

Nethemba s.r.o.

# ASTERISK

## Security Hardening Guide v1.0

<b>Author:</b>	Boris Písarčík
<b>Security consultants:</b>	Pavol Lupták
<b>Sponsored by:</b>	<a href="#">T-Mobile Czech Republic a.s.</a>
<b>Creation date:</b>	10.10.2011
<b>Version:</b>	1.0
<b>Last change:</b>	15.01.2012

## Contents

1 Foreword.....	3
2 Securing Asterisk.....	4
2.1 Encrypt signaling – use TLS as SIP transport.....	4
2.2 Encrypt voice – deploy SRTP.....	8
2.3 Restrict SIP clients on IP addresses.....	11
2.4 Protect your asterisk box with firewall.....	12
2.5 Do not let bad guys enter in – deploy IPS in front of your Asterisk.....	13
2.6 Monitor critical files for changes.....	19
2.7 Hide your identity .....	20
2.8 Harden valid SIP extension discovery .....	22
2.9 Use strong passwords, really .....	23
2.10 Know how to failover .....	23
2.11 Protect your dialplans.....	25
2.12 Log to remote servers.....	27
2.13 Put asterisk on diet – disable unnecessary functionality.....	29
2.14 Other common security related advices.....	33
3 References.....	35

# 1 Foreword

Asterisk is a great piece of software and with proper use and configuration, its versatility and customizability make it one of best options for creating good stable long serving VoIP PBX guaranteeing customer satisfaction. However, as any other software on this planet, it can fail and it does fail, it has security issues, with security vulnerabilities being part of its several years long history and it can also be configured insecurely by us, humans.

In the following few chapters we will try to explain several options and ways how to make it stronger, safer, stealthier and durable SIP PBX.

We hope that after implementing at least some of our advices about Asterisk, where couple of them can be done almost instantly and with really little effort, we can help you make your PBX servers and your VoIP clients safer.

If you have any questions, opinions, suggestions or would like more information about the subject, please contact at our e-mail [info@nethemba.com](mailto:info@nethemba.com).

## 2 Securing Asterisk

### 2.1 Encrypt signaling – use TLS as SIP transport

The first half of cryptographic protection of every SIP call consists of protecting the signaling part of the protocol pair used to make SIP calls – SIP protocol itself. Since SIP protocol is in fact text based protocol similar to HTTP protocol, it provides itself no protection to information gathering or data manipulation. By standard, SIP is transferred over 5060 port over TCP or UDP protocol. SIP protocol itself provides no means of encryption. To address lack of security of core SIP protocol, SIPS or SIP/TLS protocol wraps up the unencrypted SIP channel within SSL or TLS encryption and uses TCP port 5061 by default.

Support for SIP TLS encryption comes with asterisk since version 1.6. There are few basic steps that need to be done in order to get it working:

1. Obtain or generate SSL private key with signed certificate and corresponding signing certificate authority's root certificate or certificate chain – these MUST be in PEM format
2. Set a few SSL/TLS related options in SIP configuration file sip.conf
3. Configure client to use SIP over SSL/TLS transport

Once you have all the encryption keys and certificates, place them in asterisk keys subdirectory, by default `/var/lib/asterisk/keys`. The contents of this directory may look like following after storing keys in place:

```
ca-bundle.pem
```

```
voip.somedomain.com.crt
```

```
voip.somedomain.com.key
```

To enable SSL/TLS protocol support, set the following options in sip.conf configuration file:

```
tlsenable=yes
```

**`tlscbindaddr=0.0.0.0`** - or whatever IP address we would like bind TCP socket and accept connections (0.0.0.0 means listen on all system network interfaces)

**`tlscacfile=/var/lib/asterisk/keys/ca-bundle.pem`**

or if you need to support more certification authorities, use directory version of the previous option (or you can use both of them, when certificates are looked up first against **`tlscacfile`** file):

**`tlscacpath=/var/lib/asterisk/keys/ca/`**

However, there are few requirements in order to support directory based CA certificate loading, every file must contain one and only CA certificate and it must be named according to the CA subject name hash value, which can be extracted from the CA certificate using openssl command line tool, like in the following example:

```
#openssl x509 -in voip.somedomain.com.crt -hash
```

```
8d694709
```

```
-----BEGIN CERTIFICATE-----
```

```
MIIIJTCCBw2gAwIBAgIDBIwwMA0GCSqGSIb3DQEBBQUAMIGMMQswCQYDVQQGEwJJ  
TDEWMBQGA1UEChMNU3RhcnRDb20gTHRkLjErMCKGA1UECxMiU2VjdXJlIERpZ210  
YWwgQ2VydG1maWNhdGUgU2lnbmluZzE4MDYGA1UEAxMvU3RhcnRDb20gQ2xhc3Mg
```

```
...
```

In this case, the name of the file would be “8d694709”. In case more CA certificates end up with equal hash, they must be appended with numerical extension, like 8d694709.0 , 8d694709.1, 8d694709.2 and so on.

**`tlscertfile=/var/lib/asterisk/keys/voip.somedomain.com.crt`** – this option specifies a certificate file signed by CA for this asterisk server

**`tlprivatekey=/asterisk/var/lib/asterisk/keys/voip.somedomain.com.ke  
y`** – and this option specifies a private key for this server.

If the “`tlprivatekey`” option is not specified, “`tlscertfile`” is used to look up both private key and certificate merged in a single file.

**`tlscclientmethod=tlsv1`** – this option, an addon for asterisk 1.8, configures protocol

version used for outgoing SIP connections, and can be set to one of **tlsv1**, **sslv3** or **sslv2** values. We strongly suggest using **tlsv1** and discourage use of **sslv2** due to the known security issues of this obsolete version of SSL protocol.

**tlscipher**="particularciphers" - string with ciphers enabled for connections can be specified as a string list of supported cipher names, examples are "DES-CBC3-SHA" or "ALL".

**tlsdontverifyserver**=[yes|no] – specifies if asterisk should verify other servers' certificate when acting as a SIP client. By default, it is set to no.

Summarized all in one piece of text, the whole SIP/TLS configuration would look like this:

```
tlsenable=yes
tlsbindaddr=0.0.0.0
tlscacfile=/var/lib/asterisk/keys/ca-bundle.pem
tlscertfile=/var/lib/asterisk/keys/voip.somedomain.com.crt
tlsprivatekey=/asterisk/var/lib/asterisk/keys/voip.somedomain.com.ke
y
tlsclientmethod=tlsv1
tlscipher=ALL
tlsdontverifyserver=no
```

After successful configuration of SIP/TLS support and restarting asterisk, or reloading SIP configuration, asterisk should listen on new TCP port dedicated for TLS connections, 5061 by default, as shown by using netstat command:

*Active Internet connections (only servers)*

```
Proto Recv-Q Send-Q Local Address           Foreign Address
State      PID/Program name
```

```
tcp          0          0 0.0.0.0:5060          0.0.0.0:*
LISTEN      888/asterisk

tcp          0          0 0.0.0.0:5061          0.0.0.0:*          LISTEN
888/asterisk

udp          0          0 0.0.0.0:5060          0.0.0.0:*
888/asterisk
```

To verify manually if the correct certificate is used for TLS connections, openssl command may be used too:

```
#openssl s_client -host localhost -port 5061
CONNECTED(00000003)
depth=2 /C=IL/O=StartCom Ltd./OU=Secure Digital Certificate
Signing/CN=StartCom Certification Authority
verify error:num=19:self signed certificate in certificate chain
verify return:0
---
Certificate chain
 0 s:/description=547373-
V08STSphH1Dv407L/CN=voip.somedomain.com/emailAddress=postmaster@some
domain.com
   i:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate
Signing/CN=StartCom Class 1 Primary Intermediate Server CA
 1 s:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate
Signing/CN=StartCom Class 1 Primary Intermediate Server CA
   i:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate
Signing/CN=StartCom Certification Authority
 2 s:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate
Signing/CN=StartCom Certification Authority
   i:/C=IL/O=StartCom Ltd./OU=Secure Digital Certificate
Signing/CN=StartCom Certification Authority
---
Server certificate
```

...

After global server configuration for SIP/TLS, individual accounts need to be modified in per-peer sections to define the account will be using TLS as SIP transport using the “transport” option in sip.conf, like in the following example:

```
[voipuser12]
type=friend
context=some_context
host=dynamic
secret=thisisntreallysecretpwd
nat=yes
transport=tls
```

**IMPORTANT:** SIP, however, is only a signaling protocol whose responsibility is only to maintain call state and call related actions, not the voice data itself, so it does not protect voice part of communication between SIP clients.

## 2.2 Encrypt voice – deploy SRTP

In order to prevent eavesdropping of the voice call data made by SIP enabled technology, cryptography plays a primary role of defense. SRTP protocol (or rather a special profile of RTP protocol) is the answer to the encryption of voice data and protection against threats to voice confidentiality and integrity.

Since asterisk version 1.8, SRTP is supported natively. For asterisk to support SRTP voice encryption, libsrtp library with development headers has to be installed. For Debian and Ubuntu Linux, it can be installed with the following command:

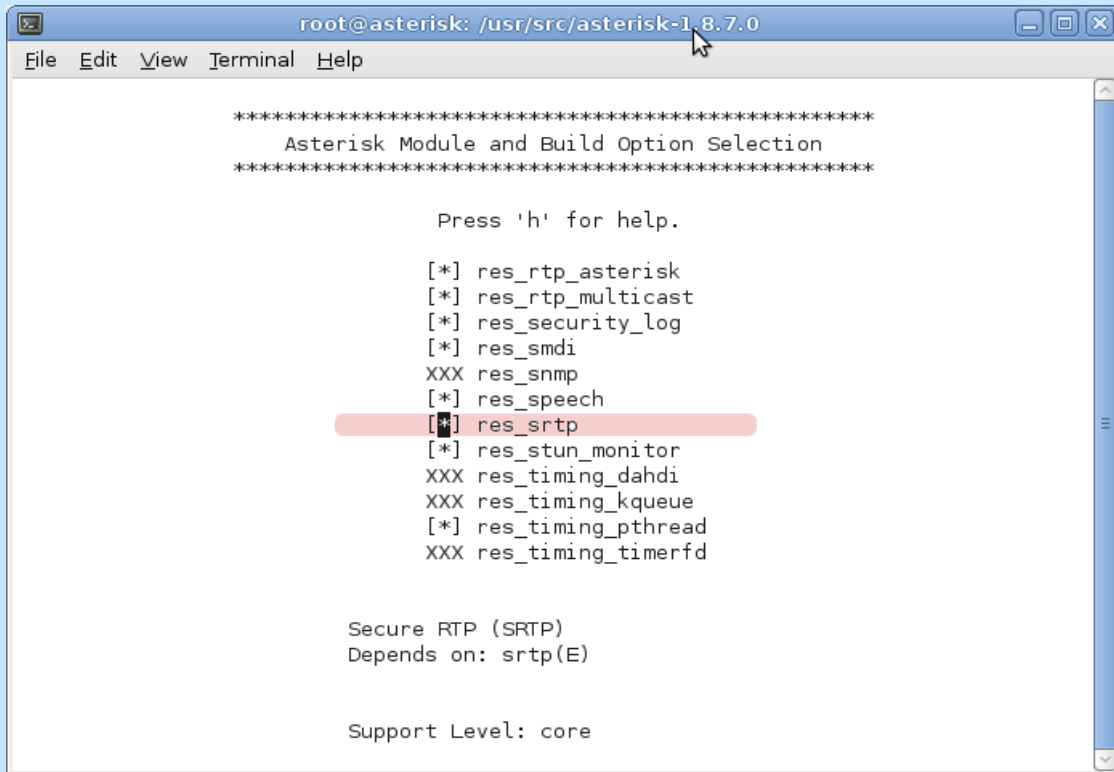
```
#aptitude install libsrtp0 libsrtp0-dev
```

For other distributions or operating systems, the source code may be necessary to download and compile, it can be downloaded [here](#).

In order to compile the SRTP support, the corresponding option has to be enabled



before asterisk compilation in “Resource Modules” top level menu of “make menuselect”.



```
root@asterisk: /usr/src/asterisk-1.8.7.0
File Edit View Terminal Help

*****
Asterisk Module and Build Option Selection
*****

Press 'h' for help.

[*] res_rtp_asterisk
[*] res_rtp_multicast
[*] res_security_log
[*] res_smdi
XXX res_snmp
[*] res_speech
[*] res_srtp
[*] res_stun_monitor
XXX res_timing_dahdi
XXX res_timing_kqueue
[*] res_timing_pthread
XXX res_timing_timerfd

Secure RTP (SRTP)
Depends on: srtp(E)

Support Level: core
```

After SRTP support is finally compiled in asterisk, the whole configuration itself is in the SIP configuration – there's only one option related to SRTP configuration, the “**encryption**” option.

Setting this option to “yes”, either in the global section or per-peer section, enforces asterisk to offer SRTP only encrypted media on outgoing calls to this peer. If the peer does not support SRTP, the call will fail with result HANGUPCAUSE = 58.

The sample peer section with enforced SRTP :

```
[voipuser12]
type=friend
context=some_context
host=dynamic
```

```
secret=thisisntreallysecretpwd
```

```
nat=yes
```

```
transport=tls
```

```
encryption=yes
```

**WARNING !** Keys used for SRTP encryption are exchanged in cleartext form and they can be easily captured by adversary. In the future, ZRTP key exchange management may solve this problem. Thus if the encryption of voice data should make sense, SIP signaling must use TLS transport protocol (also marked bold in previous sample).

**IMPORTANT:** during our practical deployment of SRTP we have experienced frequent crashing of asterisk (segmentation faults). Even the latest version of libsrtp (1.4.4) contains bug related to rtp protocol handling during SRTP call. In this case it's necessary to download libsrtp from CVS repository:

```
cvscvs -d:pserver:anonymous@srtp.cvs.sourceforge.net:/cvsroot/srtp  
login
```

```
cvscvs -z3 -d:pserver:anonymous@srtp.cvs.sourceforge.net:/cvsroot/srtp  
co -P srtp
```

The difference according to the stable version of libsrtp contains single line only:

```
http://srtp.cvs.sourceforge.net/viewvc/srtp/srtp/crypto/replay/rdb.c  
?r1=1.4&r2=1.5
```

## 2.3 Restrict SIP clients on IP addresses

In asterisk it is possible to permit or deny access to a peer or user based on their IP address, so that a particular peer can connect and register only from the predefined IP address(es). Access can also be explicitly denied for some IP addresses or address ranges in case the IP is suspected of some abusing activity.

Two configuration directives define filtering of client connections based on their IP address:

- deny
- permit

Format of these options is following:

```
deny=ip/mask
```

```
permit=ip/mask
```

Both directives can be specified multiple times for each peer definition in sip.conf and combining of these directives is allowed for the required result. The order of directives is important and EVERY deny or permit clause is consulted in the specified order, where the last matching block makes final decision.

**If no deny or permit block matches, access is permitted by default !**

Here are some permit/deny examples with the explanation :

1. Permit access only from IP address 1.2.3.4:

```
deny=0.0.0.0/0.0.0.0
```

```
permit=1.2.3.4/255.255.255.255
```

2. Deny connections from specific network subnet 10.0.1.0/24:

```
deny=10.0.1.0/255.255.255.0
```

3. Permit access from single subnet, but exclude 2 untrusted hosts:

```
deny=0.0.0.0/0.0.0.0
```

```
permit=192.168.0.0/255.255.255.0
```

```
deny=192.168.1.100/255.255.255.255
```

```
deny=192.168.1.100/255.255.255.255
```

## 2.4 Protect your asterisk box with firewall

The worst possible situation of protecting machine running production asterisk PBX would be the case when attacker can exploit some forgotten, badly configured or unprotected service running on the same box.

Hence it is crucial to allow only selected ports / services to be reachable from the outside or untrusted network.

If iptables is used for the firewall configuration, it would be reasonable to allow access to SIP ports and RTP ports only.

By default, SIP uses UDP port 5060, additionally, if TCP is enabled, TCP port 5060 and if TLS connections are configured for asterisk, also TCP port number 5061 is used.

We show a typical sample iptables firewall configuration where UDP, TCP and TLS connections are allowed to the asterisk box from everywhere and SSH management is permitted from secure locations:

```
IPTABLES=/sbin/iptables      # set proper location for the iptables
binary
$IPTABLES -t filter -P INPUT DROP # change policy to drop every
incoming traffic by default, that is not explicitly allowed later
# enable icmp
$IPTABLES -t filter -A INPUT -p icmp -j ACCEPT
# enable localhost commms
$IPTABLES -t filter -A INPUT -i lo -j ACCEPT
# enable tracked returning data for existing connections
$IPTABLES -t filter -A INPUT -m state --state ESTABLISHED,RELATED -j
ACCEPT
# enable DNS replies
```

```
$IPTABLES -t filter -A INPUT -p udp --dport 53 -j ACCEPT
# enable RTP traffic, must correspond with rtpstart and rtpend in
asterisk's rtp.conf
$IPTABLES -t filter -A INPUT -p udp --dport 10000:20000 -j ACCEPT
# enable SIP/TLS
$IPTABLES -t filter -A INPUT -p tcp --dport 5061 -j ACCEPT
# enable SIP udp + tcp
$IPTABLES -t filter -A INPUT -p udp --dport 5060 -j ACCEPT
$IPTABLES -t filter -A INPUT -p tcp --dport 5060 -j ACCEPT
# ssh from trusted client 1
$IPTABLES -t filter -A INPUT -p tcp -s 1.2.3.4/32 --dport 22 -j
ACCEPT
# ssh from trusted client 2
$IPTABLES -t filter -A INPUT -p tcp -s 2.3.4.5/32 --dport 22 -j
ACCEPT
```

## 2.5 Do not let bad guys enter in – deploy IPS in front of your Asterisk

Most common SIP targeted attacks against asterisk have few things in common and can be simply abstracted and described by certain pattern - signature. There are quite many vendors of signature based intrusion detection systems and intrusion prevention systems, but why not to use an open source sibling to open source asterisk. For this purpose you can select either well known [SNORT](#) or it's younger brother [SURICATA](#), both of them understand the same signature description language.

Typical [IDS](#) software just captures live packet stream and recognizes malicious attempts, then it logs important data like time of occurrence, source and destination IP of attack, payload data, description of recognized pattern and the appropriate reference to vulnerability or security database. This passive mode of protection does not impact the traffic flow, it just monitors traffic and signals when something goes wrong.

Further protection for VoIP (and other) services comes with [IPS](#) systems. When IPS detects an attack, in addition to what IDS already does, it actively stops delivery of malicious

data to target systems and can also do additional reactive actions like blocking attacker's IP address in firewall etc.

IDS/IPS can be deployed directly on a local machine with asterisk or when you are trying to protect multiple hosts at once, in a standalone system placed in front of VoIP servers. In this case IDS/IPS is placed in the middle between attackers and VoIP systems that should be protected.

Snort comes equipped with many standardized and useful [SIP rules](#), some of them being generic in terms they recognize generic attack patterns like packets not conforming to RFC with wrong length of some field or format if some header data, others rules are describing exact targeted attack on certain versions of SIP platforms and software. Also it's not very difficult to develop new rules with Snort's flexible and versatile rule language based on regular expressions.

Let's analyze one of SIP rules included in voip.rules snort rule file from the default rules distribution.

Snort SIP rules look very similar to this:

```
alert udp $EXTERNAL_NET any -> $HOME_NET 5060 (msg:"VOIP-SIP CSeq header format string attempt"; content:"CSeq|3A|"; nocase; content:"%"; distance:0; pcre:"/^CSeq\x3A\s*[\r\n%]*%/smi"; reference:url,www.ee.oulu.fi/research/ouspg/protos/testing/c07/sip/; reference:url,www.ietf.org/rfc/rfc3261.txt; classtype:attempted-dos; sid:11991; rev:3;)
```

where “*alert*” keyword means snort should log only every SIP packet matching this rule. If we also want to stop packet not to reach its destination, “*alert*” must be replaced with “*drop*” and snort must be switched in so called “inline” mode. The “udp” word represents transport protocol (can be changed to “tcp” if TCP is protocol used for SIP transport).

The following five fields are source ip/network and source port, direction of packet travel and destination ip/network and destination port. *\$EXTERNAL\_NET* and *\$HOME\_NET* are variable references configured earlier in the snort configuration file. Usually, *EXTERNAL\_NET* you set to everything, *HOME\_NET* to addresses or subnets you are trying to protect. Destination port is for SIP, obviously 5060.

Finally, in parentheses, keywords describing what to match, where to match and what security problem the rule is referencing and similar options are specified. The most interesting expression what is matched and considered malicious is the `pcrc:"/^CSeq\x3A\s*[\r\n%]*%/smi"`

In this case, it matches packet containing word CSeq at the beginning of a line, then “:” colon, one or more space separator characters, then whatever except newline and “%” character followed with %character. It is an example of classical string formatting overflow attempt, where “%s” or “%d” placeholders are substituted with parameters following C string library function call like “sprintf”.

Currently snort's voip.rules rule file contains over 100 rules related to SIP attacks and cousin SDP protocols, which is fairly good place to start with IPS/IDS protection of SIP equipment. Moreover, alerts and detections can be logged in a database like MySQL, PostgreSQL or others and can further be analyzed with either custom built software or one of a dozen web interfaces developed for this very purpose.

In case of suspicion that your snort does something odd with your SIP traffic, it is very easy to switch it to the passive mode or switch it off completely using a configuration settings or command line options.

**IMPORTANT:** if you try to use snort as an active IPS system, it is most likely that you have to build snort from source code on your own, because in standard distribution build, only 4 DAQ modules are present. Among them, AF\_PACKET DAQ mode supports inline mode and should serve for IPS functionality, we have experienced troubles with doing so, especially if you are trying to use snort in a more complicated failover system with interface bonding or bridging. The DAQ modes well suited for purpose of IPS are “nfq” or “ipq”, the former being superior to the latter. After successful compilation, just verify snort supports “nfq” or “ipq” inline modes with the following command:

```
#snort -daq-list
```

Available DAQ modules:

```
pcap(v3): readback live multi unpriv
```

```
nfq(v6): live inline multi
```

```
ipfw(v2): live inline multi unpriv
```

```
dump(v1): readback live inline multi unpriv
```

```
afpacket(v4): live inline multi unpriv
```

After compiling your Snort with IPS/NFQ support for inline mode, you can redirect your monitored and filtered traffic with iptables rule, like in the following sample:

```
/sbin/iptables -t filter -A FORWARD -p udp --dport 5060 -d  
subnet/mask -j NFQUEUE --queue-num 1
```

With this rule, for example, you are redirecting SIP UDP traffic to protected network (subnet/mask) into a user-land queue, where snort is waiting for further traffic analysis. (Choose whatever queue number, but it must match snort's configured one).

When snort detects any attack or malicious SIP packet, the corresponding alert log will look like this:

```
Nov 11 01:20:35 ips1 snort[5716]: [1:11975:4] VOIP-SIP Via header  
missing SIP field [Classification: Misc activity] [Priority: 3]  
{UDP} 191.3.23.3:32339 -> 112.71.123.12:5060
```

See few screenshots made from our real deployment of snort in IPS role protecting multiple asterisk hosts, with [Snorby](#) web user interface used for analysis where you can see overall statistics of detected attacks and detailed detection severity and packet details:



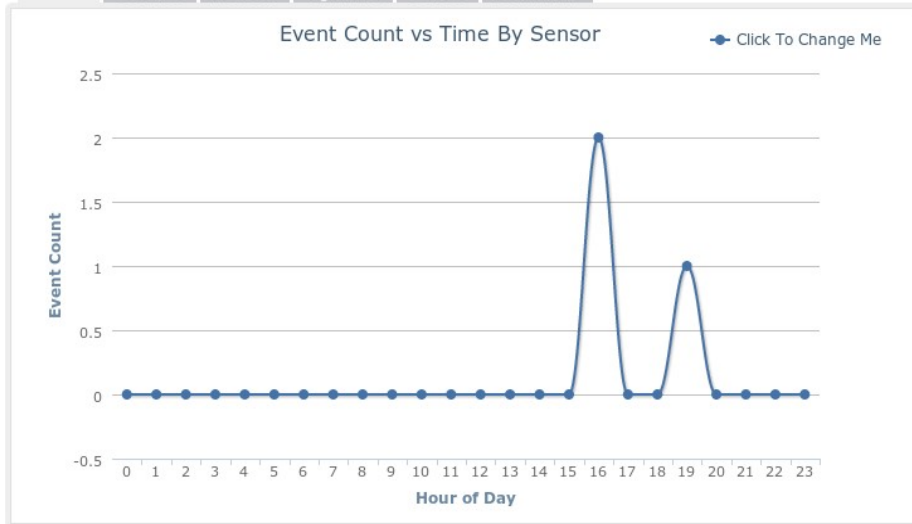
Dashboard

More Options

LAST 24 TODAY YESTERDAY THIS WEEK THIS MONTH THIS QUARTER THIS YEAR



Sensors Severities Protocols Signatures Sources Destinations



TOP 5 SENSOR

unknown: 58

TOP 5 ACTIVE USERS

Administrator 3  
ipex 0

LAST 5 UNIQUE EVENTS

SIP request line equal... 1  
VOIP-SIP To header inv... 3  
VOIP-SIP SDP negative ... 2  
VOIP-SIP From header f... 2  
VOIP-SIP from header f... 2

ANALYST CLASSIFIED EVENTS

Unauthorized Root Access 0  
Unauthorized User Access 0  
Attempted Unauthorized... 0  
Denial of Service Attack 0  
Policy Violation 0  
Reconnaissance 0  
Virus Infection 0  
False Positive 0

Listing Events (58 unclassified events)

 Hotkeys

 Classify Event(s)

<input type="checkbox"/>	Sev.	Sensor	Source IP	Destination IP	Event Signature	Timestamp
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP oversized time value	11/19/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP oversized time value	11/19/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP oversized time value	11/19/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP attribute buffer overflow attempt	11/18/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP attribute buffer overflow attempt	11/18/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	SIP request line equal To zero	11/18/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP Expires header overflow attempt	11/18/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP Expires header overflow attempt	11/18/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	VOIP-SIP Content-Type header format string attempt	11/18/2011
<input type="checkbox"/>	★ 3	unknown:	SK	CZ	VOIP-SIP SIP URI overflow attempt	11/18/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	VOIP-SIP To header invalid characters detected	11/17/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	VOIP-SIP To header invalid characters detected	11/14/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP negative time value	11/14/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP oversized time value	11/14/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP Via header hostname buffer overflow attempt	11/14/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	VOIP-SIP CSeq header format string attempt	11/14/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	VOIP-SIP CSeq header format string attempt	11/14/2011
<input type="checkbox"/>	★ 3	unknown:	SK	CZ	VOIP-SIP Via header missing SIP field	11/14/2011
<input type="checkbox"/>	★ 3	unknown:	SK	CZ	VOIP-SIP Via header missing SIP field	11/14/2011
<input type="checkbox"/>	★ 2	unknown:	SK	CZ	VOIP-SIP To header invalid characters detected	11/13/2011
<input type="checkbox"/>	★ 1	unknown:	SK	CZ	VOIP-SIP SDP oversized time value	11/13/2011

Dashboard My Queue (3) Events Sensors Search Administration

Listing Events (58 unclassified events) Hotkeys Classify Event(s)

Sev.	Sensor	Source IP	Destination IP	Event Signature	Timestamp
1	unknown:	SK	CZ	VOIP-SIP SDP oversized time value	11/19/2011

**IP Header Information** Perform Mass Classification Event Export Options Permalink

Source	Destination	Ver	Hlen	Tos	Len	ID	Flags	Off	TTL	Proto	Csum
		4	5	0	1361	39414	0	0	55	17	57922

**Signature Information**

Generator ID	Signature ID	Signature Revision	Activity (6/58)
1	11984	3	10%

Query Signature Database View Rule

**UDP Header Information**

Src Port	Dst Port	Len	Csum
42739	5060	1341	18257

**Payload** Hex Ascii

```

000004E: 32 37 33 39 3b 62 72 61 6e 63 68 3d 7a 39 68 47 34 62 4b 2d 39 32 38 70 6a 74 2739;branch=z9hG4bK-928pjt
0000068: 64 69 79 61 75 66 3b 72 70 6f 72 74 0a 46 72 6f 6d 3a 20 22 31 30 30 22 20 3c diyauf;rport.From:"100"<
0000082: 73 69 70 3a 74 65 73 74 40 31 39 35 2e 33 2e 31 37 30 2e 33 3e 3b 74 61 67 3d sip:test@>;tag=
000009C: 6d 77 67 6e 70 78 39 76 6b 73 0a 54 6f 3a 20 3c 73 69 70 3a 31 34 37 40 32 31 mwgnpx9vks.To:<sip:147@
00000B6: 32 2e 37 31 2e 31 32 39 2e 32 30 3b 75 73 65 72 3d 70 68 6f 6e 65 3e 0a 43 61 >>>;user=phone>.Ca
00000D0: 6c 6c 2d 49 44 3a 20 33 63 32 36 37 30 32 32 33 65 34 37 2d 35 69 69 64 78 35 ll-ID:.3c2670223e47-5iidx5
00000EA: 6b 6b 36 68 62 64 0a 43 53 65 71 3a 20 33 36 35 20 49 4e 56 49 54 45 0a 4d 61 kk6hbd.CSeq:.365.INVITE.Ma
0000104: 78 2d 46 6f 72 77 61 72 64 73 3a 20 37 30 0a 43 6f 6e 74 61 63 74 3a 20 3c 73 x-Forwards:.70.Contact:<s
000011E: 69 70 3a 74 65 73 74 40 3a 34 32 37 33 39 3b 74 72 61 6e 73 70 6f 72 74 3d 75 ip:test@:42739;transport=u
0000138: 64 70 3e 3b 72 65 67 2d 69 64 3d 31 0a 58 2d 53 65 72 69 61 6c 6e 75 6d 62 65 dp>;reg-id=1.X-Serialnumbe
0000152: 72 3a 20 30 30 30 34 31 33 33 42 43 44 37 33 0a 50 2d 4b 65 79 2d 46 6c 61 67 r:.0004133BCD73.P-Key-Flag
000016C: 73 3a 20 6b 65 79 73 3d 22 33 22 0a 55 73 65 72 2d 41 67 65 6e 74 3a 20 73 6e s:.keys="3".User-Agent:.sn
0000186: 6f 6d 33 30 30 2f 38 2e 34 2e 33 32 0a 41 63 63 65 70 74 3a 20 61 70 70 6c 69 om300/8.4.32.Accept:.appli
00001A0: 63 61 74 69 6f 6e 2f 73 64 70 0a 41 6c 6c 6f 77 3a 20 49 4e 56 49 54 45 2c 20 cation/sdp.Allow:.INVITE,.
00001BA: 41 43 4b 2c 20 43 41 4e 43 45 4c 2c 20 42 59 45 2c 20 52 45 46 45 52 2c 20 4f ACK,.CANCEL,.BYE,.REFER,.O
00001D4: 50 54 49 4f 4e 53 2c 20 4e 4f 54 49 46 59 2c 20 53 55 42 53 43 52 49 42 45 2c PTIONS,.NOTIFY,.SUBSCRIBE,
00001EE: 20 50 52 41 43 4b 2c 20 4d 45 53 41 47 45 2c 20 49 4e 46 4f 2c 20 55 50 44 .PRACK,.MESSAGE,.INFO,.UPD
0000208: 41 54 45 0a 41 6c 6c 6f 77 2d 45 76 65 6e 74 73 3a 20 74 61 6c 6b 2c 20 68 6f ATE.Allow-Events:.talk,.ho
0000222: 6c 64 2c 20 72 65 66 65 72 2c 20 63 61 6c 6c 2d 69 6e 66 6f 0a 53 75 70 70 6f ld,.refer,.call-info.Suppo
000023C: 72 74 65 64 3a 20 74 69 6d 65 72 2c 20 31 30 30 72 65 6c 2c 20 72 65 70 6c 61 rted:.timer,.100rel,.repla
0000256: 63 65 73 2c 20 66 72 6f 6d 2d 63 68 61 6e 67 65 0a 53 65 73 73 69 6f 6e 2d 45 ces,.from-change.Session-E
0000270: 78 70 69 72 65 73 3a 20 33 36 30 30 3b 72 65 66 72 65 73 68 65 72 3d 75 61 73 xpires:.3600;refresher=uas
000028A: 0a 4d 69 6e 2d 53 45 3a 20 39 30 0a 41 75 74 68 6f 72 69 7a 61 74 69 6f 6e 3a .Min-SE:.90.Authorization:
00002A4: 20 44 69 67 65 73 74 20 75 73 65 72 6e 61 6d 65 3d 2d 74 65 73 74 22 2c 72 65 .Digest.username="test",re
00002BE: 61 6c 6d 3d 22 32 31 32 2e 37 31 2e 31 32 39 2e 32 30 22 2c 6e 6f 6e 63 65 3d alm=">>>","nonce=

```

## 2.6 Monitor critical files for changes

One of the crucial tasks of administration of production servers is keeping eye on integrity of critical system files so that it is possible to detect intrusions into the system. On hacked servers, often genuine system binaries are replaced with backdoored versions, or trojan horses. To get informed about these situations, typically these system files and binaries are processed to produce fingerprints – cryptographic hashes, usually with MD5, SHA1,

SHA256 or SHA512 algorithm. These fingerprints MUST be stored externally out of monitored server so that the regular comparison of current system fingerprints with stored ones can produce trustworthy results.

Also, it is necessary to maintain the externally stored hash database after each system upgrade in order to have current and stored fingerprints. One typical tool used for file integrity protection is [tripwire](#), but many more opensource and commercial exist.

It is necessary to emphasize that filesystem integrity checking does not represent a suitable protection against LKM rootkits where kernel syscalls `open()`, `read()`, `exec()` are hooked.

## 2.7 Hide your identity

During communication with other SIP clients, peer or proxies, asterisk inserts own identification in user agent and similar headers and SDP session name. By default, these headers contain the word “Asterisk” with exact running version appended. Most of the times this is not the best option from the security point of view, because asterisk is advertising exact running version publicly to everyone. Potential attacker can simply guess that PBX is running on asterisk, obtain asterisk's version number and focus on existing known vulnerabilities against the specific asterisk installation.

Here comes sample snippet of SIP communication captured during session with asterisk with Asterisk name and version information marked bold:

```
SIP/2.0 200 OK
Via: SIP/2.0/TLS 192.168.1.2:29541;branch=z9hG4bK-d8754z-
33540d66b8658c25-1---d8754z-;received=1.2.3.4;rport=46849
From: "voipuser12"<sip:voipuser50@somedomain.com>;tag=51a9264c
To: <sip:200@somedomain.com>;tag=as435bba3a
Call-ID: NzE4Y2VmZjZhODkyY2Y2NjExNjI0YzY0OGZmN2Q4MTk.
CSeq: 2 INVITE
Server: Asterisk PBX 1.8.7.1
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, SUBSCRIBE, NOTIFY,
INFO, PUBLISH
Supported: replaces, timer
Contact: <sip:200@8.7.6.5:5061;transport=TLS>
```

```
Content-Type: application/sdp
Content-Length: 375

v=0
o=root 868599863 868599863 IN IP4 8.7.6.5
s=Asterisk PBX 1.8.7.1
c=IN IP4 8.7.6.5
t=0 0
m=audio 14350 RTP/SAVP 18 3 97 101
a=rtpmap:18 G729/8000
a=fmtp:18 annexb=no
a=rtpmap:3 GSM/8000
a=rtpmap:97 speex/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
a=ptime:20
a=sendrecv
a=crypto:1 AES_CM_128_HMAC_SHA1_80
inline:YvLbgvR7LLEtAMVPEMF+jJzBnxEoSH66P6VBFQZ
```

Changing this information is easy, only two options need to be changed in sip.conf:

```
useragent=AnotherVoIP
sdpsession=AnotherVoIP
```

Other thing worth of changing with not of such in importance is the “sdpowner” option in sip.conf, that would change user name in the SDP protocol, the word “root” by default.

```
sdpowner=Chuck-Norris
```

By changing it according to the previous example, the message capture would look like:

```
...
v=0
o=Chuck-Norris 365808949 365808949 IN IP4 9.8.7.6
s=PBX
c=IN IP4 9.8.7.6
t=0 0
m=audio 13256 RTP/SAVP 18 3 97 101
...
```

Now it is not so easy to recognize this PBX runs on unix-like system.

## 2.8 Harden valid SIP extension discovery

Most common attacks involved with SIP infrastructure are related to valid extensions / usernames scanning and bruteforcing. The potential attacker tries to obtain the list of SIP logins or extensions served by the attacked SIP server by various forms of scanning, like REGISTER scan, INVITE scan or OPTIONS scan.

All of these can be simulated or performed using commonly accessible software, like sipvicious or sipsak.

For example, REGISTER method flood for extensions between 200 and 10000 can be performed using single command:

```
#./svwar.py --force -e 200-10000 10.0.0.10 -m REGISTER
```

In order not to allow attacker to guess valid extensions in REGISTER or INVITE attempts, asterisk should always return the same error code and message in case extension does not exist or in case attacker has tried invalid password. Enable “alwaysauthreject” SIP option in asterisk's sip.conf configuration file, see the following sample:

```
alwaysauthreject = yes
```

## 2.9 Use strong passwords, really

This old and well known formula of IT security applies also to asterisk server and VoIP. Most VoIP abuses are still made using dictionary brute-force attacks against accounts that were set up with simple guessable password. A typical example of such weak passwords is the situation, where SIP login and password matches, passwords are based on dictionary words or too short passwords (<8 characters ) are used that can be brute-force attacked. We suggest to use at least 16 characters long passwords randomly generated and use whole range of characters, including numbers, uppercase and lowercase characters, symbols.

Examples of weak passwords: `jane23`, `062012355`, `00001111`

Examples of strong passwords: `Chee8muo6iephade`, `iex5sae7phu4Ahk2`

Having weak passwords can sometimes cause significant financial loss, if hackers obtain SIP account by means of guessing its password, he can generate bills within scale of thousands euros if he makes long international calls or calls extra-tolled lines.

## 2.10 Know how to failover

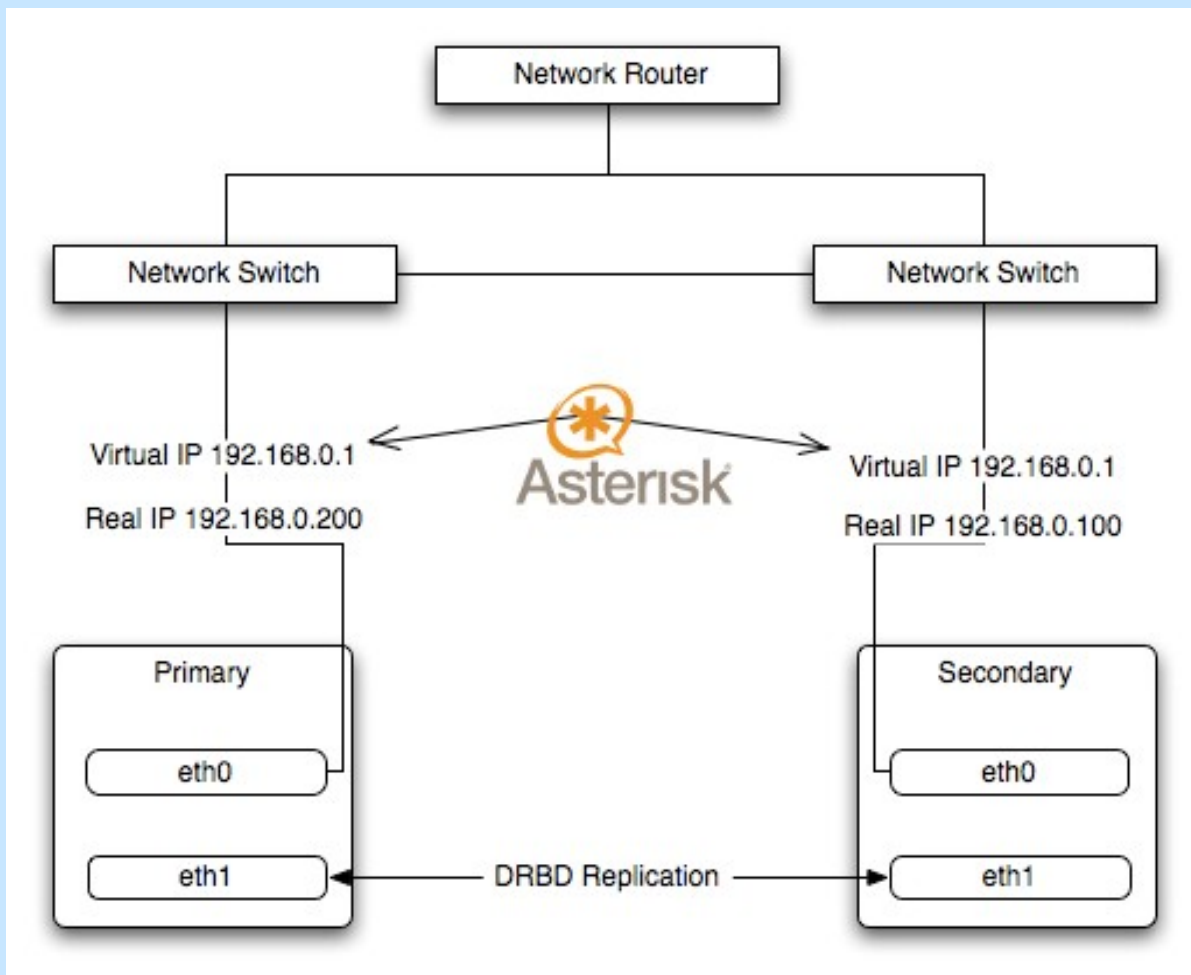
One of the very important and underestimated aspect of IT security is service availability, in our case, availability of the asterisk SIP service, so that users can make call any time they want. No asterisk installation can be considered secure enough if it fails along with broken hard drive, power supply, damaged Digium card or erased configuration.

Quite common production scenario to overcome the risk of crashed server is to use multiple servers in a so called failover cluster. In the Linux world, two solutions offering failover functionality exist and are used most often:

1. Heartbeat – part of Linux-HA projects
2. Redhat cluster suite

Both provide necessary functionality and intelligence for failover behaviour like resource or service definition and abstraction (in our case virtual moving IP address for

accessing SIP server and the asterisk application) and their movement between machines, either manually or, yet better, automatically in a case of a detected failure (machine goes down, machine loses connection, ...).



In this simple demonstrative scenario, virtual IP address transferable between 2 servers would be IP 192.168.0.1, on which asterisk would be configured for client access, and the asterisk itself would be started on currently active node and listening on the IP address 192.168.0.1 accepting incoming connections.

## 2.11 Protect your dialplans

A proper separation of extensions in dialplan contexts is one of most important things



to keep your eyes on during asterisk configuration.

**WARNING !** Use separate dialplan contexts for incoming and outgoing extensions, especially if outgoing ones provide toll services. Anonymous incoming calls should never reach contexts that can reach tolled extensions, otherwise you are exposed to risk of high financial loss.

During context creation, keep in mind, that once given channel can enter particular dialplan context, it can potentially access every extension configured within that same context.

For example, in the following sample context, user can by dialing any 4 or more digit number call outside numbers and make toll fees rise up very quickly:

```
100 => Dial(SIP/user1, 30, tT);
101 => Dial(SIP/user2, 30, tT);
_XXX. => Dial(ZAP/g1/${EXTEN});
```

It is also very important to define proper contexts for SIP clients / phones definition. If group of SIP users should access for example only Czech extensions and only via given SIP external line, you should create separate context, that might look like following one:

```
context myczechsipprovider {

    includes {
        local-extensions;
    };

    _00421XXXXXXXXXX! => {
        Dial(SIP/${EXTEN}@12345678, ${RINGTIME}, tT);
        Congestion;
    }
}
```

Afterwards, you assign this context for every user you plan to access Czech and local

numbers only:

```
[loginxxx]
type=friend
secret=anothersecretpassword
context= myczechsipprovider
```

Extensions like

```
_00421XXXXXXXXX! => {
Dial(SIP/${EXTEN}@12345678,${RINGTIME},tT);
Congestion;
}
```

from the previous sample should never be accessible by anonymous users !

Special care should be taken also with use of pattern matching extensions. Asterisk by default waits for another digit input if it has more than one extension matching number dialed so far. In case there are no more digits coming for a preconfigured time, it will run first dialplan matching dialed number, which is NOT the first one defined in the dialplan configuration, because the order of extensions considered by asterisk may be different that you define.

Be sure to take into account various other serious security problems arising from use various dialplan applications, like “System”, “Record” and similar, accessing your filesystem or operating system directly. If various variables are used insecurely, they can provide simple gateway straight into your system by running shell commands of attacker's choice or rewriting critical system files. In the dialplan configuration same rules apply as in other information security areas – never trust user input and provided data, it may be tainted and contain malicious content resulting in command injection etc.

Following dialplan could have really disastrous consequences for your system once exploited, because malicious user can in many cases set his CALLERID to whatever he wants (name; shutdown -h now for example):

```
666 => {  
    System(echo huhu ${CALLERID} >> /tmp/calllog);  
    Playback(bye);  
};
```

## 2.12 Log to remote servers

By default asterisk logs its activity, including failed logins, into local file, most often `/var/log/asterisk/messages`. For the purposes of eventual abuse or post-crime investigation, it is necessary to have access to all log records with the assurance no-one could manipulate these (when asterisk machine is hacked).

Asterisk logging can be extended to standard unix “syslog” logging, which can be redirected to remote or multiple remote locations. Logging in asterisk is configured in `logger.conf` and syslog logging can be turned on like in the following example:

```
syslog.local0 => notice,warning,error
```

or

```
syslog.local0 => *
```

In the first sample, only messages with notice, warning and error priority are logged to syslog facility `local0`, in the second sample, all messages go to the same syslog facility.

Redirecting logs with facility configured previously in asterisk's `logger.conf` to remote host via UDP in rsyslog logger daemon, for example, is done with the following directive in `rsyslog.conf`:

```
local0.* @10.0.0.133
```

And via TCP:

`local0.* @@10.0.0.134`

The best option for remote syslog logging provides new RELP protocol supported by rsyslog. In this case, rsyslog is able to provide reliable syslog message delivery to destination and in case it becomes unreachable for a time, it spools all messages into preconfigured spool file. Messages are then delivered later, when remote logger daemon becomes reachable again.

In order to achieve reliable syslog delivery, add following options in rsyslog.conf:

```
$ModLoad imrelp
$ModLoad omrelp
$InputRELPServerRun 2514
```

The first 2 options load necessary modules (rsyslog is modular), last creates RELP listener on TCP port 2514. Next, append:

```
$WorkDirectory /var/spool/rsyslog # where to place spool files
$ActionQueueFileName centralsyslog # spool file prefix for this
destination
$ActionQueueMaxDiskSpace 1g # 1gb space limit
$ActionQueueSaveOnShutdown on # save messages to disk on shutdown
$ActionQueueType LinkedList # run asynchronously
$ActionResumeRetryCount -1 # infinite retries if host is down
local0.*
:omrelp:10.0.0.10:2514;RSYSLOG_ForwardFormat
```

If SELinux MAC security is enabled, choose spool directory with proper security labels, otherwise logging would not work (rsyslog will not use RELP – check your security audit logs).

## **2.13 Put asterisk on diet – disable unnecessary functionality**

Default installation of asterisk includes wide range of functionality, in loaded modules and configuration files, much of them never will be used in your deployment scenario. For example, if you do not use or plan to use H323 as your voice communication channel, it's better not to compile, load or configure H323 support at all.

The first step available for minimizing asterisk functionality profile is compilation process. Select only necessary channel providers, applications or dialplan functions. This detailed upfront selection may be a bit daunting process and may take considerable time to check every item in each selection menu, but pays off at the end. You get asterisk with minimal necessary functionality, you risk less code bugs and crashes compiled in, you are exposed to much narrower attacking surface and, finally, you may save some computing resources as well.

```
root@nethemba:~/sr/src/asterisk-1.8.7.1
File Edit View Terminal Help

*****
Asterisk Module and Build Option Selection
*****

Press 'h' for help.

[ ] chan_agent
XXX chan_alsa
[ ] chan_bridge
XXX chan_console
XXX chan_dahdi
XXX chan_gtalk
XXX chan_h323
[ ] chan_iax2
XXX chan_jingle
[*] chan_local
[ ] chan_mgcp
XXX chan_misdn
[ ] chan_multicast_rtp
XXX chan_nbs
[ ] chan_oss
[ ] chan_phone
[*] chan_sip
[ ] chan_skinny
[ ] chan_unistim
XXX chan_usbradio
XXX chan_vpb

Agent Proxy Channel
Depends on: chan_local(M), res_monitor(M)

Support Level: core
```

The second way of lowering asterisk's functionality footprint is to disable loading of selected modules when autoloading of modules is enabled. This is done with “noload” directive in modules.conf configuration file. For example, if your asterisk deployment does not use IAX2 protocol and voicemail, fax or similar functionality, you would disable loading corresponding modules as in the following sample:

```
noload = chan_iax2.so
noload = app_voicemail.so
noload = app_minivm.so
```

```
noload = res_fax.so
noload = pbx_dundi.so
...
```

The list of modules loaded in running instance of asterisk can be obtained from asterisk's console using “module show” command. The output will look similar to following sample:

```
res_calendar.so          Asterisk Calendar integration
1
res_ael_share.so         share-able code for AEL
0
res_crypto.so           Cryptographic Digital Signatures
0
res_smdi.so             Simplified Message Desk
Interface (SMDI) 0
res_adsi.so             ADSI Resource
0
res_srtp.so             Secure RTP (SRTP)
0
res_monitor.so          Call Monitoring Resource
0
res_stun_monitor.so     STUN Network Monitor
0
res_speech.so           Generic Speech Recognition API
0
res_agi.so              Asterisk Gateway Interface
(AGI)                   1
...
...
```

The third column from the previous output represents how many times the given module is used, zero means it is not currently used (however, it may be user just a few seconds after you get this output, in case there's a new ongoing call that will utilize few codec modules, function modules, channel module etc).

You can also unload currently unused module with “module unload modulename.so” command from console or via asterisk AGI.

Here is short list of possible modules for eventual consideration of “noloading”:

- `chan_agent.so`: Used for the ACD see more information in Asterisk Agents docs
- `chan_h323.so`: H.323 VoIP driver. See Asterisk H323 Channels.
- `chan_iax.so`: IAX version 1, see Asterisk IAX channels
- `chan_iax2.so`: IAX version 2.
- `chan_local.so`: Local Proxy Channel. See Asterisk local channels
- `chan_mgcp.so`: MGCP VoIP protocol. See Asterisk MGCP channels
- `chan_modem.so`: Base driver for modems. See Asterisk modem channels
- `chan_modem_aopen.so` `chan_modem_bestdata.so`: Old attempts to make voice modems work.
- `chan_modem_i4l.so`: ISDN modem driver, alternative to the CAPI driver.
- `chan_oss.so`: OSS sound driver, turns sound card into phone channel.
- `chan_phone.so`: Linejack cards driver.
- `chan_sip.so`: SIP VoIP driver. See Asterisk SIP Channels
- `chan_skinny.so`: Skinny VoIP driver. Used by Cisco call manager. See Asterisk skinny channels
- `chan_zap.so`: Zapata channel driver. See Asterisk ZAP Channels
- `codec_a_mu.so`: A-law and Mulaw direct Coder/Decoder. G711
- `codec_adpcm.so`: Adaptive Differential PCM Coder/Decoder.
- `codec_alaw.so`: A-law Coder/Decoder.
- `codec_gsm.so`: GSM/PCM16 (signed linear) Codec Translator.
- `codec_ilbc.so`: iLBC/PCM16 (signed linear) Codec Translator.
- `codec_lpc10.so`: LPC10 2.4kbps (signed linear) Voice Coder.
- `codec_ulaw.so`: Mu-law Coder/Decoder.



- `format_g729.so`: Raw G729 data. Requires license
- `format_gsm.so`: Raw GSM data.
- `format_h263.so`: Raw h263 data.
- `format_jpeg.so`: JPEG (Joint Picture Experts Group) Image Format.
- `format_pcm.so`: Raw uLaw 8khz Audio support (PCM).
- `format_pcm_alaw.so`: Raw aLaw 8khz PCM Audio support.
- `format_vox.so`: Dialogic VOX (ADPCM) File Format.
- `format_wav.so`: Microsoft WAV format (8000hz Signed Linear).
- `format_wav_gsm.so`: Microsoft WAV format (Proprietary GSM).
- `pbx_config.so`: Text Extension Configuration.
- `pbx_spool.so`: Outgoing Spool Support Asterisk auto-dial out.
- `pbx_wilcalu.so`: Wil Cal U Auto Dialer.
- `res_adsi.so`: Resource for ADSI applications.
- `res_agi.so`: Asterisk Gateway Interface. See Asterisk AGI
- `res_crypto.so`: Resource for cryptographic applications.
- `res_indications.so`: Resource for indications(`ring`,`busy`,`congestion`,`dialtone`). See Asterisk config `indications.conf`
- `res_monitor.so`: Resource for recording channels.
- `res_musiconhold.so`: resource for music on hold. See Asterisk `cmd musiconhold`
- `res_features.so`: resource for parking calls (using `features.conf`).

## 2.14 Other common security related advices

- Keep previous versions / binaries of asterisk. In case you are using compiled versions of asterisk, it can be handy and life-saver to have previous, functional and tested version by hand, if newer version manifest itself problematic and asterisk service can be quickly reversed back to the previous functioning state. One of possible means of achieving this is to use

“*configure –prefix=/usr/local/asterisk-x.y.z*” schema for having each version of asterisk in it's own directory (eg. /usr/local/asterisk-1.6.2, /usr/local/asterisk-1.8.7.2, etc).

- Use stable asterisk versions – this rule applies to almost every kind of software. Trying experimental, testing and patched code may result in stability and security problems as well.
- Use hardware echo cancelers – the more you offload your CPU , the more channels can be served with asterisk and thus you are more compliant with “service availability” security paradigm.
- Regularly update your Zaptel hardware cards with latest fixes and firmware, especially if you are experiencing stability problems using these devices.

## 3 References

- <http://www.ietf.org/rfc/rfc3261.txt>
- <http://www.voip-info.org/>
- <http://www.asterisk.org/>
- <http://www.voipsa.org/>
- <http://www.hackingvoip.com/>
- <http://www.wikipedia.org/>