:41:39.479261      INVITE                                        (4151502->5660066)
[5]  16:41:39.479264      INVITE                                        (4151502->5660066)
[6]  16:41:41.479343      INVITE                                        (4151502->5660066)
[7]  16:41:41.479348      INVITE                                        (4151502->5660066)
[8]  16:41:45.480851      INVITE                                        (4151502->5660066)
[9]  16:41:45.480854      INVITE                                        (4151502->5660066)
[10] 16:41:53.480515      INVITE                                        (4151502->5660066)
[11] 16:41:53.480518      INVITE                                        (4151502->5660066)
[12] 16:42:09.479770      INVITE                                        (4151502->5660066)
[13] 16:42:09.479774      INVITE                                        (4151502->5660066)
SipInv*:
```

Figure 18 - NAT Timeout Conversation Overview

It seems rather obvious at this stage that the SIP endpoint is not responding to the SIP INVITE messages. There may be multiple reasons for this. Depending on the description of the problem by the customer, the reason can be better surmised. If no calls were working at all, inbound or outbound, the reason could be a networking problem for the SIP endpoint. In the case of working outbound calls but inbound calls receiving no reply, a NAT timeout should be suspected.

The next stage in debugging would be looking at a single INVITE message, to make sure that the request is being sent to the correct IP address/port. If not, there might be a faulty SIP ALG causing the registration entry (the IP address/port location of the SIP endpoint) to be incorrect. In that case, the SIP registrar can usually be configured to ignore the reported IP address of the endpoint and rather use the source IP address of the REGISTER request to mitigate the problem. Another possibility of a problem would be a misconfigured SIP trunk.

Let us look at one of these INVITE messages by issuing *'show conversation 0 message 0'*.

```
SipInv*: show conversation 0 message 0
16:41:37.979995 IP 10.10.10.2.5060 > 178.19.48.48.5060: SIP: INVITE sip:5660066@178.19.48.48 SIP/2.0
INVITE sip:5660066@178.19.48.48 SIP/2.0
Via: SIP/2.0/UDP 10.10.10.2:5060;branch=z9hG4bK72aeec24
Max-Forwards: 70
From: "4151502" <sip:4151502@10.10.10.2>;tag=as71102a1e
To: <sip:5660066@178.19.48.48>
Contact: <sip:4151502@10.10.10.2:5060>
Call-ID: 35e857d642ef6280292e2d0e4cac4f13@10.10.10.2:5060
CSeq: 102 INVITE
User-Agent: Asterisk PBX 13.1.0~dfsg-1.1ubuntu4.1
Date: Mon, 16 Apr 2018 16:41:37 GMT
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
Supported: replaces, timer
P-Asserted-Identity: "4151502" <sip:4151502@10.10.10.2>
Content-Type: application/sdp
Content-Length: 457

v=0
o=root 453687495 453687495 IN IP4 10.10.10.2
s=Asterisk PBX 13.1.0~dfsg-1.1ubuntu4.1
c=IN IP4 10.10.10.2
b=CT:384
t=0 0
m=audio 54028 RTP/AVP 9 8 0 101
a=rtpmap:9 G722/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=maxptime:150
a=sendrecv
m=video 53738 RTP/AVP 31 34 99 98
a=rtpmap:31 H261/90000
a=rtpmap:34 H263/90000
a=rtpmap:99 H264/90000
a=rtpmap:98 h263-1998/90000
a=sendrecv
```

Figure 19 - NAT Timeout SIP Invite

If we are happy that the source and destination IP addresses and ports are correct, then judging from this data, a reasonable assumption would be that a router/firewall configuration or a NAT timeout is at fault, and we can adjust configuration on both ends accordingly to try to mitigate the problem.

2.4.3 Improper Caller-ID Configuration

Another common problem is improper Caller-ID configuration.

SIP trunks are often thought of as virtual circuits, like the ones traditionally used in telecommunications before the prevalence of VoIP. They “connect” two points, which in the case of VoIP is usually just a set of IP addresses and ports. In the case of a traditional PSTN trunk, this used to be a physical copper or optical link. There are optional restrictions on the number of concurrent calls “on the trunk”, either due to limited bandwidth or licensing reasons.

Another common property of trunks, SIP or otherwise, is a restriction on which Caller-ID can be used when making outbound calls. This is especially the case for trunks configured for PBXs that companies might run for their offices, and less so for trunk interconnects between telecommunication companies.

The reason for this property or limitation is twofold; to prevent a malicious user from using the trunk to make fraudulent calls or using it for otherwise nefarious or deceptive purposes, and to make sure that the SIP trunks / PBXs are properly configured and sending the correct Caller-ID.

We shall now look at an example of a failure of the latter.

2.4.4 Improper Caller-ID Configuration Example

The best way to diagnose this problem is to ask the end user to make a phone call to a pre-determined telephone number, so we can easily filter the data.

Another common way of filtering would be to filter calls by the source IP address, as in such a case, the traffic originating from the IP address would generally be minimal.

Let us load the data and filter by the destination number.

```
$ python sipvestigator.py txts/callerid-problem.txt
Processing file txts/callerid-problem.txt...
406 conversations (2090 SIP messages) extracted
SipInv: filter
SipInv[filter]: add dstnum 5660066
1 conversations
SipInv[filter]*: exit
SipInv*:
```

Figure 20 - Loading and Filtering Improper Caller-ID Data

As before, our large dataset of 406 conversations / 2090 SIP messages gets filtered down to just 1 conversation, making analysis that much easier.

We can now look at an overview of the one conversation by issuing *'show conversations sorted'*.

```
SipInv*: show conversations sorted
[0]   15:54:44.871687: INVITE 1502->5660066
SipInv*:
```

Figure 21 - Improper Caller-ID Conversation Overview

From this data, it is already evident that the problem is most likely that the Caller-ID is wrong, as 1502 is not a valid Caller-ID in Iceland according to the Icelandic numbering plan[7] and the capture was made outside of the PBX. As before, we can see more detail by issuing the command *'show conversation 0'*.

[7] https://www.pfs.is/english/telecom-affairs/numbering/

```
SipInv*: show conversation 0
===  Conversation Call-ID: 4bd2133a5be9a247447c5d3e44dd58cd@10.20.0.22:5060  ===
[0]  15:54:44.871687      INVITE                                        (1502->5660066)
[1]  15:54:44.871969      100 Trying                                    (1502->5660066)
[2]  15:54:44.900286      403 Forbidden                                 (1502->5660066)
[3]  15:54:44.900831      ACK                                           (1502->5660066)
SipInv*:
```

Figure 22 - Improper Caller-ID Conversation Details

Now it should be abundantly clear to anyone with a knowledge in SIP communications and the Icelandic numbering plan that the call is being rejected due to an improper Caller-ID, assuming the SIP trunk has such restrictions.

Finally, we look at the actual INVITE to see whether there is something else that is improper by issuing the command *'show conversation 0 message 0'*.

```
SipInv*: show conversation 0 message 0
15:54:44.871687 IP 10.20.0.22.5060 > 10.11.12.13.5060: SIP, length: 1040
INVITE sip:5660066@10.11.12.13:5060 SIP/2.0
Via: SIP/2.0/UDP 10.20.0.22:5060;branch=z9hG4bK193c86a2
Max-Forwards: 70
From: "..rn" <sip:1502@10.20.0.22>;tag=as0a5e0c70
To: <sip:5660066@10.11.12.13:5060>
Contact: <sip:1502@10.20.0.22:5060>
Call-ID: 4bd2133a5be9a247447c5d3e44dd58cd@10.20.0.22:5060
CSeq: 102 INVITE
User-Agent: Asterisk PBX 13.1.0~dfsg-1.1ubuntu4.1
Date: Mon, 16 Apr 2018 15:54:44 GMT
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
Supported: replaces, timer
Content-Type: application/sdp
Content-Length: 457
```

Figure 23 - Improper Caller-ID INVITE message (message clipped before SDP)

This gives us no new information. Everything looks proper apart from the Caller-ID number of 1502.

2.4.5 Missing or Wrong Diversion Headers

When an incoming call is diverted (or call-forwarded) to another number in SIP, there are multiple methods of doing so. The most commonly used in Iceland, due to its simplicity, is a SIP header called Diversion.

When diverting a call, the user receiving the diverted call will generally want to know who is calling him (or calling the number being diverted). The user receiving the diverted call usually does not care what number diverted the call, as it is usually the user himself who has configured the diversion.

The reason a header such as this is needed is to preserve the information about who is diverting the call, so that the call can be authorized and billed correctly, as the user who originally dialed the number probably does not have permission to make outbound calls on the trunk.

To give an example, **User A** calls **User B**. **User B** has set up a call-forward/diversion to his mobile phone number, which we will call **User C**. The Caller-ID of **User A** has been preserved, so to the SIP trunk and **User C**, it looks like **User A** is calling **User C**, which is not the case. Therefore, a Diversion header stating that **User B** has diverted the call is inserted into the INVITE message.

We shall now look at an example where a Diversion header is incorrect.

2.4.6 Wrong Diversion Header Example

As before, one of the best filters to use is simply the destination number. Generally, if someone is complaining about a diversion / call forward not working, he will know to which number the call is being forwarded, though of course in some cases the wrong destination number might of course be part of the problem.

As before, we start by loading the data and setting the filter.

```
$ python sipvestigator.py
SipInv: load txts/diversion-problem.txt
Processing file txts/diversion-problem.txt...
494 conversations (2495 SIP messages) extracted
SipInv: filter
SipInv[filter]: add dstnum 5660066
1 conversations
SipInv[filter]*: exit
SipInv*:
```

Figure 24 - Loading and Filtering Wrong Diversion Header

Alternatively, we could have used another filter, either standalone or in conjunction with the one we used, called *has diversion*, which states that we want to see calls that actually have a Diversion header present. Had we used the *has diversion* filter with an end-result of no calls found, that would have given us an indication that the header was actually missing.

```
SipInv[filter]: add has diversion
16 conversations
SipInv[filter]*: add dstnum 5660066
16 conversations
SipInv[filter]*: set condition and
1 conversations
SipInv[filter]*:
```

Figure 25 - Filtering Wrong Diversion Header with Has Diversion and Dstnum

In this instance, the result from using this filter was the same, since we only had one call to start with after filtering by the destination number. It was merely used for demonstration purposes.

As we only have one conversation, we can skip looking over the list of conversations as that would likely not give us any useful information in this case. Therefore, we just issue a command to see the conversation overview, using tab-completion to fill-in the Call-ID of the call (though we could have just as easily entered the number 0 as there is only one conversation).

```
SipInv*: show conversation 256c2b012c04d48e5bbbad970bca3268@10.20.0.22:5060
[0]  16:11:51.307696      INVITE                                        (7712552->5660066)
[1]  16:11:51.307948      100 Trying                                    (7712552->5660066)
[2]  16:11:51.337896      403 Forbidden                                 (7712552->5660066)
[3]  16:11:51.338437      ACK                                           (7712552->5660066)
SipInv*:
```

Figure 26 - Wrong Diversion Header Conversation Details

We can see that the Caller-ID in this instance is 7712552, which according to the Icelandic Numbering Plan is a mobile telephone number. As this is a call originating from a SIP trunk, that indicates that the call is being diverted[8], so let us look at the message and look for the header.

```
SipInv*: show conversation 256c2b012c04d48e5bbbad970bca3268@10.20.0.22:5060 message 0
16:11:51.307696 IP 10.20.0.22.5060 > 10.11.12.13.5060: SIP, length: 1085
INVITE sip:5660066@10.11.12.13:5060 SIP/2.0
Via: SIP/2.0/UDP 10.20.0.22:5060;branch=z9hG4bK63f115b9
Max-Forwards: 70
From: "7712552" <sip:7712552@10.20.0.22>;tag=as44e3a506
To: <sip:5660066@10.11.12.13:5060>
Contact: <sip:7712552@10.20.0.22:5060>
Call-ID: 256c2b012c04d48e5bbbad970bca3268@10.20.0.22:5060
CSeq: 102 INVITE
User-Agent: Asterisk PBX 13.1.0~dfsg-1.1ubuntu4.1
Date: Mon, 16 Apr 2018 16:11:51 GMT
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO, PUBLISH, MESSAGE
Supported: replaces, timer
Diversion: <sip:1502@178.19.49.10>
Content-Type: application/sdp
Content-Length: 457
```

Figure 27 - Wrong Diversion Header INVITE message

As we can see, there is indeed a Diversion header present. However, its numeric value of 1502 is not a legal number in the Icelandic numbering plan. Further, the IP address in the Diversion header does not match the source IP address of the capture, which might indicate that the diversion is happening elsewhere. This is not necessarily an error but might rather indicate that the diversion took place someplace else than on the PBX itself. It depends on the setup of the PBX and its clients.

Correcting the Diversion header to include the full telephone number of the party diverting the call would resolve the issue.

[8] Alternatively, someone might be trying to use a fake Caller-ID. Since we used a filter that used an intersect of the destination number 5660066 and the condition *has diversion*, we know that a Diversion header must be present.

3 Conclusion

VoIP using SIP is a technology that is seeing more and more use due to the rapid decline of fixed-line alongside the rapid adoption of VoLTE. There are rather few tools available to aid analysis outside of the telecommunications sphere where specialized tools intended for use with the vendor products are provided.

In the author's experience, once a SIP/VoIP platform has been installed and setup, most debugging requests are for something rather simple, such as a misconfigured Caller-ID or invalid number type for the destination number. Debugging such simple problems can take up a disproportionate amount of time considering the simplicity of the problem. This is due to too much information on the SIP gateways where the debugging is generally taking place. Finding the correct SIP messages to look at is time consuming for the user due to this vast amount of data. The result is often that the debug is dumped to a PCAP file, transferred to the user's personal computer and opened in a tool like Wireshark.

With SIPvestigator, the user can quickly and reliably find the problematic SIP messages due to the easy-to-use filtering. As this can be performed directly on the server where the capture is taking place, analysis can start immediately. Should further analysis be needed, such as analysis of audio problems, the PCAP file can be transferred onto the user's computer for deeper analysis.

This tool is built for users with experience in reading and debugging SIP messages. It has no automatic debugging features, such as detecting a codec mismatch between parties, but rather just makes all the relevant information available to the user so he can start debugging quickly without having to wade through an ocean of irrelevant information.

Future work includes better support for taking live captures with progress updates in real-time, exporting captured and filtered data, better fault tolerance for malformed SIP messages, more contexts so that a user could e.g. enter a '*conversation 0'* context so he need only enter *'show message 0'* to see the message etc. Even without this further work, SIPvestigator could prove very useful in the quick analysis of problems.

References

[1] Post and Telecom Administration in Iceland, May 2011, “Statistics on the Icelandic Telecommunications Market 2010”, https://www.pfs.is/upload/files/Tölfræðiskýrsla_PFS_2010.pdf

[2] Post and Telecom Administration in Iceland, May 2017, “Statistics on the Icelandic Telecommunications Market 2016”, https://www.pfs.is/library/Skrar/Tolfraedi/Tolfraediskyrslur-PFS/Tolfraedi_um_islenska_fjarskiptamarkadinn_2016.pdf

[3] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, June 2002, RFC 3261, “SIP: Session Initiation Protocol”, https://www.rfc-editor.org/info/rfc3261

[4] Schulzrinne, H., Casner, S., Frederick, R., and V. Jacobson, July 2003, RFC 3550, “RTP: A Transport Protocol for Real-Time Applications”, https://www.rfc-editor.org/info/rfc3550

[5] Song, D., April 2018, “dpkt” (Version 1.9.1) [Software]. Available from https://pypi.python.org/pypi/dpkt

[6] P. Biondi, April 2018, “Scapy” (Version 2.2.0) [Software]. Available from http://www.secdev.org/projects/scapy/