



Seguridad en servidores CentOS con **Elastix**



Seguridad en Servidores CentOS con Elastix[®] + Buenas Prácticas

V. 0.8.6



Rodrigo A. Martin

Este artículo se distribuye bajo la licencia Attribution 3.0 de Creative Commons, más información:

<http://creativecommons.org/licenses/by/3.0/>

El presente documento tiene solo fines informativos y su contenido está sujeto a cambios sin notificación alguna.



Tabla de contenido

<i>Introducción</i>	3
Firewall-Iptables	3
Asegurando SSH en su sistema	5
Fail2ban	7
<i>Port-knocking o Golpeo de Puertos: "Llamar antes de entrar"</i>	11
Buenas Prácticas en nuestro Servidor	16
Siete pasos para mejorar la seguridad SIP en Asterisk	16
Backdoor detectado en FreePBX (hasta la versión 2.0.1 de Elastix - Agosto 2010).....	18

Introducción

Antes de comenzar el desarrollo del presente, debo aclarar que no se trata de un manual que cubre todos los aspectos de seguridad a tener en cuenta al poner un servidor elastix en producción, sino, ciertas recomendaciones y configuraciones básicas del S.O junto con otros aplicativos, extraídas de libros, manuales y experiencias personales que nos ayudaran a mantenerlo seguro y más si este, no está dentro de una VPN o atrás de un Firewall físico (que sería lo ideal).

Mi recomendación inicial es cambiar todos los passwords que trae elastix por defecto, esto lo podemos encontrar en el libro "Elastix a Ritmo de Merengue" capítulo 12, (desde la versión 2.0 en Elastix se solicita que se carguen los passwords al final de la instalación) otra observación importante es que realicen todo este tipo de pruebas y configuraciones sobre servidores en laboratorio (un excelente aliado para esto es la Virtualización) y así comprender realmente lo que se esta haciendo permitiéndonos equivocarnos o inclusive mejorar las mismas, dicho esto, comencemos:

Firewall-Iptables

Iptables es el firewall que trae instalado nuestro CentOS , este es muy popular y uno de los más utilizados en distribuciones Linux por su robustez y estabilidad. Lo ideal es que el firewall sea un componente independiente separado de los demás servidores, pero si no tenemos dicho dispositivo, podemos aplicar estas reglas directamente sobre nuestro servidor y así establecer políticas con los puertos de red del mismo.

Lo que haremos por medio de esta herramienta será habilitar o "abrir" solo los puertos que estamos utilizando y cerrar todos los demás brindando así mayor protección al servidor. Es importante el orden de ingreso de las reglas, ya que *iptables* va "macheando" por estas desde la primera que se ingresa, hasta llegar a la última, si no encuentra ninguna coincidencia, la última regla que escribiremos será cerrar todos los demás puertos, así que si no coincide con los especificados al principio, no podrá gestionar alguna conexión por dicho puerto.

En el ejemplo tomamos que la tarjeta de red por defecto es "eth0"

Permitir acceso total desde la LAN (suponiendo que estamos en la red 192.168.0.0, deberá modificar este parametro dependiendo de su red)

```
# iptables -A INPUT -s 192.168.0.0/24 -j ACCEPT
```

Aceptando el tráfico SIP:

```
# iptables -A INPUT -p udp -m udp -i eth0 --dport 5060 -j ACCEPT
```

Aceptando el tráfico IAX2:

```
# iptables -A INPUT -p udp -m udp -i eth0 --dport 4569 -j ACCEPT
```

Aceptando el tráfico RTP (Suponiendo que no se ha alterado el archivo rtp.conf):

```
# iptables -A INPUT -p udp -m udp -i eth0 --dport 10000:20000 -j  
ACCEPT
```

Aceptando tráfico MGCP (solo aplicar si es que se va a usar. En la mayoría de los casos no es necesario):

```
# iptables -A INPUT -p udp -m udp -i eth0 --dport 2727 -j ACCEPT
```

Aceptando el tráfico de mensajería instantánea (si es que se va a acceder desde fuera):

```
# iptables -A INPUT -p tcp -i eth0 --dport 9090 -j ACCEPT
```

Aceptando el tráfico del servidor de correo y POP/IMAP:

```
# iptables -A INPUT -p tcp -i eth0 --dport 25 -j ACCEPT  
# iptables -A INPUT -p tcp -i eth0 --dport 110 -j ACCEPT  
# iptables -A INPUT -p tcp -i eth0 --dport 143 -j ACCEPT
```

Aceptando tráfico Web (HTTP y HTTPS) para poder visitar la interface administrativa de Elastix; si vamos a utilizar "Portknocking" (desarrollado más adelante) no abrir los puertos 80 ni 443:

```
# iptables -A INPUT -p tcp -i eth0 --dport 80 -j ACCEPT  
# iptables -A INPUT -p tcp -i eth0 --dport 443 -j ACCEPT  
# iptables -A INPUT -m conntrack --ctstate ESTABLISHED,RELATED -j  
ACCEPT
```

Aceptando tráfico SSH para poder remotamente a la consola del S.O (si utilizamos un puerto no estándar abrir ese y no el 22) misma aclaración anterior, si vamos a utilizar "Portknocking" (desarrollado más adelante) con este puerto, no abrir el mismo:

```
# iptables -A INPUT -p tcp -i eth0 --dport 22 -j ACCEPT
```

Finalmente denegando el acceso a todo lo demás en la interfaz eth0:

```
# iptables -A INPUT -p all -i eth0 -j DROP
```

Para verificar si las reglas se aplicaron correctamente usamos el comando:

```
# iptables -L
```

Si algo no salió de acuerdo a lo esperado, podemos hacer un "flush" (borrar) de todas las reglas con el comando:

```
# iptables -F
```

Para guardar nuestras reglas y que estas sean aplicadas cada vez que reiniciemos nuestro server usamos el comando:

```
# /sbin/service iptables save
```

Asegurando SSH en su sistema

OpenSSH (o Shell Seguro) se ha convertido en el estándar para el acceso remoto, reemplazando al protocolo telnet. SSH ha hecho que los protocolos como telnet sean redundantes, en su mayoría, por el hecho de que la conexión es encriptada y las contraseñas no son enviadas en texto plano para que todos la puedan ver. De todas maneras, una instalación por defecto de ssh no es perfecta, y cuando corremos un servidor ssh, hay algunos pasos simples que pueden endurecer dramáticamente una instalación.

1. Usar contraseñas Fuertes

Si usted está corriendo ssh y se encuentra expuesto al mundo exterior, una de las primeras cosas que podrá notar serán los intentos de los hackers tratando de acceder para adivinar su nombre de usuario/contraseña. Típicamente un hacker escaneará el puerto 22 (el puerto por defecto por el cual ssh escucha) para encontrar máquinas que estén corriendo ssh, y entonces intentarán un ataque de fuerza-bruta contra ésta. Con contraseñas fuertes en su lugar, felizmente cualquier ataque será registrado y notificado antes de que pueda tener éxito.

Si usted no está utilizando contraseñas fuertes, trate de escoger combinaciones que contengan:

8 Caracteres como mínimo

Mezcle letras mayúsculas y minúsculas

Mezcle letras y números

Use caracteres no alfabéticos (ej: caracteres especiales como ! " £ \$) (% ^ etc.)

Los beneficios de una contraseña fuerte no son específicos de SSH, pero tienen fuerte impacto en todos los aspectos de la seguridad del sistema.

2. Deshabilitar el acceso como root

La configuración del servidor SSH se encuentra almacenada en el fichero `/etc/ssh/sshd_config`. Para deshabilitar el acceso de root, asegúrese de tener la siguiente entrada:

```
#Prevent root logins:  
PermitRootLogin no
```

y reiniciar el servicio sshd:

```
service sshd restart
```

Si usted necesita acceder como root, acceda como un usuario normal y use el comando `su -`.

3. Deshabilitar el protocolo 1

SSH posee dos protocolos que se pueden usar, protocolo 1 y protocolo 2. El viejo protocolo 1 es menos seguro y debe estar deshabilitado, a menos que usted lo requiera para algo específico. Busque la siguiente línea en el fichero `/etc/ssh/sshd_config`, descoméntela y modifíquela como sigue:

```
# Protocol 2,1
Protocol 2
```

Reinicie el servicio `sshd`.

4. Usar un Puerto No-Estándar

Por defecto, `ssh` escucha las conexiones entrantes en el puerto 22. Para que un hacker determine si `ssh` se está ejecutando en su máquina, lo más probable es que escanee el puerto 22. Un método efectivo es ejecutar `ssh` en un puerto no-estándar. Cualquier puerto sin usar funcionará, pero es preferible usar uno por encima del 1024.

Muchas personas escogen el 2222 como un puerto alternativo (porque es fácil de recordar), de la misma forma que el 8080 es conocido, a menudo, como un puerto HTTP alternativo. Por esta razón, probablemente esta no es la mejor opción. De la misma forma que cualquier hacker escaneará el puerto 22, seguramente también escaneará el puerto 2222 solo como buena medida. Es mejor escoger cualquier puerto alto al azar que no sea usado por ningún servicio conocido.

Para hacer los cambios, añada una línea como esta a su fichero `/etc/ssh/sshd_config`:

```
# Ejecutar ssh en un puerto No-Estándar:
Port 22 #Cambiar Aquí!!! Por ej: Port 7634
```

y reinicie el servicio `sshd`.

Recuerde hacer cualquier cambio necesario a los puertos de reenvío en su **enrutador** y a cualquier regla aplicable al **firewall**.

Fail2ban

Fail2Ban es un analizador de logs que busca intentos fallidos de registro y bloquea las IP's de donde provienen estos intentos. Se distribuye bajo la licencia GNU y típicamente funciona en todos los sistemas que tengan interfaz con un sistema de control de paquetes o un firewall local.

Fail2Ban tiene una gran configuración pudiendo, además, crear reglas para programas propios o de terceros.

Instalación y Configuración

Debemos de Instalar python en nuestro servidor, para ello ejecutamos

```
[root@elastix]# yum install python*
```

Descargamos Fail2ban en la carpeta /usr/src desde su página en sourceforge

```
[root@elastix]# cd /usr/src
[root@elastix src]# wget
http://sourceforge.net/projects/fail2ban/files/fail2ban-stable/fail2ban-0.8.4/fail2ban-0.8.4.tar.bz2/download
```

Descomprimos e instalamos Fail2Ban

```
[root@elastix src]# tar xvf fail2ban-0.8.4.tar.bz2
[root@elastix src]# cd fail2ban-0.8.4
[root@elastix fail2ban-0.8.4]# python setup.py install
```

Con esto tendremos instalado Fail2Ban en /usr/share/fail2ban. Los scripts los encontramos en /usr/bin.

Ahora debemos de configurar Fail2Ban para que analice los logs que deseamos y bloquee IP,s y nos sean enviadas notificaciones vía e-mail.para ello modificaremos el archivo jail.conf que encontramos en /etc/fail2ban

```
[root@elastix src]# cd /etc/fail2ban
[root@elastix fail2ban]# vim jail.conf
```

Lo primero a modificar es el valor bantime, este valor determina el tiempo en segundos que quedará bloqueada la ip que esta atacándonos, por defecto el valor viene en 600.

Después buscamos el valor maxretry que serán el número de veces que una IP puede tener una autenticación fallida antes de ser bloqueada.

Para que busque intentos fallidos de logueo por ssh identificamos la siguiente sección:

```
[ssh-iptables]
enabled = false
filter = sshd
action = iptables[name=SSH, port=ssh, protocol=tcp]
sendmail-whois[name=SSH, dest=you@mail.com, sender=fail2ban@mail.com]
logpath = /var/log/sshd.log
maxretry = 5
```

y la modificamos de la siguiente manera

```
[ssh-iptables] # Configuración de fail2ban para el puerto ssh

enabled = true
filter = sshd
action = iptables[name=SSH, port=22, protocol=tcp]
        sendmail-whois[name=SSH, dest=miMail@mail.com,
sender=fail2ban@mail.com]
logpath = /var/log/secure #indicamos que log controlar
maxretry = 3 #le damos solo tres intentos
bantime = 36000 #tiempo de baneo expresado en segundos
```

Configurando Fail2ban para Asterisk

Ahora podemos configurarlo para que lea los registros de **Asterisk**

```
cd /etc/fail2ban/filter.d
```

```
Creamos el archivo asterisk.conf
vim asterisk.conf
```

Copiamos estas líneas...

```
# Fail2Ban configuration file
#
#
# $Revision: 250 $
#

[INCLUDES]

# Read common prefixes. If any customizations available -- read them
from
# common.local
#before = common.conf

[Definition]

#_daemon = asterisk

# Option: failregex
# Notes.: regex to match the password failures messages in the
logfile. The
#         host must be matched by a group named "host". The tag
"<HOST>" can
#         be used for standard IP/hostname matching and is only an
alias for
#         (?:::f{4,6}:)?(?P<host>\S+)
# Values: TEXT
#

failregex = NOTICE.* .*: Registration from '.*' failed for '<HOST>' -
Wrong password
          NOTICE.* .*: Registration from '.*' failed for '<HOST>' -
No matching peer found
```

```
NOTICE.* .*: Registration from '.*' failed for '<HOST>' -
Username/auth name mismatch
NOTICE.* <HOST> failed to authenticate as '.*'$
NOTICE.* .*: No registration for peer '.*' (from <HOST>)
NOTICE.* .*: Host <HOST> failed MD5 authentication for
'.*' (.*)
NOTICE.* .*: Registration from '.*' failed for '<HOST>' -
Device does not match ACL
NOTICE.* .*: No registration for peer '.*' \((from <HOST>\)
NOTICE.* .*: Host <HOST> failed MD5 authentication for
'.*' (.*)
NOTICE.* .*: Failed to authenticate user .*@<HOST>.*
NOTICE.* .*: Registration from '.*' failed for '<HOST>' -
Peer is not supposed to register
NOTICE.* .*: Registration from '.*' failed for '<HOST>' -
ACL error (permit/deny)
VERBOSE.*SIP/<HOST>-.*Received incoming SIP connection
from unknown peer
```

```
# Option: ignoreregex
# Notes.: regex to ignore. If this regex matches, the line is
ignored.
# Values: TEXT
#
ignoreregex =
```

Con esto le decimos a fail2ban que tiene que controlar eventuales accesos indeseados en el archivo de registro de Asterisk. Ahora tenemos que modificar el archivo de configuración de fail2ban

```
# cd /etc/fail2ban
# vim jail.conf
```

y añadimos al final del archivo de texto estas líneas

```
[asterisk-iptables]

enabled = true
filter = asterisk
action = iptables-allports[name=ASTERISK, protocol=all]
        sendmail-whois[name=ASTERISK, dest=root@localhost,
sender=fail2ban@pbx.dyndns.org]
logpath = /var/log/asterisk/full
maxretry = 4
bantime = 18000
```

Para que funcione tenemos que chequear la configuración de los archivos de registro de Asterisk:

```
# vim /etc/asterisk/logger.conf
```

y averiguar que la línea messages tenga el parámetro que aparece en negrita

```
messages => notice, debug
```

Terminamos con la configuración del script de arranque:

```
# cd /usr/src/fail2ban-0.8.4/files/
```

y copiamos el script en el init.d para que corra como servicio

```
# cp redhat-initd /etc/init.d/fail2ban  
# chmod 755 /etc/init.d/fail2ban
```

Usamos chkconfig para que levante fail2ban cada vez que iniciemos nuestro servidor.

```
# chkconfig fail2ban on
```

Iniciamos el servicio

```
# service fail2ban start  
Starting fail2ban: [ OK ]
```

Para probar si nuestro fail2ban tiene bien configurados las expresiones regulares para los correspondientes macheos podemos ejecutar el siguiente comando, donde indicamos el archivo de log a examinar y el archivo de filtro de fail2ban, para asterisk por ejemplo:

```
# fail2ban-regex /var/log/asterisk/full  
/etc/fail2ban/filter.d/asterisk.conf
```

Si aparece algún error tendremos que revisar la configuración.

Port-knocking o Golpeo de Puertos: “Llamar antes de entrar”

Port-Knocking es una simple pero efectiva herramienta que puede aportar mucho a la hora de securizar nuestro servidor.

Para entender su funcionamiento podemos realizar una analogía trayendo a colación una situación muy típica en los años en que regia la ley seca y no se podía consumir alcohol en ningún lado; frente a esto, existían bares clandestinos que a simple vista desde afuera estaban totalmente cerrados y aparentaban estar abandonados para no levantar sospechas, en estas casonas se podía conseguir el preciado alcohol; los hábitos de estos lugares coordinaban “una contraseña” que solo ellos conocían y se traducían en la forma en la que debían golpear la puerta; por ejemplo el típico “Paparapapa papá!!” (nose si sonó como me lo imagino, pero en fin... supongo que se entiende) en ese momento los que estaban adentro se percataban que la persona conocía la combinación y abrían la puerta con gusto diciendo ¡Bienvenido a casa!, si el golpe sonaba de otra manera no accedían a abrir a puerta y de esta forma mantenían su negocio.

De la misma manera (valga la analogía), Port-knocking realiza una determinada acción en nuestro servidor cuando recibe una petición a una combinación específica de puertos definida previamente; por ejemplo, podríamos cerrar el puerto 443 desde Iptables e indicar a Port-Knocking que, cuando realicemos una petición al los puertos 7453, 1874, 5759 respectivamente, nos abra el puerto indicado y de esta forma acceder a nuestra interfaz de login de Elastix solamente desde la IP que estamos realizando la petición; para cerrar el puerto, de la misma forma ingresamos una combinación secuencial por ej: 5759, 1874, 7453 y el mismo se cerrara.

Manos a la obra...

Instalación:

Descargamos el paquete RPM (para plataformas de 32 bits):

```
# wget http://apt.sw.be/redhat/el5/en/i386/rpmsforge/RPMS/knock-0.5-1.el5.rf.i386.rpm
```

Instalamos el paquete

```
# rpm -i knock-0.5-1.el5.rf.i386.rpm
```

Identificamos el archivo “/etc/knockd.conf”, borramos su contenido y agregamos las siguientes líneas para cerrar/abrir el puerto ssh y https quedando de la siguiente manera:

```
[options]
    logfile = /var/log/knockd.log

[openSSH]
    sequence      = 7000,8000,9000
    seq_timeout   = 5
    tcpflags      = syn
    command       = iptables -A INPUT -s %IP% -p tcp --dport 22 -j
ACCEPT

[closeSSH]
    sequence      = 9000,8000,7000
    seq_timeout   = 5
    tcpflags      = syn
    command       = iptables -D INPUT -s %IP% -p tcp --dport 22 -j
ACCEPT

[openHttps]
    sequence      = 7001,8001,9001
    seq_timeout   = 5
    tcpflags      = syn
    command       = iptables -I INPUT -s %IP% -p TCP --dport 443 -j
ACCEPT

[closeHttps]
    sequence      = 9001,8001,7001
    seq_timeout   = 5
    tcpflags      = syn
    command       = iptables -D INPUT -s %IP% -p TCP --dport 443 -j
ACCEPT
```

Ahora ingresamos a /etc/rc.d/init.d y creamos el archivo “knock” y en el mismo copiamos las siguientes líneas para manejar el demonio como un servicio.

```
#!/bin/bash
#
# chkconfig: 345 92 08
# description: Demonio de Knockd
#           http://www.zeroflux.org/projects/knock
```

```
# process name: knockd
#
#
# Author: Rodrigo Martin
#

# Source function library.
. /etc/init.d/functions

# Check that the config file exists
#[ -f /etc/knockd.conf] || exit 0

KNOCKD="/usr/sbin/knockd -d"

RETVAL=0

getpid() {
    pid=` ps -eo pid,comm | grep knockd | awk '{ print $1 }'`
    #echo $pid
}

start() {
    echo -n $"Starting knockd: "
    getpid
    if [ -z "$pid" ]; then
        $KNOCKD start > /dev/null
        RETVAL=$?
    fi
    if [ $RETVAL -eq 0 ]; then
        touch /var/lock/subsys/knockd
        echo_success
    else
        echo_failure
    fi
    echo
    return $RETVAL
}

stop() {
    echo -n $"Stopping knockd: "
    getpid
    RETVAL=$?
    if [ -n "$pid" ]; then
        #$KNOCKD stop > /dev/null
        sleep 1
        getpid
        if [ "$pid" ]; then

            kill "$pid"
        fi
    fi
}
```

```
        rm -f /var/lock/subsys/knockd
        echo_success
    else
        echo_failure
    fi
else
    echo_failure
fi
echo
return $RETVAL
}

# See how we were called.
case "$1" in
    start)
        start
        ;;
    stop)
        stop
        ;;
    status)
        getpid
        if [ -n "$pid" ]; then
            echo "knockd (pid $pid) is running..."
            # $KNOCKD status
        else
            RETVAL=1
            echo "knockd is stopped"
        fi
        ;;
    restart)
        stop
        sleep 2
        start
        ;;
    *)
        echo $"Usage: $0 {start|stop|status|restart}"
        exit 1
        ;;
esac

exit $RETVAL
```

Damos permiso de ejecución al mismo:
chmod 755 /etc/rc.d/init.d/knock

Establecemos que se cargue siempre cuando inicie nuestro servidor

```
# chkconfig knock on
```



Iniciamos el servicio

```
# service knock start
```

Para probar el port-knocking debemos tener cerrados los puertos involucrados por defecto desde Iptables.

Desde Linux en un equipo remoto hacemos el "golpeo de puertos" para abrir por ejemplo el puerto https de la siguiente manera:

```
# telnet ip.del.server.elastix 7001 ; telnet ip.del.server.elastix  
8001 ; telnet ip.del.server.elastix 9001
```

Luego presionamos Ctrl+C 4 veces y nuestro servidor abrirá el puerto indicado para nuestra ip; podemos cerrarlo ingresando la otra combinación.

Si necesitamos "golpear" desde Windows podemos llevarlo a cabo con la aplicación "knock-win32-port" (para 32 bits).

Buenas Prácticas en nuestro Servidor

- Conocer perfectamente el sistema por dentro.
- Llevar un control exhaustivo del sistema.
- Estar al día con las actualizaciones, vulnerabilidades y soluciones.
- Llevar un mantenimiento continuo.
- Observar los logs del sistema.
- Evitar utilizar puertos estándares.
- Utilizar un firewall.
- Utilizar herramientas como fail2ban para evitar escaneos y ataques DoS.
- Denegar peticiones al 5060/4569 UDP desde el exterior siempre que no tengamos usuarios SIP/IAX externos.
- DROPEAR cualquier paquete procedente de cualquier IP que no sea de nuestra red de confianza.

Siete pasos para mejorar la seguridad SIP en Asterisk

En los últimos meses han aparecido una serie de nuevas herramientas que hace posible a cualquier novato atacar y cometer fraudes en equipos SIP, incluyendo los sistemas basados en Asterisk.

Existen herramientas fácilmente disponibles que hacen un barrido de redes en busca de hosts que ofrezcan servicios SIP, luego, una vez encontrado, realiza un barrido en busca de extensiones y contraseñas.

Existen ciertas reglas, de aplicación inmediata, que eliminan muchos de los problemas de seguridad, protegiendo al servidor Asterisk de los barridos masivos y los ataques posteriores. Estos métodos y herramientas de protección ya existen, simplemente hay que aplicarlos.

1) No aceptar pedidos de autenticación SIP desde cualquier dirección IP.

Utilizar las líneas "permit=" y "deny=" de sip.conf para sólo permitir un subconjunto razonable de direcciones IP alcanzar cada usuario/extensión listado en el archivo sip.conf. Aún aceptando llamadas entrantes desde "anywhere" (via [default]) no se debe permitir a esos usuarios alcanzar elementos autenticados.

2) Establecer el valor de la entrada "alwaysauthreject=yes" en el archivo sip.conf. Esta opción está disponible desde la versión 1.2 de Asterisk, pero su por defecto su valor es "no", lo que puede ser potencialmente inseguro. Estableciendo este valor en "yes" se rechazarán los pedidos de autenticación fallidos utilizando nombres de extensiones válidas con la misma información de un rechazo de usuario inexistente. De esta forma no facilitamos la tarea al atacante para detectar nombres de extensiones existentes utilizando técnicas de "fuerza bruta".

3) Utilizar claves SEGURAS para las entidades SIP. Este es probablemente la más importante medida de seguridad. Si alguna vez viste programas que generan y prueban claves por fuerza bruta sabrás que se necesita algo más que palabras y números para una clave segura. Usar símbolos, números, una mezcla de letras minúsculas y mayúsculas y al menos 12 caracteres de largo.

4) Bloquear los puertos del Asterisk Manager Interface. Usar "permit=" y "deny=" en manager.conf para limitar las conexiones entrantes sólo a hosts conocidos. Una vez más utilizar claves seguras aquí también, 12 caracteres al menos en una combinación de números, letras y símbolos.

5) Permitir sólo una o dos llamadas por vez por entidades SIP cuando sea posible. Limitar el uso no autorizado de las líneas voip es una sabia decisión, esto también es útil para el caso que usuarios legítimos hagan pública su clave y pierdan control de su uso.

6) Los nombres de usuarios SIP deben ser diferentes que sus extensiones. A pesar de ser conveniente tener una extensión "1234" que mapee a una entrada SIP "1234" la cual es también el usuario SIP "1234", esto también facilita a los atacantes para descubrir nombres de autenticación SIP. En su lugar usar las direcciones MAC del dispositivo, o alguna combinación de frases comunes + extensión MD5 hash (por ejemplo: desde el shell prompt, hacer "md5 -s ThePassword5000")

7) Asegurarse que el contexto [default] sea seguro. No permitir que llamadores no autenticados alcancen contextos que les permitan llamar. Permitir sólo una cantidad limitada de llamadas activas pasen por el contexto default (utilizar la función "GROUP" como contador). Prohibir totalmente las llamadas no autenticadas (si es que así lo queremos) estableciendo "allowguest=no" en la parte [general] de sip.conf.

Backdoor detectado en FreePBX (hasta la versión 2.0.1 de Elastix - Agosto 2010)

Reseña

En el mes de agosto de 2010, Andrés Babativa, miembro de lista general de Elastix descubrió que uno de los usuarios definidos para acceder a la base de datos, podía acceder como usuario Admin por el front-end de freepbx; mas alla de que nos hayamos esmerado muchísimo en usuarios y contraseñas fuertes, si vengo con el "gold super user" entro sin ningún problema con permisos de admin y puedo hacer lo que quiero en cualquier asterisk que tenga un freepbx!.

Para verificar esto podemos acceder a la interfaz de freepbx, cuando nos solicite user y passowrd ingresamos:

```
asteriskuser/eLaStIx.asteriskuser.2oo7
```

Frente a esto ese mismo día, horas más tarde el equipo de desarrolladores de Palosanto Solutions ya tenía listo el paquete RPM de freepbx corregido en los repositorios de Elastix.

Para aplicarlo debemos actualizar freepbx; para esto ejecutamos en la consola como usuario root:

```
yum upgrade freePBX*
```

Una vez actualizado, podemos corroborar que el problema está solucionado ingresando el usuario y password anteriormente mencionados y notaremos que ya no podremos acceder.

Fuentes:

Comunicaciones Unificadas con Elastix, vol. 1 y 2 – Edgar Landivar

Elastix a Ritmo de Merengue – Alfio Muñoz

wiki.centos.org/es/HowTos/Network/SecuringSSH

blogs.digium.com/2009/03/28/sip-security/

www.fail2ban.org

www.sinologic.net

blogs.elastix.org/es/2010/01/14/instalacion-de-fail2ban/

www.voztovoice.org/?q=node/135

<http://www.zeroflux.org/projects/knock>

<http://elmampano.wordpress.com/2011/05/24/port-knocking/>