How to Secure RHEL 6.2
Part 3

Motivation

This paper is part of a multi-part series on securing Redhat Enterprise Linux 6.2. This paper focuses on implementing Secure Shell and PKI certificates. The intended audience for this paper is laymen and journeymen system administrators that need a quick reference and solid solutions. If you have comments or feedback on how I can represent this better, please email me your ideas to the email address listed on the website (securityhardening.com).

Secure Shell (SSH)

Installation

Use either YUM or RPM to install the openssh server and client (assuming they are not installed). Mount the DVD RHEL 6.2 distribution and change directory to the Package directory.

```
# mount /media/cdrom
# cd /media/cdrom/Packages
# yum install openssh-server-5.3p1-70.el6.x86_64.rpm \
              openssh-5.3p1-70.el6.x86_64.rpm \
              openssh-clients-5.3p1-70.el6.x86_64.rpm
```

Configuration

Change your sshd_config and ssh_config to be more secure:

Zero out the existing configs:

```
# cat /dev/null > /etc/ssh/ssh_config
# cat /dev/null > /etc/ssh/sshd_config
```

Copy the following into their respective files:

/etc/ssh/ssh_config:

```
#       $OpenBSD: ssh_config,v 1.25 2009/02/17 01:28:32 djm Exp $

# This is the ssh client system-wide configuration file.  See
# ssh_config(5) for more information.  This file provides defaults for
# users, and the values can be changed in per-user configuration files
# or on the command line.

# Configuration data is parsed as follows:
#  1. command line options
#  2. user-specific file
#  3. system-wide file
```

```
# Any configuration value is only changed the first time it is set.
# Thus, host-specific definitions should be at the beginning of the
# configuration file, and defaults at the end.

# Site-wide defaults for some commonly used options.  For a comprehensive
# list of available options, their meanings and defaults, please see the
# ssh_config(5) man page.

# Host *
ForwardAgent no
ForwardX11 no
#   RhostsRSAAuthentication no
#   RSAAuthentication yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   GSSAPIAuthentication no
#   GSSAPIDelegateCredentials no
#   GSSAPIKeyExchange no
#   GSSAPITrustDNS no
#   BatchMode no
#   CheckHostIP yes
#   AddressFamily any
#   ConnectTimeout 0
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/identity
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
Port 22
Protocol 2
Cipher blowfish
Ciphers aes256-ctr,aes192-ctr,aes128-ctr,aes128-
cbc,arcfour256,arcfour128,3des-cbc
MACs hmac-ripemd160,hmac-sha1,hmac-md5,umac-64@openssh.com
#   EscapeChar ~
Tunnel no
#   TunnelDevice any:any
#   PermitLocalCommand no
#   VisualHostKey no
Host *
        GSSAPIAuthentication yes
        ForwardX11Trusted yes
# Send locale-related environment variables
        SendEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY
LC_MESSAGES
        SendEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
        SendEnv LC_IDENTIFICATION LC_ALL LANGUAGE
        SendEnv XMODIFIERS
```

/etc/ssh/sshd_config:

```
#       $OpenBSD: sshd_config,v 1.80 2008/07/02 02:24:18 djm Exp $
#
Port 22
ListenAddress 192.168.139.88
Protocol 2
#
AllowUsers masterf
#
```

```
# HostKeys for protocol version 2
HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key
#
# Lifetime and size of ephemeral version 1 server key
KeyRegenerationInterval 15m
ServerKeyBits 2048
#
# Logging
SyslogFacility AUTHPRIV
LogLevel DEBUG
#
# Authentication:
LoginGraceTime 45
PermitRootLogin no
StrictModes yes
MaxAuthTries 6
MaxSessions 10
#
#RSAAuthentication yes
#PubkeyAuthentication yes
#AuthorizedKeysFile     .ssh/authorized_keys
#AuthorizedKeysCommand none
#AuthorizedKeysCommandRunAs nobody
#
# For this to work you will also need host keys in /etc/ssh/ssh_known_hosts
RhostsRSAAuthentication no
# similar for protocol version 2
HostbasedAuthentication no
# Change to yes if you don't trust ~/.ssh/known_hosts for
RhostsRSAAuthentication no
#IgnoreUserKnownHosts no
# Don't read the user's ~/.rhosts and ~/.shosts files
IgnoreRhosts yes
#
# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
PermitEmptyPasswords no
PasswordAuthentication yes
#
# Change to no to disable s/key passwords
#ChallengeResponseAuthentication yes
ChallengeResponseAuthentication no
#
# Kerberos options
#KerberosAuthentication no
#KerberosOrLocalPasswd yes
#KerberosTicketCleanup yes
#KerberosGetAFSToken no
#KerberosUseKuserok yes
#
# GSSAPI options
#GSSAPIAuthentication no
#GSSAPICleanupCredentials yes
#GSSAPIStrictAcceptorCheck yes
#GSSAPIKeyExchange no
#
```

```
# Set this to 'yes' to enable PAM authentication, account processing,
# and session processing. If this is enabled, PAM authentication will
# be allowed through the ChallengeResponseAuthentication and
# PasswordAuthentication.  Depending on your PAM configuration,
# PAM authentication via ChallengeResponseAuthentication may bypass
# the setting of "PermitRootLogin without-password".
# If you just want the PAM account and session checks to run without
# PAM authentication, then enable this but set PasswordAuthentication
# and ChallengeResponseAuthentication to 'no'.
UsePAM yes
#
# Accept locale-related environment variables
AcceptEnv LANG LC_CTYPE LC_NUMERIC LC_TIME LC_COLLATE LC_MONETARY LC_MESSAGES
AcceptEnv LC_PAPER LC_NAME LC_ADDRESS LC_TELEPHONE LC_MEASUREMENT
AcceptEnv LC_IDENTIFICATION LC_ALL LANGUAGE
AcceptEnv XMODIFIERS
#
AllowAgentForwarding yes
AllowTcpForwarding no
GatewayPorts no
X11Forwarding yes
X11DisplayOffset 10
X11UseLocalhost yes
PrintMotd no
PrintLastLog no
TCPKeepAlive yes
UseLogin no
UsePrivilegeSeparation yes
PermitUserEnvironment no
Compression delayed
ClientAliveInterval 300
ClientAliveCountMax 0
ShowPatchLevel no
UseDNS no
PidFile /var/run/sshd.pid
MaxStartups 10
PermitTunnel no
ChrootDirectory none
#
# no default banner path
Banner /etc/issue.net
#
# override default of no subsystems
Subsystem       sftp    /usr/libexec/openssh/sftp-server
```

Restart the ssh service so the configs are used:

```
[root@secure62 ~]# service sshd restart
Stopping sshd:                                             [  OK  ]
Starting sshd:                                             [  OK  ]
[root@secure62 ~]#
```

Notice the line in sshd_config with 'AllowUsers masterf'. This allows only this account to authenticate. Change the line to meet your sites list of authorized users.

Key Generation

As your user id, e.g. masterf (my little fun as Master Foo), run:

```
[masterf@secure62 ~]$ ssh-keygen -b 2048 -C "from `hostname` for ${USER}" -M
128 -t rsa -v
Generating public/private rsa key pair.
Enter file in which to save the key (/home/masterf/.ssh/id_rsa):
Created directory '/home/masterf/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/masterf/.ssh/id_rsa.
Your public key has been saved in /home/masterf/.ssh/id_rsa.pub.
The key fingerprint is:
73:d9:91:3d:18:a7:87:4e:08:23:89:b3:e3:e2:87:35 from secure62.fortress.lan
for masterf
The key's randomart image is:
+--[ RSA 2048]----+
|     ...o   . .  |
|    o .. o . O   |
|     o    . B +  |
|    o      = o . |
|   . .  S o o    |
|  . E    o       |
| . + .           |
|  o .            |
|   .             |
+-----------------+
[masterf@secure62 ~]$
```

Key Distribution

Push your key to all servers you will need to connect to (e.g. one server):

```
[masterf@secure62 ~]$ ssh-copy-id -i /home/$USER/.ssh/id_rsa \
                                    masterf@192.168.139.75
```

Wait, what if you had an entire subnet you need to push these keys to:

```
[masterf@secure62 ~]$ for X in `seq 2 254`
                      do
                          ssh-copy-id -i /home/$USER/.ssh/id_rsa \
                                       masterf@192.168.139.$X
                      done
```

Or you are on IP address with the last octet, .88:

```
[masterf@secure62 ~]$ for X in `seq 2 87; seq 89 254`
                      do
                          ssh-copy-id -i /home/$USER/.ssh/id_rsa \
                                       masterf@192.168.139.$X
                      done
```

Testing

On the local server, test with this command:

```
[masterf@secure62 ~]$ ssh masterf@192.168.139.88
```

Usage Examples

Connecting to a remote server with ssh:

```
[masterf@secure62 ~]$ ssh masterf@192.168.139.75
```

Connecting to all servers sequentially to perform some task:

```
[masterf@secure62 ~]$ for X in `seq 2 87; seq 89 254`
                      do
                          ssh masterf@192.168.139.${X}
                      done
```

Execute a single command on remote server:

```
[masterf@secure62 ~]$ ssh masterf@192.168.139.75 "hostname"
```

Execute a single command on all of your servers:

```
[masterf@secure62 ~]$ for X in `seq 2 87; seq 89 254`
                      do
                          ssh masterf@192.168.139.${X} "hostname"
                      done
```

Public Key Infrastructure (PKI)

Installation

I prefer using the Java keytool to generate my PKI keys because most of the systems I support have java 1.6 or 1.7 on them. Also, this block of instruction assumes you are going to generate a key pair, then a Certificate Signing Request (CSR) to send to a valid Certificate Authority (CA) and how to import that script back into your keystore.

Generate your keystore:

```
#!/usr/bin/env bash

PATH=$PATH:/opt/java/java/bin
PASSWORD_OUTPUT=./.cert.passwords
HOSTNAME=$( hostname )
MACADDR=$( /sbin/ifconfig | grep HWaddr | awk '{print $5}' )
CURRDATE=$( date +%Y%m%d-%H:%M:%S:%N )
PASSWORD=$( echo ${HOSTNAME} ${MACADDR} ${CURRDATE} | /usr/bin/md5sum | \
            cut -b 3-14 | tr 'a-z' 'A-Z' )

printf "\n\n\nStarted on $(date +%Y%m%d-%H:%M:%S)\n" >> ${PASSWORD_OUTPUT}

##
#       First generate the keypair.
keytool -genkeypair -v -alias ${HOSTNAME} -keyalg RSA -sigalg SHA1withRSA \
```

```
        -dname "CN=${HOSTNAME},OU=Other,OU=PKI,O=Your Company,C=US" \
        -keysize 2048 -keypass ${PASSWORD} \
        -validity 1095 -keystore ${HOSTNAME}.keystore.jks \
        -storepass ${PASSWORD}

##
#       Now generate the CSR.
keytool -certreq -alias ${HOSTNAME} -keystore ${HOSTNAME}.keystore.jks \
        -storepass ${PASSWORD} -keypass ${PASSWORD} -file ${HOSTNAME}.csr

##
#       Now print the password.
printf "Hostname is:\t${HOSTNAME}\t\tThe password is:\t${PASSWORD}\n" \
        >> ${PASSWORD_OUTPUT}

printf "\n\n Finished on $(date +%Y%m%d-%H:%M:%S)\n" >> ${PASSWORD_OUTPUT}

exit 0
```

Email or submit to the web portal for your CA the CSR. Example format:

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIC2zCCAcMCAQAwZjELMAkGA1UEBhMCVVMxFTATBgNVBAoTDFlvdXIgQ29tcGFueTEMMAoGA1UE
CxMDUEtJMQ4wDAYDVQQLEwVPdGhlcjEiMCAGA1UEAxMZcmhlbDYyYXBwbjAxLmZvcnRyZXNzLmxh
bjCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBALgr+P4eiE4QGVtGxFVepbxz9y8e1MIy
XqCTbub0O+qjF941xwSLzPNPr1/2kq2bEhIZ7AK3/POFj0fU2NXHzcO7x9nk5OIycc73yKLtpe6G
7QNM6FF9d6H8T0jRA7b+ihF97z68ASXuKE8VzAJn9D/gfrWRyJ+S4+pjlLS8a+6qsJVkqyIrgi1O
J0BWzvA1UzX+k7Z5C2BA/9fHLhggDhNTvJHb2vKlmx9G3vafxXpvWVXuOpWz1siX6FLte4TIsgq9
qzryLoZHerqyUoAGfvOMzvppPiNrUS9wNsrzYcyytkm7qy8NfjQSvZ/mOUnQFKzusMdIJvNfhXqU
KfVDFfECAwEAAaAwMC4GCSqGSIb3DQEJDjEhMB8wHQYDVR0OBBYEFHdCmUJyP18wjP2DTQ5U+DKW
KGIQMA0GCSqGSIb3DQEBCwUAA4IBAQCLqcRPqTSBND2atktXFWQ0vq2RE3zANBpAQIvOl/TbEMrO
ChSykGhKYhJT8UEbc48SvFztjHgSsTht5PYe/wuV0rNXwpbE31/5E8W4VbV10JmiAXHNlBJciojS
K8BNhXt6UylD4EVYhdSV0ykZuddY6jRPykNbbNaU1++/IIjNcuOzn++yGibbfVY/LXpVW9zRPDNc
S2T7CqitdLJaqDH6TH6V7DEbSW/dHdzA/ktPEdKxmJtbXxMLsm431ujjVzYwtLU9wYPUMSJWPsik
ixp4VIHOBOMrcZrSVSTEdX6n/yme92/Hkhn8jPfMg6whEDh75N0M84rRzqceiGL37Msr
-----END NEW CERTIFICATE REQUEST-----
```

Once signed, copy the base64 signed request with root chain into a temporary file, e.g. $HOSTNAME.signed.pem and run:

```
keytool -import -alias ${HOSTNAME} -file ${HOSTNAME}.signed.pem \
        -keystore ${HOSTNAME}.keystore.jks
```

Configuration

Assuming you are using a web application server like Tomcat, I would configure my server.xml file as:

```
<?xml version='1.0' encoding='utf-8'?>
<Server port="8005" shutdown="SHUTDOWN">
  <Listener className="org.apache.catalina.core.AprLifecycleListener"
SSLEngine="on" />
  <Listener className="org.apache.catalina.core.JasperListener" />
  <Listener
className="org.apache.catalina.core.JreMemoryLeakPreventionListener" />
  <Listener
className="org.apache.catalina.mbeans.GlobalResourcesLifecycleListener" />
```

```
  <Listener
className="org.apache.catalina.core.ThreadLocalLeakPreventionListener" />

  <GlobalNamingResources>
    <Resource name="UserDatabase" auth="Container"
              type="org.apache.catalina.UserDatabase"
              description="User database that can be updated and saved"
              factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
              pathname="conf/tomcat-users.xml" />
  </GlobalNamingResources>
  <Service name="Catalina">

    <Connector port="8080" protocol="HTTP/1.1"
               connectionTimeout="20000"
               maxThreads="200"
               redirectPort="8443" />

    <Connector port="8443"
               maxHttpHeaderSize="8192"
               minSpareThreads="25"
               maxSpareThreads="75"
               enableLookups="false"
               disableUploadTimeout="true"
               protocol="HTTP/1.1"
               maxThreads="150"
               scheme="https"
               secure="true"
               keystoreFile="certs/secure62.keystore.jks"
               keystorePass="BD35D8435251"
               keyAlias="secure62.fortress.lan"
               truststoreFile="certs/secure62.keystore.jks"
               truststorePass="BD35D8435251"
               SSLEnabled="true"
               clientAuth="false"
               sslProtocol="TLS" />

    <Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />

    <Engine name="Catalina" defaultHost="secure62.fortress.lan">

      <Realm className="org.apache.catalina.realm.LockOutRealm">
        <Realm className="org.apache.catalina.realm.UserDatabaseRealm"
               resourceName="UserDatabase"/>
      </Realm>

      <Host name="secure62.fortress.lan"  appBase="webapps"
            unpackWARs="true" autoDeploy="true">

        <Valve className="org.apache.catalina.valves.AccessLogValve"
directory="logs"
               prefix="localhost_access_log." suffix=".txt"
               pattern="%h %l %u %t &amp;quot;%r&amp;quot; %s %b" />

      </Host>
    </Engine>
  </Service>
</Server>
```

Modify the yellow sections to fit your environment.

Testing

Restart the tomcat web server and use your favorite browser to view the web page. Make sure you use the https protocol, e.g. "https://192.168.139.88:8443/ROOT/favicon.ico".

Conclusion

Both of these protocols are strong, but the real strength here is that the information passing over the network is now encrypted. It would probably take 500+ years to break the cryptographic strength of AES with a key size of 2048. SSH should be used to manage the systems, while the secure web pages will keep user accounts encrypted over the unsecured network. In today's age, there is absolutely no reason to communicate over un-secured networks. Thinking that someone is not tapping your communication channels is un-realistic.