SSH Certificate Authentication

Background:

SSH Certificate Authentication is an alternate means (compared to just key pair authentication) of authenticating ssh sessions which has the following benefits over just key pairs:

1. ***More centralized administration*** – key pairs don't have to be placed in ~/.ssh/authorized_keys of all systems where key-based authentication is desired.
2. If host certificates are used, ***eliminate the need to have to accept the host's public key*** on first use. This eliminates having to distribute lists of valid host keys or having the bad practice of "just accepting the key without checking". What host keys are valid is controlled by what host keys are signed.
3. ***Validity date range*** – a certificate can have a future start date and/or future end date. **Note:** for this and the next feature to be effective, key-based authentication must be required (either by itself or in combination with another required method if two-factor authentication is used) by using AuthenticationMethods in sshd-config. If other options are available then successful login via those options is possible after certificate authentication fails.
4. ***Revocation*** – a certificate can be revoked.
5. Allow specifying a ***list*** of authorized hosts or users ("principals").

Per https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-creating_ssh_ca_certificate_signing-keys, this feature was introduced in RHEL 6 with (apparently) openssh 5.3, possibly later with other distributions.

Point of clarification: SSH certificates are ***not*** SSL (X509) certificates as are used on a web server, they are simply signed SSH public keys created using ssh-keygen (see later). All of this is documented in the man pages for ssh, sshd, sshd_config and ssh-keygen. The issue is that the scattered references are not coherently explained. Web tutorials can be found if someone knows to look. There does appear to be an X509 implementation for ssh (http://tech.ciges.net/blog/openssh-with-x509-certificates-how-to/) but requires special software and is out-of-scope for this presentation.

As an aside, there are other restrictions which can be implemented by either (1) certificates or (2) authorized_keys or (3) sshd_config (as well as a few that are only available from one source):

1. Agent forwarding (1), (2)
2. Restriction of commands run - "force-command" (1), "command" (2), "ForceCommand" (3). The parameter to these entries ***can be*** a script allowing a large degree of customization. (see later)
3. terminal allocation (PTY) - (1), (2). This needs to be properly understood, not allocating a PTY does prevent such things as changing a password but does ***not*** prevent such things as "ls -al" - the results will be returned even though there is no command prompt.
4. Source IP address – "source-address" (1), "from" (2)
5. Port forwarding (1), (2)
6. Use of ~/.ssh/rc - (1), (2)
7. X11-forwarding (1), (2)

The process for host authentication:

(Consider doing the below on a system dedicated for this purpose.) Create a CA for host signing (recommend using separate “CA”s for host and user signing):

`ssh-keygen -f host_ca` This creates files host_ca and host_ca.pub (pick whatever name you want). It does not put them in ~/.ssh but in the current working directory, it might be desirable to specify the encryption method with -t and a comment with -C. Note that this key pair is only special because it is used for signing, otherwise it is like any other key pair.

Sign the server's host key with the host CA's private key

```
ssh-keygen -s host_ca -I host_ID -h /etc/ssh/ssh_host_rsa_key.pub
```

- where host_ca (or whatever name you use) is the ssh CA private key for hosts
- the -I parameter is an ID and should probably be more descriptive
- /etc/ssh/ssh_host_rsa_key.pub is the pre-existing public host key.

edit /etc/ssh/sshd_config Add the following line

```
HostCertificate /etc/ssh/ssh_host_rsa_key-cert.pub
```

- Testing surfaced the fact that the “correct” host key (dsa, ecdsa, ed25519 or rsa) had to be used or certificate-based host authentication failed. The “correct” key refers to the host key which is actually supplied by sshd in response to a connection request. During testing, using “ssh -vv ...” showed that the ecdsa key was offered. It may be possible to discern which key will be offered by looking at the list under HostKeyAlgorithms in the sshd_config man page (if the order listed is the order offered).
- This can be controlled by:
  - removing the comment from a (or using only one) HostKey specification and using the corresponding encryption type in HostCertificate.
  - Adding HostKeyAlgorithms to sshd_config and supplying the desired order as a parameter – the man page for sshd-config lists the algorithms under the HostKeyAlgorithms entry. This is the method (arbitrarily) being used for the presentation just to demonstrate its use.

Restart sshd

On a client system **which has not yet accessed the ssh server**, edit ~/.ssh/known_hosts and add the following (note that it could also be placed in /etc/ssh/known_hosts to apply to all users):
@cert-authority <DNS name or IP specification of target system> <encryption method such as ssh-rsa> <the contents of the host CA's public key excluding the {user@system} part>

- Note: “IP specification” can be an actual IP address or subnet but the subnet **MUST** be specified using asterisks or question marks rather than CIDR format. For example, 192.168.1.* works, 192.168.1.0/24 does not. work.

test the result should be that no prompt for accepting the host key is delivered.

The process for user authentication:

If you want a key of a specific length, use the -b option. However, https://ef.gy/hardening-ssh states that keys longer than 8192 may break some implementations.

```
ssh-keygen -f user_ca
```

```
cp -p user_ca.pub /etc/ssh/
```

This will need to be done for each system using this "CA" for authentication. The private key needs to be appropriately secured somewhere.

```
edit /etc/ssh/sshd_config
```

Add the following line(s)

```
TrustedUserCAKeys /etc/ssh/user_ca.pub
```

(optionally, if it's available on the server –, openssh 7.4 doesn't have it) add `ExposeAuthInfo yes` to set a per-user variable (SSH_USER_AUTH) which will point to a file containing the certificate's information. From testing, the file is placed in /tmp and named sshauth.<arbitrary generated string>. It's contents are the user's certificate (id_rsa-cert.pub) with the string "publickey " prepended to it. In order to be useful the contents must be edited and "publickey " removed. Once that is done then `ssh-keygen -L -f <file containing the contents>` will display useful information. Once the user disconnects the file is removed.

(create a user key pair or get a copy of a user's existing public key. For security it is recommended that the private key of newly generated pairs not be kept thus reducing the sources of them.)

```
ssh-keygen -s user_ca -I user_<username> -n <username> -V +52w id_rsa.pub
```

- Change user_ca to the name of the "CA" private key.
- The -I parameter is simply an identity but very important, it shows up in ssh's authentication log (/var/log/auth.log for Ubuntu) and can thus be used to easily identify who authenticated – make it unambiguous.
- The -n parameter specifies the name of the user (on the target system) **if** you want to restrict it that way.
- This -V parameter provides a 52 week validity from the current date, as little as one day has been tested. Use "ssh-keygen -L -f <signed key>" to show the details.

copy the signed public key (and the private key if this is a new key pair rather than an existing key pair) to the user's .ssh directory

restart the ssh server

test

Note: a user without a signed public key but with ~/.ssh/authorized_keys on the server containing their public key will still be able to authenticate.

Revoking a key:

What if a user's signed public key (or their private key) is compromised (or something happens to a host certificate)? Key revocation is far more efficient than having to scan the users' ~/.ssh/authorized_keys file on a large number of servers and remove the compromised public key.

```
ssh-keygen -k -f /etc/ssh/revoked-user-keys.krl -s user_ca <specification>
```

- where <specification> can be a user certificate, certificate ID or other items – read the manual

edit /etc/ssh/sshd_config   Add the following line
RevokedKeys /etc/ssh/revoked-user-keys.krl

restart sshd

test   the result should be something to the effect of prompting for a password (if that is also an acceptable method) “no such identity” or “Permission denied (publickey)”. Fortunately, testing indicates that a revoked certificate takes precedence over ~/.ssh/authorized_keys (having the user's public key in this file doesn't allow access when the certificate is revoked).

Issues:

1. Having an id_rsa-cert.pub file in ~/.ssh may lead to error messages when connecting to non-certificate-based ssh servers or blocking access if only public key authentication is allowed. If password authentication is also allowed this is just an annoyance, if only public key authentication is allowed then a careful roll out plan will be needed.
2. Using an ssh CA ***could*** mean that all servers trusting it will allow all users who have had their public key signed by it (and have a login listed in principals) to login as one of the listed principals. In order to have segregation, two known options are:
   (a) Use an AuthorizedPrincipalsFile (an sshd_config keyword). The file(s) specified would contain a list of principals allowed to login. The user certificates would no longer be signed with usernames as principals but rather with the names of principals contained in the AuthorizedPrincipalsFile. The names do not have to exist as users on the system (but an actual user does have to be specified on the client), the only requirement is that the principal listed in the certificate matches a name in the AuthorizedPrincipalsFile. This would allow pseudo-groups to be created. For example, certificates could be signed with a “web-admin” principal, all web servers would list “web-admin” as a principal in the AuthorizedPrincipalsFile, non-web-servers wouldn't thus providing some segregation. Note that this file can be relative to each users home directory (among other things). This would mean that even a user having a certificate with the correct principal name would not be able to login if the AuthorizedPrincipalsFile in their home directory didn't contain the name of the principal. Another implication is that a user could have a different login name on different systems. This gets quite complex and should be thoroughly tested before rollout.
   (b) Multiple CAs. Two known methods to accommodate this are:
      i. The TrustedUserCAKeys file on ssh servers can contain multiple CA public keys.
      ii. Users can have their id_rsa.pub signed by multiple CAs. If this is done then id_rsa-cert.pub must be renamed to something unique before being given to the user and instructions given to the user concerning configuring ~/.ssh/config to use different certificate files for different servers. The configuration is as follows:
```
Host <specification 1>
    CertificateFile ~/.ssh/<certificate file for host 1>
Host <specification 2>
    CertificateFile ~/.ssh/<certificate file for host 2>
```
3. Although having to replicate a CA public key and associated configuration to a number of servers is time consuming, it is hopefully an infrequent effort at worst. Using a key revocation list will likely require much more frequent updating. Two known methods of addressing this are:
   (a) Use of a software distribution/configuration management system such as Ansible, Chef or Puppet. This too is a significant undertaking but may already be in place in larger

organizations. There are also security risks associate with this class of system.

   (b) Use of a network-based filesystem such as nfs, samba, sshfs or OpenAFS to share a "master" key revocation list (and CA public key if desired). However, network-based filesystems introduce their own set of considerations such as availability, bandwidth and vulnerability issues.

4. Centralizing anything comes with the risk that, if the central infrastructure is compromised, the impact is far greater than an individual system breach.
5. Non-default software also has to be considered: What about Cygwin, Putty, WinSCP, MacOS and other software which utilizes ssh? Do they contain the certificate-based authentication capabilities?

Addendum: using a "command" variant to limit what commands an ssh user can execute

This was referred to in the "aside" above. The below script is an example which prevents the use of scp while also gathering some information. The environment variables listed below (and others in addition) are documented in "man ssh". Of interest may be what is contained in SSH_ORIGINAL_COMMAND when scp or rsync is used. The reason for the empty string (second case "block") is that ssh itself sends an empty string in SSH_ORIGINAL_COMMAND.

```
#/bin/bash
/bin/echo "USER: $USER, SSH_CONNECTION: $SSH_CONNECTION,
SSH_ORIGINAL_COMMAND: $SSH_ORIGINAL_COMMAND, cert-path: $SSH_USER_AUTH"
>> /etc/ssh/cmd-hist
/bin/cat $SSH_USER_AUTH >> /etc/ssh/cmd-hist
base_cmd=`/bin/echo "$SSH_ORIGINAL_COMMAND" | /usr/bin/cut -d' ' -f 1`
case $base_cmd in
   "scp")
     /bin/echo "scp not allowed"
     exit 1
     ;;
   "")
     /bin/bash
     ;;
   *)
     $SSH_ORIGINAL_COMMAND
     ;;
esac
```