Two-factor time based (TOTP) SSH authentication with pam_oath and Google Authenticator

Two-factor authentication (2FA) is becoming an increasingly useful way of providing an extra layer of security to services above and beyond passwords.

OATH is an open mechanism for generating either event-based or time-based One Time Passwords and there are a number of hardware tokens and software implementations available, which makes it ideal for a small scale implementation without requiring lots of infrastructure or expense.

Setting up a simple trial to add 2FA to a remote access server using Google Authenticator as a software token, I thought it would be useful to document the bits that I glued together.

These instructions are for RHEL/CentOS 6, and you'll need the EPEL repo for the oath packages (or install the packages and their dependencies directly). pam_oath (and its documentation) is available directly or it might be provided by your OS distribution, if you're not using RHEL/CentOS.

You should leave a session logged in as root while you test this, in case you break anything and need to undo it.

Install the relevant packages, and symlink the pam_oath module into the right place:

```
# yum install pam_oath oathtool
# ln -s /lib64/security/pam_oath.so
/usr/lib64/security/pam_oath.so
```

Enable ChallengeResponse auth in /etc/ssh/sshd_config:

```
ChallengeResponseAuthentication yes
PasswordAuthentication no
UsePAM yes
```

and restart sshd:

```
# service sshd restart
```

If you're using a software token, you'll want to generate a random seed. A good way to generate a random string of an appropriate size and format:

```
# head -10 /dev/urandom | md5sum | cut -b 1-30
15ad027b56c81672214f4659ffb432
```

Set up your oath seed in /etc/users.oath:

```
HOTP/T30/6  yourusername    -   15ad027b56c81672214f4659ffb432
```

You can add as many users as you need, one line at a time. You should also secure that file appropriately, as these strings are effectively a password:

```
# chmod 600 /etc/users.oath
# chown root /etc/users.oath
```

You can generate an OTP using oathtool. Run this with the -v option and your chosen key. The Base32 version of the secret is the one that you will need for the Google Authenticator smartphone app. You can type this in, or generate a QR code later...

```
#  oathtool --totp -v 15ad027b56c81672214f4659ffb432
Hex secret: 15ad027b56c81672214f4659ffb432
Base32 secret: CWWQE62WZALHEIKPIZM77NBS
...
960776
```

Since you probably don't want OTP enabled all the time for all users, create /etc/security/access-local.conf - you can set differing options depending on your requirements.

This configuration would allow access without requiring an OTP from a trusted network:

```
+ : ALL : 192.168.0.0/24
+ : ALL : LOCAL
- : ALL : ALL
```

This configuration only requires OTP for members of the 'otpusers' unix group. This might be useful to selectively 2FA enable user accounts as part of a gradual rollout, or you might decide to only require 2FA for users who have permission to su to root.

```
+ : ALL : LOCAL
- : (otpusers) : ALL
+ : ALL : ALL
```

You can be quite creative with these rules - they follow the standard pam_access syntax, so check the documentation for that.

Finally, I added the following lines to /etc/pam.d/system-auth-ac and /etc/pam.d/password-auth-ac (This is a RHEL/CentOS-ism) - where you put them will depend on your pam configuration and OS. The pam_access entry is optional, but it does make the above choices possible.

```
auth [success=1 default=ignore] pam_access.so accessfile=/etc/se-
curity/access-local.conf
auth required pam_oath.so usersfile=/etc/users.oath window=30
```

Now you can ssh into your server (don't close the root session you currently have open in case you've broken something!). You can generate your OTP using oathtool:

```
# oathtool --totp 15ad027b56c8167214f4659ffb432
329770
```

Log in quickly (before that token expires), and you should find it lets you in:

```
username@host:~$ ssh securehost
Password:
One-time password (OATH) for `username':
Last login: Wed Jul 10 22:38:53 2013 from somehost.example.com
username@securehost:~$
```

To set up the Google Authenticator smartphone app, you can take your Base32 formatted secret, and either enter it manually or generate a QR code. To make a QR code, you

need a URL formatted string, as below. The example of 'username@securehost' is a simple description, so it can be anything you like.

```
otpauth://totp/username@securehost?secret=CWWQE62WZAL-
HEIKPIZM77NBS
```

Feed this into a QR code generator that you trust (remember, this is effectively a password), and scan the code using the app.

With the secret saved into your smartphone app you should now be able to log in using the codes that it generates.

Extra things

The /etc/users.oath file gets updated every time you log in, which can make this a challenge to manage centrally across multiple hosts. It is possible to update this with a custom augeas lens if you're using puppet. I've also got an ANSI escape commandline QR code/seed generator. These are a bit of a bodge, but do seem to work. If there's demand I'll see about sticking a copy of them and the relevant puppet manifest up somewhere.

Contact: ben@spod.cx