

FreeBSD, I don't know

Version 0.2

By Gerrit Seré

March 8, 2021

Contents

1	Installing BSD	3
1.1	Configuration OpenBSD in Virtualbox	3
1.2	Instaling OpenBSD in VirtualBox.	8
1.3	Installing multiple OpenBSD virtual servers	13
2	SSH Certificate Authorities (CA)	16
2.1	Introdction	16
2.2	Standard way to login into server	16
2.2.1	With a password	17
2.2.2	Private and public key	17
2.3	Recap John actions	18
2.4	Using a CA with SSH	19
2.5	Configuring the Certificate Authority	19
2.6	Host Certificate Authority	20
2.7	User Certificate Authority	23
2.8	Logging in as root	25
2.8.1	With principals	25
2.8.2	With AuthorizedPrincipalsFile	26
2.9	With AuthorizedPrincipalsCommand	27
2.10	Fine tuning privileges	29
2.10.1	With principals	30
2.10.2	With AuthorizedPrincipalsFile	31
2.10.3	With AuthorizedPrincipalsCommand	32
2.11	Revoking keys	32
2.11.1	Revoking Users keys	32
2.11.2	Revoking hosts keys	36
2.12	Disable authorized_keys	36
2.13	Conclusion	36
2.14	Resources	36
3	PostgreSql	38
4	zfs snaphost	39

Who am I and why?

I started my sysadmin unix career in 1998 with the Flemish Government. The operation system we were using was Solaris 2.6. In those days, for each new application we bought a physical server, mostly a Sun Ultra 450 Enterprise. A lot of them had a tape device for doing backups. To monitor those servers and deamons we used Big Brother. In our main department we had a NIS+ system. But most of our time we did basic sysadmin, creating users, installing printer queus, updating DHCP configuration, NFS, restarting deamons, checking logs, rotate them, backup, restore and sent mails to people to please cleaup up their home folders. There was no ticket system, we used only mail and hang on the telephone for hours.

First, we used a terminal server. Later a few of my colleagues switched to Linux. I tried using FreeBSD for a few weeks, a lot of time went into compiling, especially KDE took a few hours. FreeBSD was very stable but a few programs were not available, switching me back to Linux.

Nowadays, in 2019, everybody in our team use a laptop. Most of them use Mac Os or Fedora. At the server side everything changed. Physical servers transformed into virtual servers and those servers are running in a Cloud environment. Solaris OS changed to Ubuntu and CentOS.

In April 2018, I was at very small conference (LOADays) in Belgium and one of those talks, “FreeBSD is not a Linux distro” by Kristof Provost, changed my mind. I really appreciated his talk and started looking back to FreeBSD. Some time later, I bought a few books from Michael W Lucas, watched “BSD Now” podcasts and read other BSD documentation from the internet. Now, I have extra fun with the BSD family.

When starting a new project, like installing a webserver. First we need to collect information by reading, filtering and then transforming the input into a computer-like format: config files, wiki, All those steps take a lot of time and effort. Maybe at the end of the story, the setup is not always going into production but it would be pity to drop it into a trashcan.

FreeBSD, I don't know:

- I'm not a writer.
- My primary language is not English. I don't know how to write nice sentences.
- In my daily work, I don't use BSD, So, I don't know everything about it.
- Sometimes, I write, “I don't know”, because I don't find a good or complete answer about the subject.

But:

- I like the way BSD's solve problems.
- It is a bug when the documentation is wrong or incomplete.
- FreeBSD was the my first BSD.
- Welcome readers from other operation systems. Maybe some part of the documentation you could use in your environment.
- Information is constantly changing. It is difficult to keep up constantly ... I try but I don't know ...
- Dnn't read this book top down. Pick the chapters you like.
- FreeBSD is not owned by a company. So, independence guaranteed.

Chapter 1

Installing BSD

If you start a project and need multiple servers, then you should have a virtual environment like VirtualBox. Normally, it is not too difficult to install a BSD operation system. Which one to choose depends on your use case and your personal choice, FreeBSD, OpenBSD, NetBSD, HardenedBSD, ... Here we discuss OpenBSD.

First make a configuration in VirtualBox. The most important thing is to use 2 network interfaces.

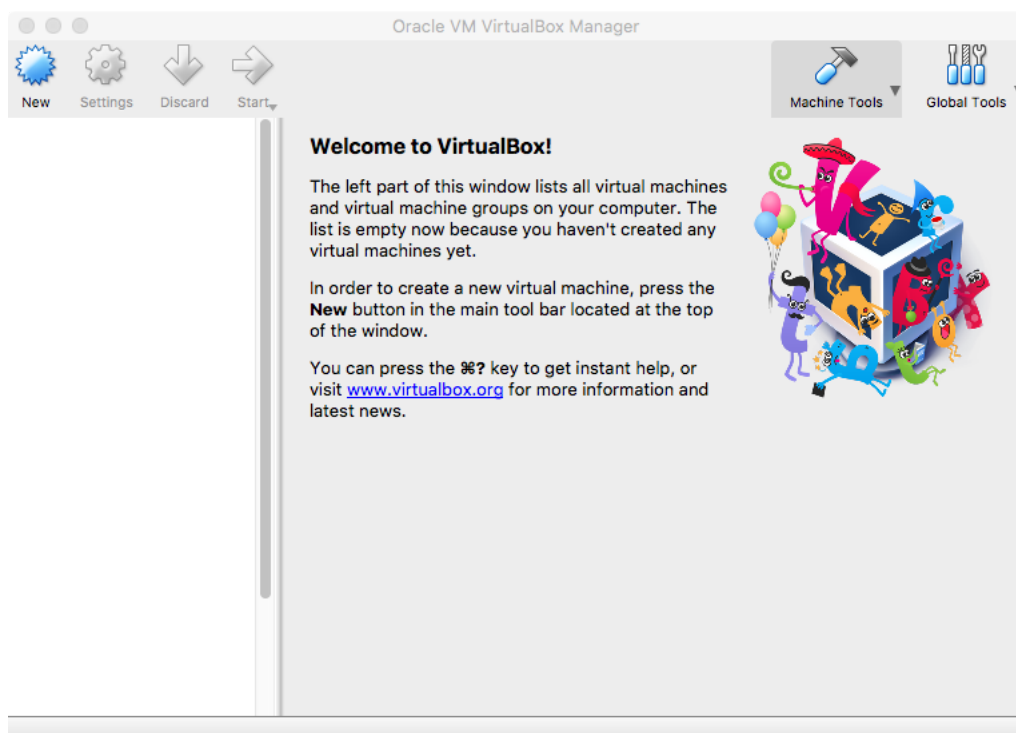
- NAT. With this interface, you communicate with the internet, mostly for downloading packages. You can not communicate with other virtual servers.
- Host-only network. With this interface we can access the different virtual servers in the our sub network and the host (laptop) can communicate with those virtual servers.

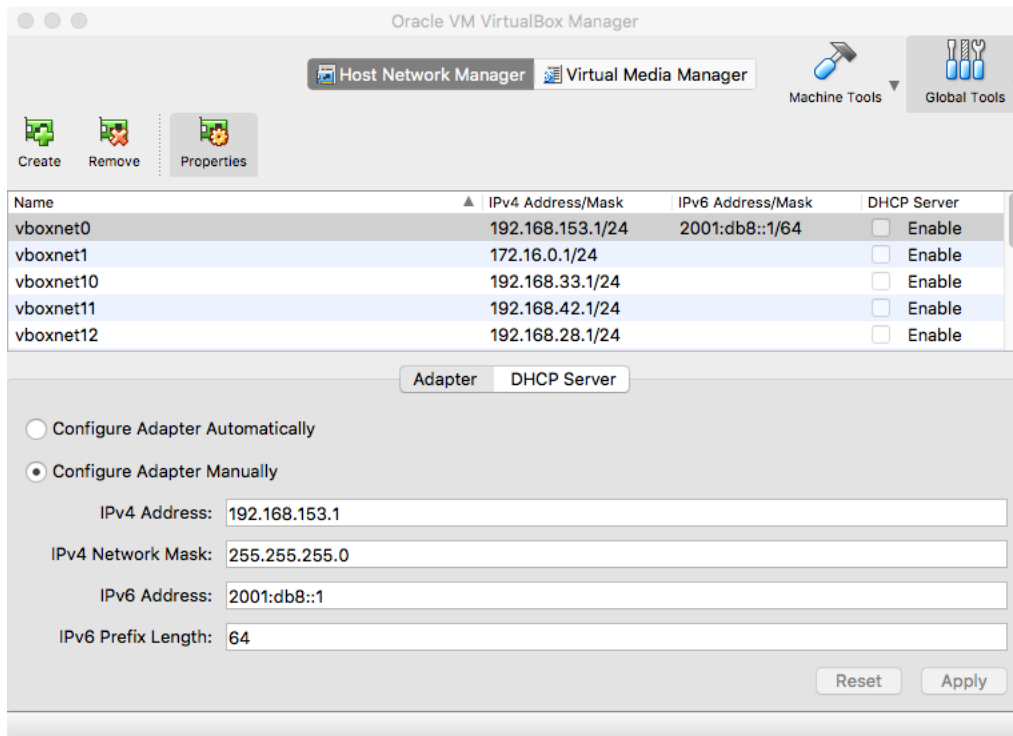
Installing the OS, it just giving the right answer to the questions.

- The user “captain” is in the wheel group. With this user you can do “su -” to get root.
- All the passwords are the same.

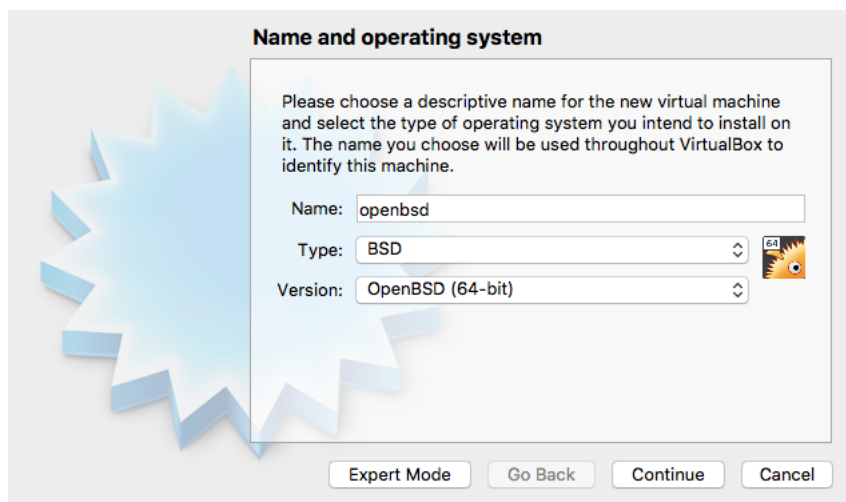
1.1 Configuration OpenBSD in Virtualbox

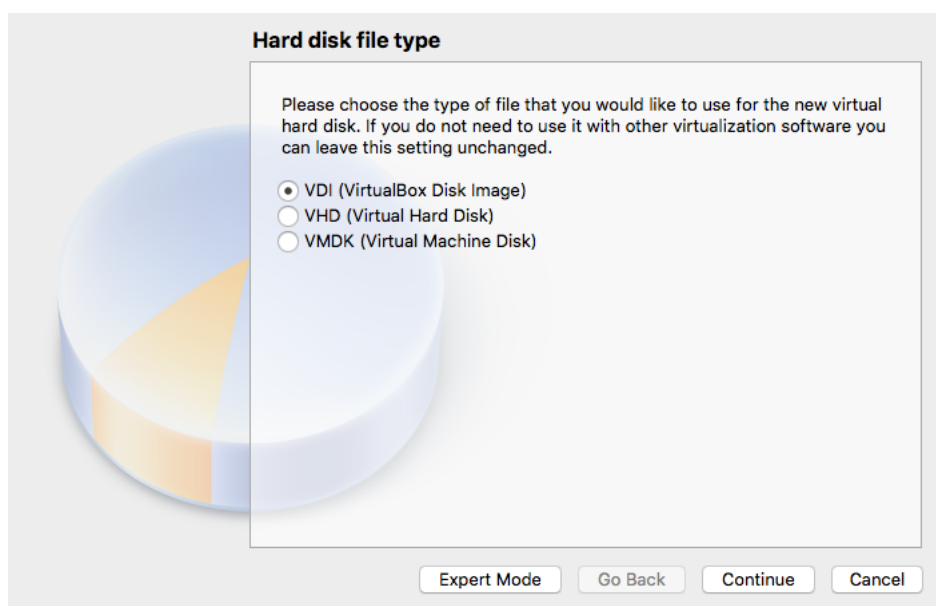
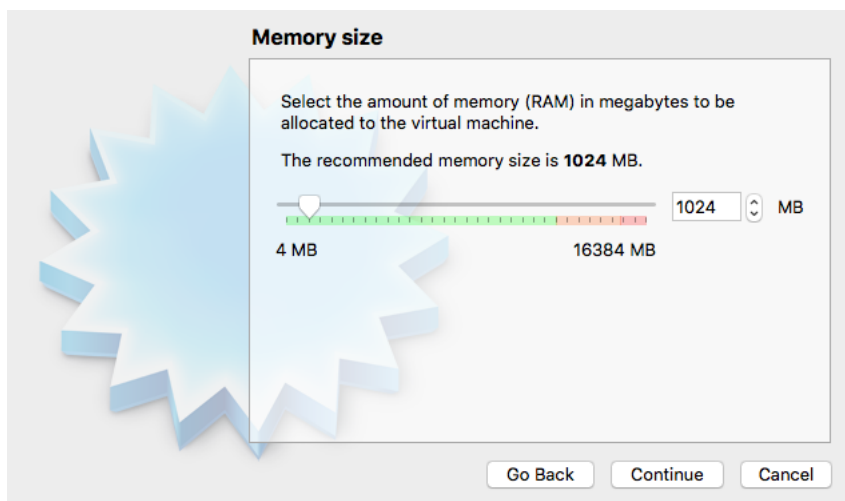
The screenshots show how to configure a virtual server.

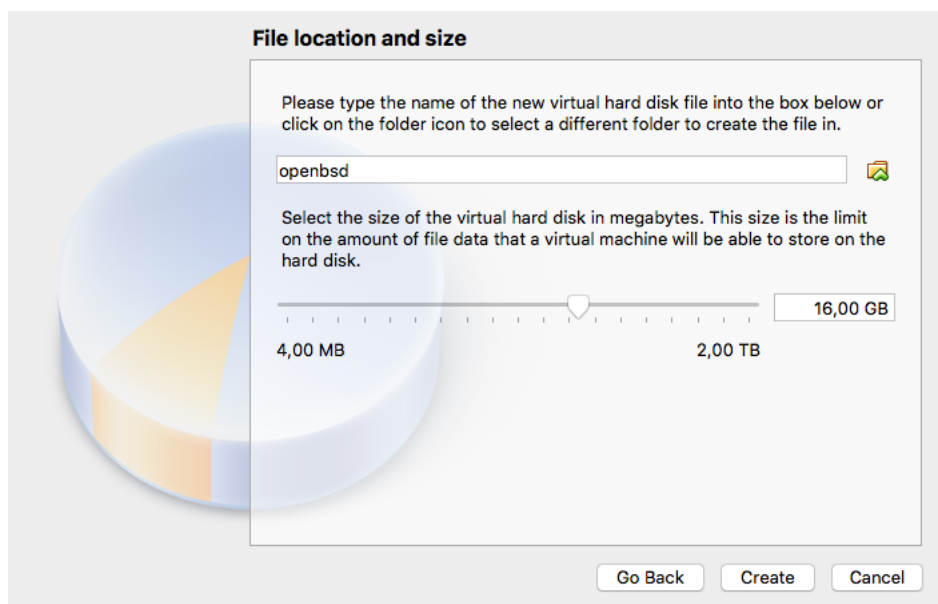


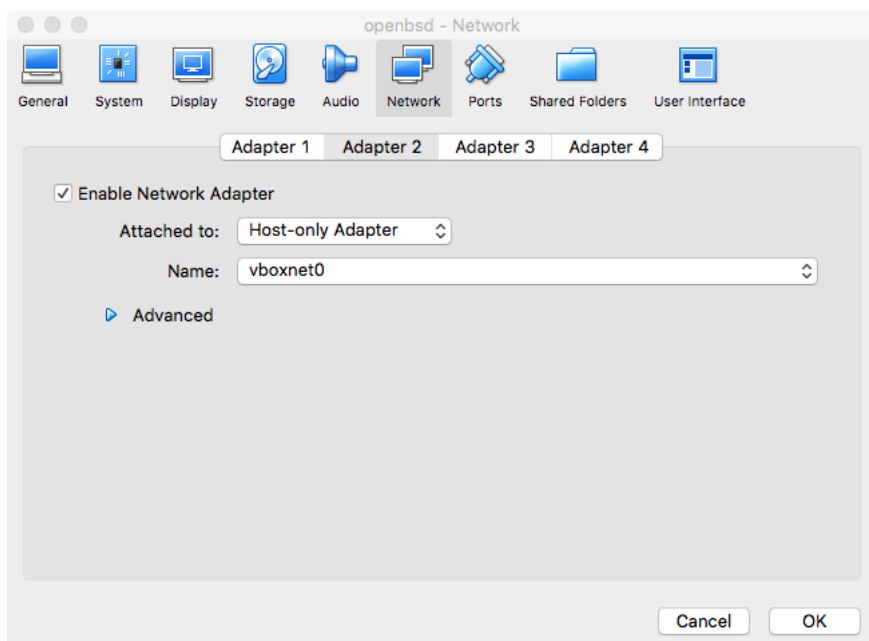
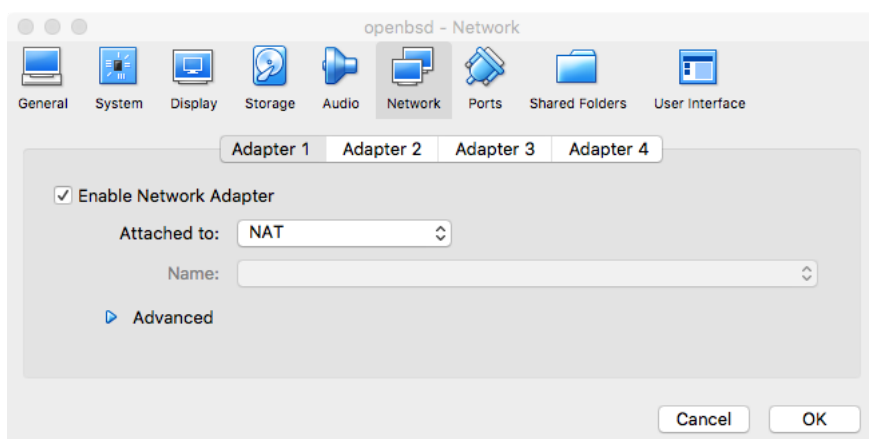
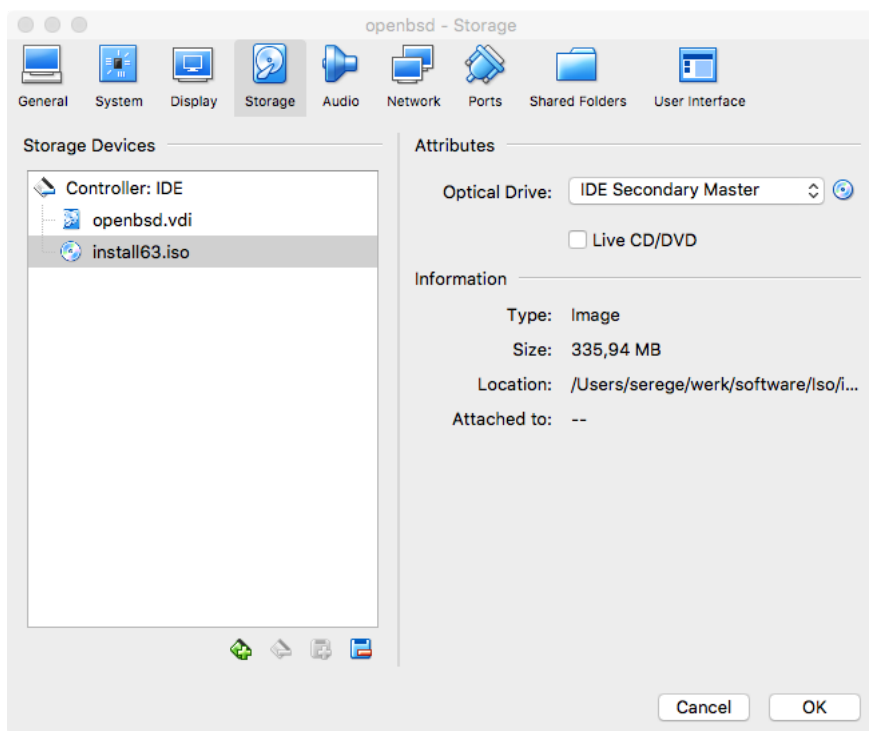


The screen “Host Network Manager” is important. It shows a list of installed network interfaces. These interfaces show up when running “ifconfig -a” or “ip a” in your console terminal. There are 2 ways of installing a network interface. DHCP or manual. DHCP is the easiest but for servers we do manual setting. Prefer the option you like. Once a server is setup with DHCP, it will always get the same ip adres. Here we use manual because I want to choose my ip addresses.









1.2 Instalng OpenBSD in VirtualBox.

Now, the fun part begins. It is not that difficult to answer the questions. On the first interface (em0), which is the NAT interface, we will use DHCP. The second network interface (em1), we will install when the server is booted. We don't need an X system.

```

openbsd [Running]

Welcome to the OpenBSD/amd64 6.3 installation program.
(I)nstall, (U)pgrade, (A)utoinstall or (S)hell? I
At any prompt except password prompts you can escape to a shell by
typing '!'. Default answers are shown in []'s and are selected by
pressing RETURN. You can exit this program at any time by pressing
Control-C, but this can leave your system in an inconsistent state.

Choose your keyboard layout ('?' or 'L' for list) [default] be
System hostname? (short form, e.g. 'foo') openbsd

Available network interfaces are: em0 em1 vlan0.
Which network interface do you wish to configure? (or 'done') [em0]
IPv4 address for em0? (or 'dhcp' or 'none') [dhcp]
em0: bound to 10.0.2.15 from 10.0.2.2 (52:54:00:12:35:02)
IPv6 address for em0? (or 'autoconf' or 'none') [none]
Available network interfaces are: em0 em1 vlan0.
Which network interface do you wish to configure? (or 'done') [done]
DNS domain name? (e.g. 'example.com') [my.domain] gerrysherry.be
Using DNS nameservers at 10.0.2.3

Password for root account? (will not echo)
Password for root account? (again)
Start sshd(8) by default? [yes]
Do you expect to run the X Window System? [yes] no

```

```

openbsd [Running]

System hostname? (short form, e.g. 'foo') openbsd

Available network interfaces are: em0 em1 vlan0.
Which network interface do you wish to configure? (or 'done') [em0]
IPv4 address for em0? (or 'dhcp' or 'none') [dhcp]
em0: bound to 10.0.2.15 from 10.0.2.2 (52:54:00:12:35:02)
IPv6 address for em0? (or 'autoconf' or 'none') [none]
Available network interfaces are: em0 em1 vlan0.
Which network interface do you wish to configure? (or 'done') [done]
DNS domain name? (e.g. 'example.com') [my.domain] gerrysherry.be
Using DNS nameservers at 10.0.2.3

Password for root account? (will not echo)
Password for root account? (again)
Start sshd(8) by default? [yes]
Do you expect to run the X Window System? [yes] no
Setup a user? (enter a lower-case loginname, or 'no') [no] captain
Full name for user captain? [captain]
Password for user captain? (will not echo)
Password for user captain? (again)
WARNING: root is targeted by password guessing attacks, pubkeys are safer.
Allow root ssh login? (yes, no, prohibit-password) [no]

Available disks are: wd0.
Which disk is the root disk? ('?' for details) [wd0]

```

```

openbsd [Running]
Full name for user captain? [captain]
Password for user captain? (will not echo)
Password for user captain? (again)
WARNING: root is targeted by password guessing attacks, pubkeys are safer.
Allow root ssh login? (yes, no, prohibit-password) [no]

Available disks are: wd0.
Which disk is the root disk? ('?' for details) [wd0]
No valid MBR or GPT.
Use (W)hole disk MBR, whole disk (G)PT or (E)dit? [whole]
Setting OpenBSD MBR partition to whole wd0...done.
The auto-allocated layout for wd0 is:
#      size      offset  fstype [fsize bsize  cppl]
a:      562.2M        64  4.2BSD   2048 16384    1 # /
b:      904.5M    1151520    swap
c:     16384.0M        0  unused
d:      779.6M    3003904  4.2BSD   2048 16384    1 # /tmp
e:     1151.8M    4600448  4.2BSD   2048 16384    1 # /var
f:     1312.2M    6959360  4.2BSD   2048 16384    1 # /usr
g:      631.3M    9646816  4.2BSD   2048 16384    1 # /usr/X11R6
h:     2260.7M   10939776  4.2BSD   2048 16384    1 # /usr/local
i:     1464.9M   15569696  4.2BSD   2048 16384    1 # /usr/src
j:     3401.8M   18569792  4.2BSD   2048 16384    1 # /usr/obj
k:     3909.7M   25536640  4.2BSD   2048 16384    1 # /home
Use (A)uto layout, (E)dit auto layout, or create (C)ustom layout? [a]

```

Just lazy and install all packages.

```

openbsd [Running]
Let's install the sets!
Location of sets? (cd0 disk http or 'done') [cd0]
Pathname to the sets? (or 'done') [6.3/amd64]

Select sets by entering a set name, a file name pattern or 'all'. De-select
sets by prepending a '-', e.g.: '-game*'. Selected sets are labelled '[X]'.
[X] bsd          [X] comp63.tgz   [X] xbase63.tgz   [X] xserv63.tgz
[X] bsd.rd       [X] man63.tgz    [X] xshare63.tgz
[X] base63.tgz   [X] game63.tgz   [X] xfont63.tgz
Set name(s)? (or 'abort' or 'done') [done]
Directory does not contain SHA256.sig. Continue without verification? [no] yes
Installing bsd          100% |*****| 12800 KB  00:00
Installing bsd.rd       100% |*****| 9636 KB   00:00
Installing base63.tgz   100% |*****| 142 MB    00:14
Extracting etc.tgz      100% |*****| 260 KB     00:00
Installing comp63.tgz   100% |*****| 77973 KB  00:09
Installing man63.tgz    100% |*****| 7025 KB   00:01
Installing game63.tgz   100% |*****| 2720 KB   00:00
Installing xbase63.tgz  100% |*****| 18164 KB  00:02
Extracting xetc.tgz     100% |*****| 7057      00:00
Installing xshare63.tgz 100% |*****| 4420 KB   00:01
Installing xfont63.tgz  100% |*****| 39342 KB  00:04
Installing xserv63.tgz  100% |*****| 12574 KB  00:01
Location of sets? (cd0 disk http or 'done') [done]

```

```

openbsd [Running]
Installing bsd          100% |*****| 12800 KB  00:00
Installing bsd.rd       100% |*****| 9636 KB   00:00
Installing base63.tgz   100% |*****| 142 MB    00:14
Extracting etc.tgz      100% |*****| 260 KB     00:00
Installing comp63.tgz   100% |*****| 77973 KB  00:09
Installing man63.tgz    100% |*****| 7025 KB   00:01
Installing game63.tgz   100% |*****| 2720 KB   00:00
Installing xbase63.tgz  100% |*****| 18164 KB  00:02
Extracting xetc.tgz     100% |*****| 7057      00:00
Installing xshare63.tgz 100% |*****| 4420 KB   00:01
Installing xfont63.tgz  100% |*****| 39342 KB  00:04
Installing xserv63.tgz  100% |*****| 12574 KB  00:01
Location of sets? (cd0 disk http or 'done') [done]

What timezone are you in? ('?' for list) [Canada/Mountain]
Saving configuration files...done.
Making all device nodes...done.
Relinking to create unique kernel...done.

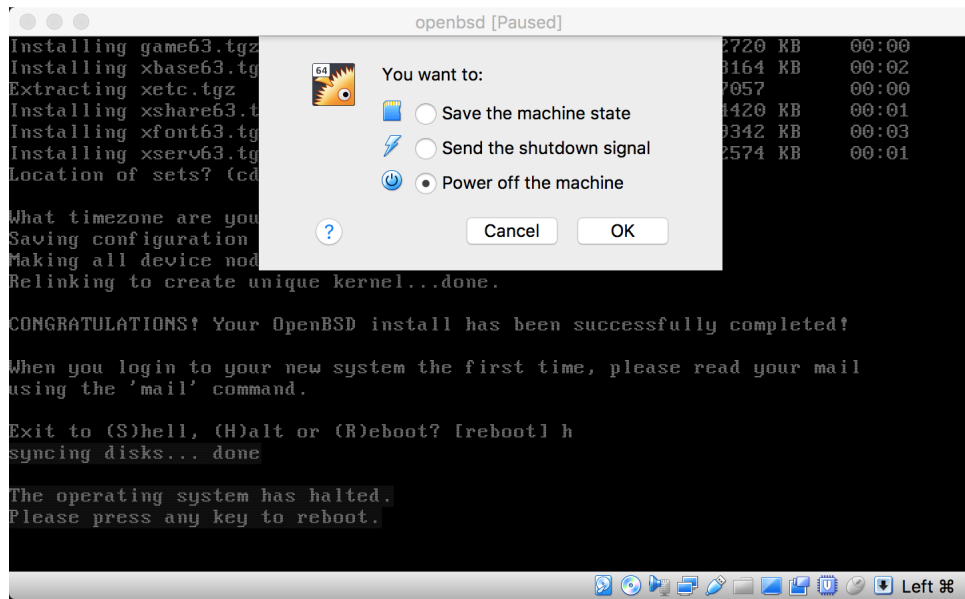
CONGRATULATIONS! Your OpenBSD install has been successfully completed!

When you login to your new system the first time, please read your mail
using the 'mail' command.

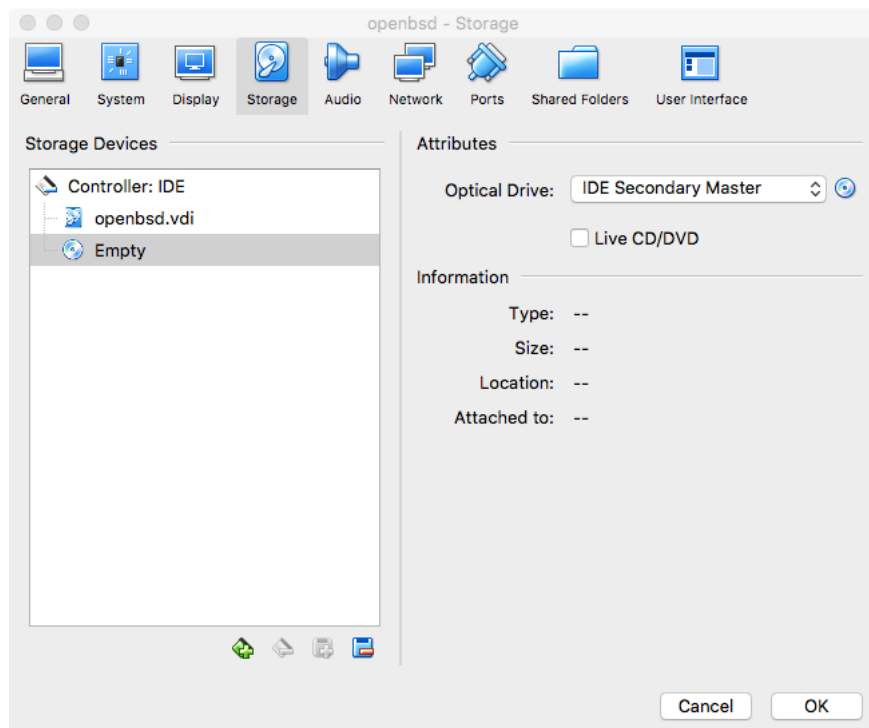
Exit to (S)hell, (H)alt or (R)eboot? [reboot] halt

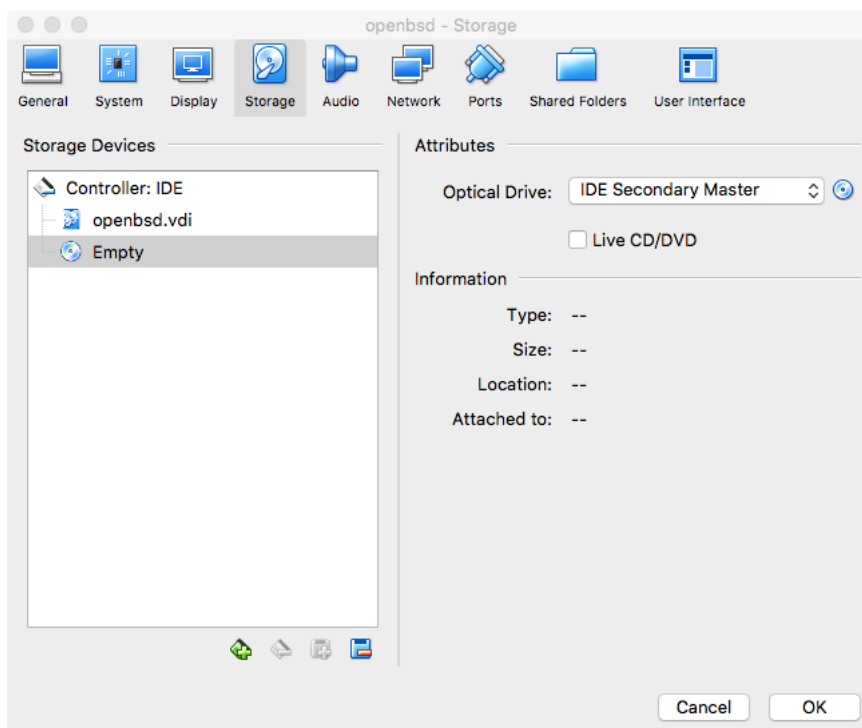
```

Poweroff the system in VirtualBox. Close the window where we install openbsd and a pop-up box appear.

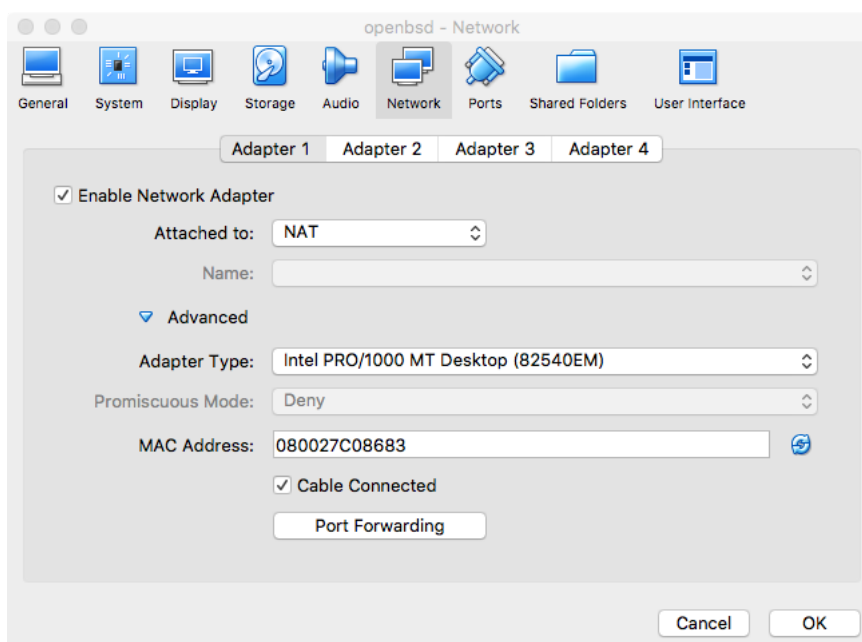


Go back to the settings of the virtual server and remove the install disk from the virtual server. Otherwise the install proces restarts again.

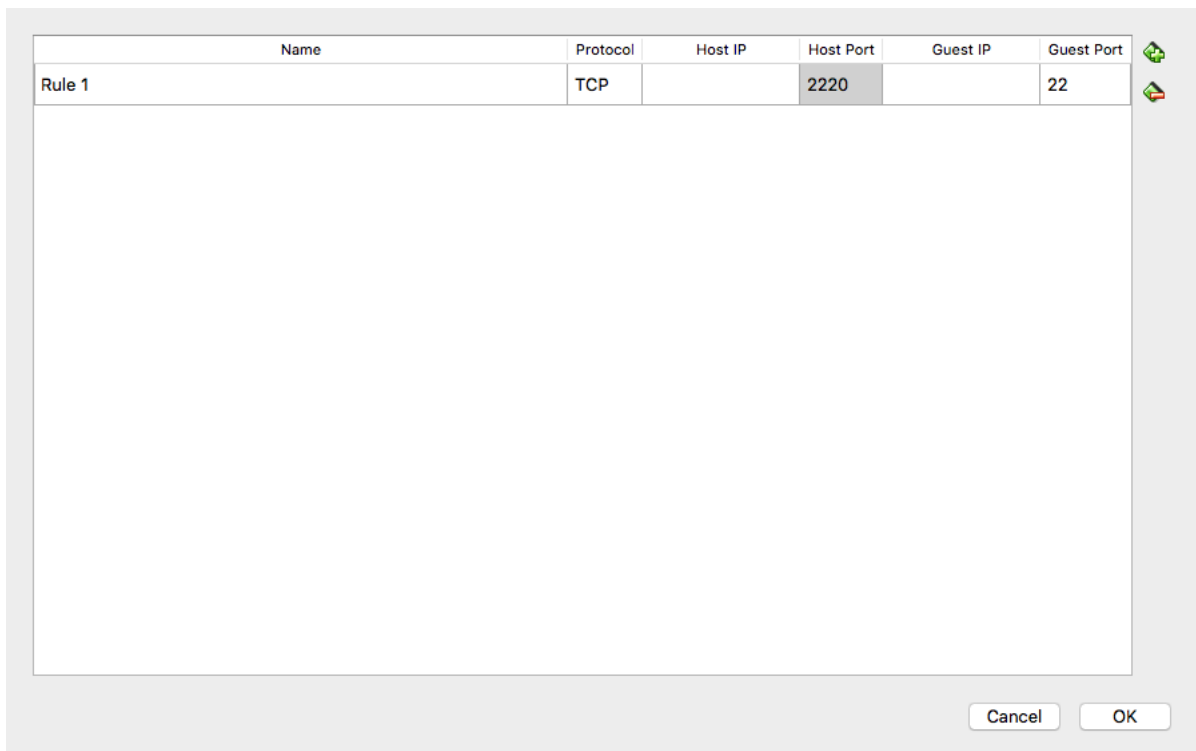




On the NAT interface we will do port forwarding. This makes it easy to login for your host server (laptop). This can also be done when the virtual server is running.



Here we forward port 2220 to ssh.



Start the virtual server. A few seconds later, you are welcomed by a login prompt.

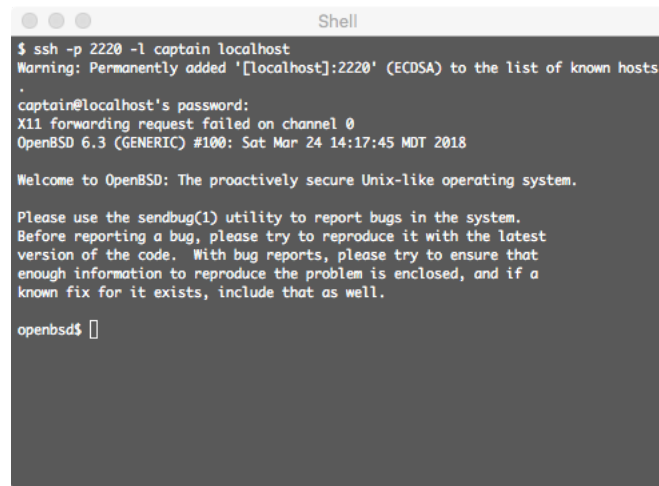
```

openbsd [Running]
kbd: keyboard mapping set to be
pf enabled
starting network
em0: bound to 10.0.2.15 from 10.0.2.2 (52:54:00:12:35:02)
reordering libraries: done.
openssl: generating isakmpd/iked RSA keys... done.
ssh-keygen: generating new host keys: RSA DSA ECDSA ED25519
starting early daemons: syslogd pflogd ntpd.
starting RPC daemons:.
savecore: no core dump
checking quotas: done.
clearing /tmp
kern.securelevel: 0 -> 1
creating runtime link editor directory cache.
preserving editor files.
starting network daemons: sshd smtpd sndiod.
running rc.firsttime
Path to firmware: http://firmware.openbsd.org/firmware/6.3/
Installing: intel-firmware
starting local daemons: cron.
Tue Oct 16 07:58:07 MDT 2018

OpenBSD/amd64 (openbsd.yerrysherry.be) (ttyC0)

login: _

```



```

$ ssh -p 2220 -l captain localhost
Warning: Permanently added '[localhost]:2220' (ECDSA) to the list of known hosts
.
captain@localhost's password:
X11 forwarding request failed on channel 0
OpenBSD 6.3 (GENERIC) #100: Sat Mar 24 14:17:45 MDT 2018

Welcome to OpenBSD: The proactively secure Unix-like operating system.

Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code. With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

openbsd$

```

After the system is installed, check the web page of OpenBSD what to configure next or “man afterboot” or <https://blog.aktsbot.in/c/getting-started.html>. But first we need to configure the second interface. Grab an ip adres in the network range you specified in the “Host Network Manager”

```

openbsd$ su -
Password:
openbsd# vi /etc/hostname.em1
inet 192.168.153.20 255.255.255.0
openbsd# sh /etc/netstart
openbsd# ifconfig em1
em1: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
    lladdr 08:00:27:7a:20:26
    index 2 priority 0 llprio 3
    media: Ethernet autoselect (1000baseT full-duplex)
    status: active
    inet 192.168.153.20 netmask 0xffffffff broadcast 192.168.153.255

```

Now, you can ping and login from your host (laptop) to the virtual server. Well done!

```

my_laptop$ ping 192.168.153.20
PING 192.168.153.20 (192.168.153.20): 56 data bytes
64 bytes from 192.168.153.20: icmp_seq=0 ttl=255 time=0.428 ms
64 bytes from 192.168.153.20: icmp_seq=1 ttl=255 time=0.450 ms
^C
--- 192.168.153.20 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 0.428/0.439/0.450/0.011 ms
my_laptop$ ssh -l captain 192.168.153.20
captain@192.168.153.20's password:
X11 forwarding request failed on channel 0
Last login: Tue Oct 16 08:39:59 2018 from 192.168.153.1
OpenBSD 6.3 (GENERIC) #100: Sat Mar 24 14:17:45 MDT 2018

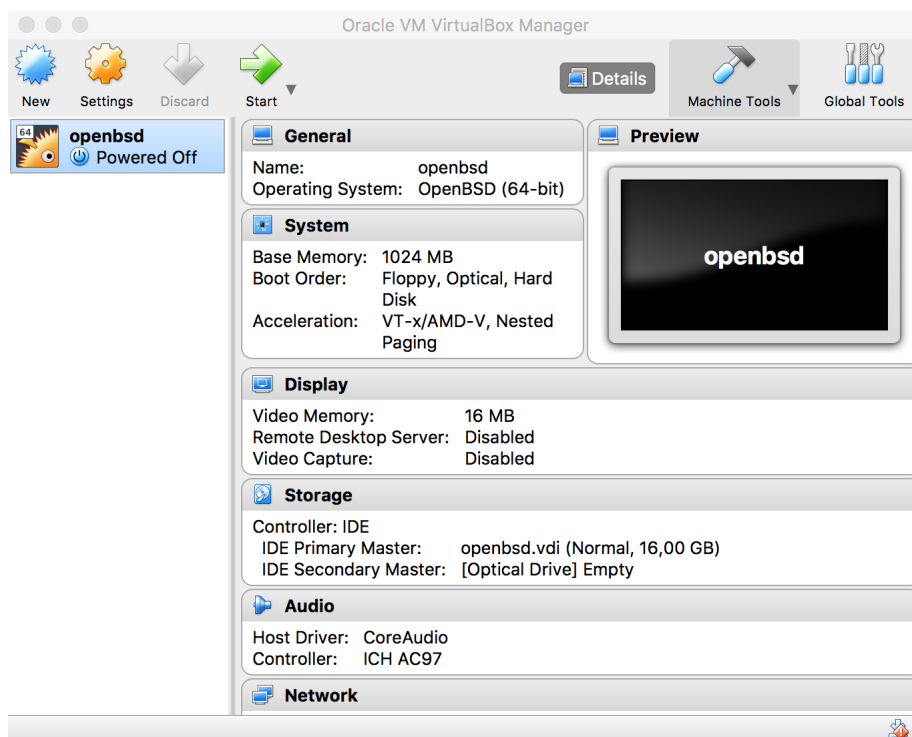
Welcome to OpenBSD: The proactively secure Unix-like operating system.
Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code. With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.

openbsd$

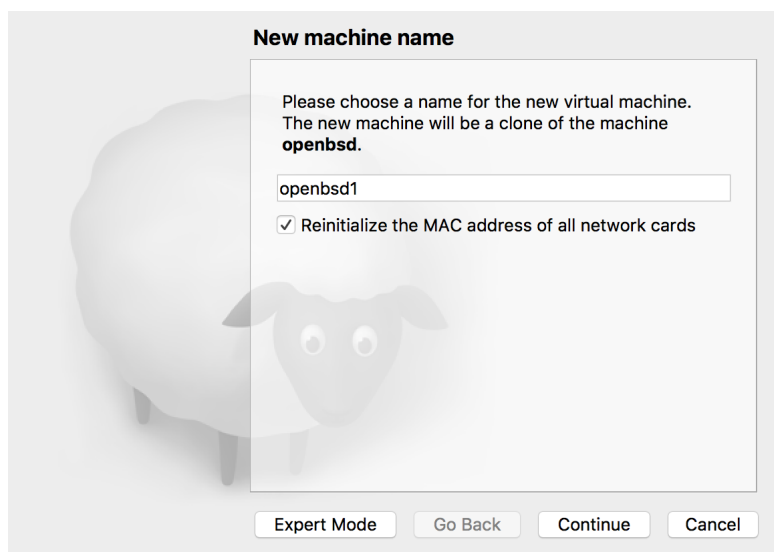
```

1.3 Installing multiple OpenBSD virtual servers

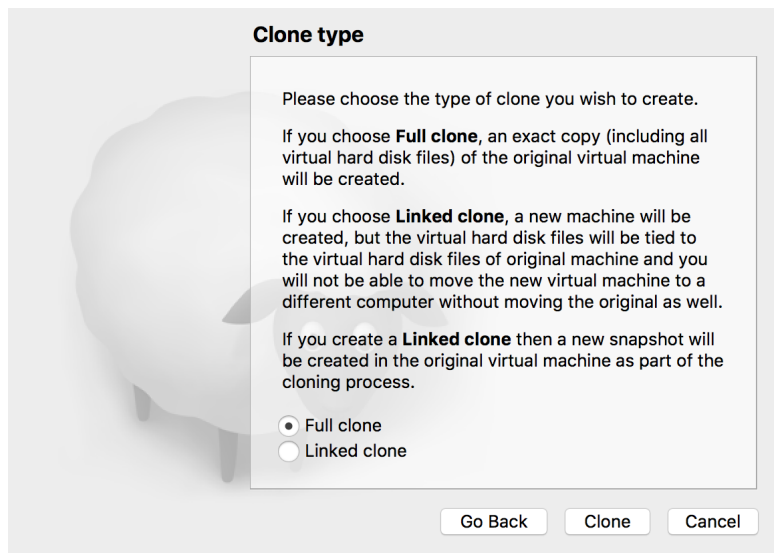
Now, we installed one server but for testing perposes but often we need to install more. You can do the whole procedure again or use a tool like “packer”. Another way is to clone the virtuel server. In VirtualBox, select the virtual server which you want to clone.



Click “Machine -> Clone” and then enter the new hostname. Check also “Reinitialize the MAC address of all network cards”



And select “Full clone”



Because this is a clone, we need to change a few things. Change the port forwarding settings of the NAT Network Adopter to an other number. Click on “Network -> Adaptor 1 -> Advanced -> Port Forwarding” (ex 2221). Start the virtual server. Now, we need to change some things on the virtual server. Login to this server from a terminal and change:

- the hostname
- the em1 network interface
- regenerate ssh keys

At the console it looks like:

```
$ ssh -l captain -p 2221 localhost
captain@localhost's password:
openbsd$ su -
Password:
openbsd#
openbsd# vi /etc/myname
openbsd# vi /etc/hostname.em1
openbsd# cd /etc/ssh; rm ssh_host*
openbsd# ssh-keygen -A
openbsd# sh /etc/netstart
em0: bound to 10.0.2.15 from 10.0.2.2 (52:54:00:12:35:02)
openbsd1#
```


Chapter 2

SSH Certificate Authorities (CA)

2.1 Introduction

There was a very interesting blog written by Marlon Dutra from Facebook about how they scale and secure access with SSH. Nowadays everybody use SSH to login into servers. But Facebook and probably other companies, use a CA SSH to authenticate hosts and users. Some reasons are:

- They have a lot of servers and want to control who can log in.
- No dependencies like LDAP. If there are problems with LDAP, they still can login.

Setup the multiple OpenBSD servers for our test environment:

Server name	Info	Port forwarding	ip adres (/ec/hostname.em1)
mylaptop	Simulating the user laptop	2500	192.168.153.50
sshca	Simulating the Certificate Authority server.	2501	192.168.153.51
postgresql	Simulating a database server	2502	192.168.153.52

Enter folowing lines in /etc/hosts:

```
192.168.153.50 mylaptop.yerrysherry.be mylaptop
192.168.153.51 sshca.yerrysherry.be sshca
192.168.153.52 postgresql.yerrysherry.be postgresql
```

When everything is configured, power down the 3 virtual servers and make a snapshot for each of them. Select a virtual server in VirtualBox, click on “Machine Tools” -> “Snapshots”. Click on the “take +” icon and name your snapshot. When we have problems you can restore from a snapshot. It is very important that your clock is accurate!! Do not let it drift, especially in virtual environments like VirtualBox.

2.2 Standard way to login into server

Create a user “john” on the virtual server “mylaptop” and “postgresql”.

```
# useradd -m john
```

Change the password on the server “postgresql”.

John, a new member of the database team, wants to start quickly in his new role. The sysadmin administrator gives him the new coordinates for logging into his environment.

Add following line to all “.profile” file when creating users.

```
PS1="\h-\u:\$PWD$ "
```

or

```
echo 'PS1="\h-\u:\$PWD$ "' >> .profile
```

Now, we activate the prompt by logging out and in again:

```
. ./profile
```

In our prompt we have the server name, user name and the folder path.

2.2.1 With a password

John starts a new session:

```
mylaptop-john:/home/john$ ssh postgresql.yerrysherry.be
The authenticity of host 'postgresql.yerrysherry.be (192.168.153.52)' can't be established.
ECDSA key fingerprint is SHA256:55gcD2Di21oivq6qG20dL0SGxLiGMjYjeuLw8iC2H0o.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'postgresql.yerrysherry.be,192.168.153.52' (ECDSA) to the list of known hosts.
john@postgresql.yerrysherry.be's password:
Last login: Mon Nov 19 09:28:10 2018 from 192.168.153.50
OpenBSD 6.3 (GENERIC) #100: Sat Mar 24 14:17:45 MDT 2018
Welcome to OpenBSD: The proactively secure Unix-like operating system.
Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code. With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.
postgresql-john:/home/john$
```

He is happy that he can do some stuff. Now, he opens a lot of sessions with the following command.

```
mylaptop-john:/home/john$ ssh 192.168.153.52
john@192.168.153.52's password:
Last login: Mon Nov 19 09:31:37 2018 from 192.168.153.50
....
postgresql-john:/home/john$
```

This is not the best way to login into a server. When this server is connected to the internet, we see a lot of entries in our log files. People, bots, .. are trying to guess standard user accounts and easy passwords to get in.

2.2.2 Private and public key

Hmm, John thinks, why should I always enter my password. My password is always short and easy to guess and that is not good. A well known solution is creating a private and public key. The private key stays on the laptop and the public key is distributed to other servers. On the laptop, generate the key pair with “ssh-keygen”. The contents of “/home/john/.ssh/id_rsa.pub” is distributed to other servers into the “authorized_keys”. Most of the time we use the same user id, “john”. It is possible to use an other one like “captain”. It is possible to copy the public key to root but this is bad practice. Always log into as a user and then use su or sudo to get root.

Enter a passphrase for more security. In this example we hit enter, so no passphrase.

```
mylaptop-john:/home/john$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/john/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/john/.ssh/id_rsa.
Your public key has been saved in /home/john/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:KZJaJfY3HXPw4xGtjECAd7n5vcY1Vd+d2N7xQLKADLY john@mylaptop.yerrysherry.be
The key's randomart image is:
+---[RSA 2048]----+
|      .+=.o.  ..  |
|      ....* .o.o . |
```

```
|  o.E. +o+==o *|
|  . = oo.=++.=|
|  + o S.... .o+|
|  o . o . . . o.o|
|  .      . o . |
|      +      |
|      .      |
+----[SHA256]-----+
```

One more time logging in with his password on postgresql server and then copy the public key “id_rsa.pub” from his laptop account to the account on the postgresql server.

```
mylaptop-john:/home/john$ cat /home/john/.ssh/id_rsa.pub ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAQDDgKfVQ7LWF7rLMMK...1236tRX4pAJ1+OYQIHLMvk81GpktdDt john@mylaptop.yerrysherry.be
postgresql-john:/home/john$ echo "AAAAB3NzaC1yc2EAAAADAQABAAQDDgKfVQ7LWF7rLMMK...1236tRX4pAJ1+OYQIHLMvk81GpktdDt" \
>> /home/john/.ssh/authorized_keys
```

And finally John logs into the postgresql server without a password.

```
mylaptop-john:/home/john$ ssh postgresql.yerrysherry.be
Last login: Mon Nov 19 09:58:56 2018 from 10.0.2.2
...
postgresql-john:/home/john$
```

Sometimes John needs to be root. Here we have 2 solutions:

- Copy his public key in the “authorized_keys” of root. This is very easy but difficult to maintain. We have no clue which people log into the root account.
- Deploy sudo. This is a better way to control what John can do and we have logs.

2.3 Recap John actions

When John makes his first connection to the server postgresql, he quickly enters “yes” without thinking. Most of the time people enter “yes”, otherwise they can not log in.

```
mylaptop-john:/home/john$ ssh postgresql.yerrysherry.be
The authenticity of host 'postgresql.yerrysherry.be (192.168.153.52)' can't be established.
ECDSA key fingerprint is SHA256:55gcD2Di21oivq6qG20dLOSGxLiGMjYjeuLw8iC2H0o.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'postgresql.yerrysherry.be,192.168.153.52' (ECDSA) to the list of known hosts.
```

This question only occurs the first time. The client server (mylaptop) and the remote server (postgresql) need to exchange their keys. This means that the client needs the public key of the server. After entering “yes”, the public key of the server is stored in “/home/john/.ssh/known_hosts” file. The next time john logs in, he is no longer challenged to answer the question. Client and server trust each other. This process is called host-based authentication.

Most of the time it is fine when working internally and you know your servers. Are you still confident when you log into a server you don’t know? How do we know that this server is ok? Maybe DNS is pointing to a bad server? You can call to the sysadmin and ask if this is the right public key. Maybe, the sysadmin refers you to a web page of all his public keys.

There is another way of getting all the public keys from a server.

```
mylaptop-john:/home/john$ ssh-keyscan postgresql.yerrysherry.be
# postgresql.yerrysherry.be:22 SSH-2.0-OpenSSH_7.7
postgresql.yerrysherry.be ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDKm/EjVR8J4TukNvSEIdT0hSJ...
# postgresql.yerrysherry.be:22 SSH-2.0-OpenSSH_7.7
postgresql.yerrysherry.be ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHA...
# postgresql.yerrysherry.be:22 SSH-2.0-OpenSSH_7.7
postgresql.yerrysherry.be ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAIly+WB81gIslA/+74t1MxTNh9HDEQakFqDx98Z5YNctB
```

Each time John makes a connection, SSH will check the “known_hosts” file, looking for the name John entered in the command line and the public key. If the name is not found, an extra line is add to the “known_host” file. You can put all names in the first field, separated with a comma.

For some reasons the sysadmin of the postgresql server changes the public and private keys.

```

postgresql# rcctl stop sshd
postgresql# cd /etc;
postgresql# cp -Rp ssh ssh.old
postgresql# cd ssh; rm ssh_h*
postgresql# ssh-keygen -A
postgresql# rcctl start sshd

```

The next morning John starts early at work:

```

mylaptop-john:/home/john$ ssh postgresql.yerrysherry.be
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@    WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!     @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that a host key has just been changed.
The fingerprint for the ECDSA key sent by the remote host is
SHA256:tmIwgrnXZXHlixRouCzmFPH3RTDV88RItiz2WPu80Rs.
Please contact your system administrator.
Add correct host key in /home/john/.ssh/known_hosts to get rid of this message.
Offending ECDSA key in /home/john/.ssh/known_hosts:1
ECDSA host key for postgresql.yerrysherry.be has changed and you have requested strict checking.
Host key verification failed.
mylaptop-john:/home/john$

```

John receives a nice warning. The sysadmin of the postgresql server changed the public and private keys. It's too early to call the system administrator. John deletes the specific line in the "known_hosts" file and continues to work. We talk a lot about host-based authentication, later we talk more about User-based authentication.

2.4 Using a CA with SSH

When we have a lot of servers and people like sysadmins, database admins, web admins, it is very difficult to maintain. It is impossible to put all our users locally in the "/etc/passwd" file. That's why for regular users we put them in LDAP, Kerberos or an other system. But because such systems are a single point of failure and we still want to maintain our servers, we can create a few local users and then "su" or "sudo" to get root access. Every time, we make a first connection from our laptop to an other server, we need to type "yes" to update the "known_hosts" file. If we want to login without a password, we need to distribute our public keys to other servers. Is it possible to get rid of this?

We need a third party which we can trust, an OpenSSH Certificate Authority. When setting up a CA server, we could consider to login as the "root" user.

2.5 Configuring the Certificate Authority

There is nothing special about a CA server but it needs to be highly protected. There are 2 types of authentication and it is preferable to generate different keys for each type:

- Host authentication
- User authentication

What do we mean about highly protected CA server, some guide lines:

- Only a few people need access to this server.
- Use a bastion server and with trusted network. The CA server is only reachable through a bastion server.
- Better is to allow only outbound connections. The CA server gets the public keys to be signed. The other way is that the clients copy the keys to the CA server. (inbound connections)
- Signing certificates can be done by a local user. There is no reason to do it as root.
- Why not install 2 CA servers. One for signing Host public keys and another for User public keys.

- A good starting point is setting the expiration date no longer than 1 year in the future. Do not wait a few days before resigning the keys. In one year a lot of things could change, news bugs in OpenSsh, better encryption tools, ... that's why we do not set the expiration too long in the future.
- Use a good passphrase when generating keys.
- Use a provision tool like Puppet, Ansible, Chef, ... to automate the whole process. This is really a must when regenerating, revoking keys. In a cloud environment we reinstall servers most of the time instead of updating them.

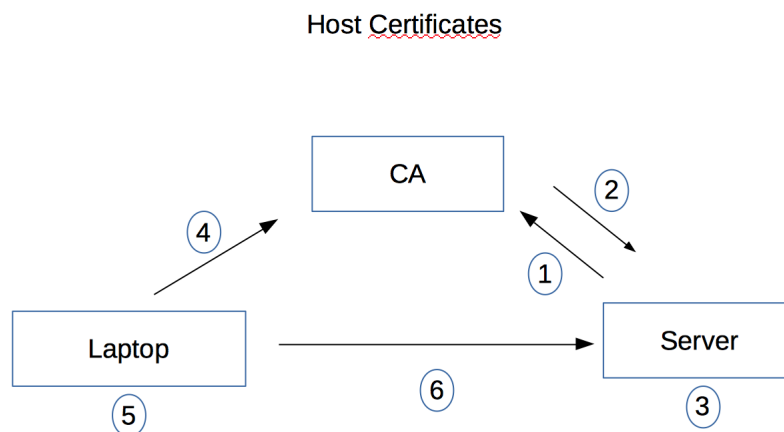
First install the “ca” user on the server “sshca”. This is the login who will sign the keys.

```
sshca# useradd -m ca
sshca# su - ca
sshca$ echo 'PS1="\h-\u:\$PWD$ "' >> .profile
sshca$ . ~/.profile
sshca-ca:/home/ca$ mkdir host-sshca user-sshca hosts users
sshca-ca:/home/ca$ cd host-sshca
sshca-ca:/home/ca/host-sshca$ ssh-keygen -t rsa -f 2018-12-07-host-sshca-key -C 'CA host key create on 2018-12-07'
sshca-ca:/home/ca/host-sshca$ cd /home/ca/user-sshca/
sshca-ca:/home/ca/user-sshca$ ssh-keygen -t rsa -f 2018-12-07-user-sshca-key -C 'CA user key create on 2018-12-07'
```

First we add the user and update the shell prompt, so we have a nicer view of who is logged in and where we are in the folder structure. Then, we create 4 folders. We generate keys in the “host-sshca” and “user-sshca”. Feel free to choose a good passphrase. The date is part of the name of our keys, so we know when the keys are generated and probably we want to change our keys in the future. Those keys we use for signing the different public keys from the hosts and users. For each host and user, a separate folder will be created. In this way we have a nice overview.

2.6 Host Certificate Authority

The host certificate authenticates hosts to users. All our servers host keys are signed with the CA. When a client connects to a server, the user won't be bothered about trusting the key. No need to type “yes” to update the “known_hosts” file. Most of the time the client is our laptop. Let's have an overview of this process.



1. After installing the server, copy the public keys in the folder “/etc/ssh/*pub” to the CA server for signing. Making a incoming connection to the CA from a lot of servers is not that secure. There are better ways for getting those public keys. OpenSSH has 4 types of public keys. DSA keys are not safe anymore, use RSA or ed25519. The latter is more secure then RSA but older environments may not support it.

- ssh_host_dsa_key.pub
- ssh_host_ecdsa_key.pub
- ssh_host_ed25519_key.pub
- ssh_host_rsa_key.pub

2. The CA server signs the public hosts keys and send them back to the server.

3. For each signed host key, a “HostCertificate” exists in the file “sshd_config”. Restart the ssh daemon.
4. Get the public host CA key.
5. On the client side (laptop) changes the ‘known_host’ file. Either globally or for each user. Each line exits of 3 columns
 - @cert-authority
 - domainname
 - public key from the CA
6. Make a ssh connection to the server.

Let’s play with the account “john”

1. Get the public keys from the server postgresql.

```
sshca-ca:/home/ca/hosts$ mkdir postgresql.yerrysherry.be
sshca-ca:/home/ca/hosts$ cd postgresql.yerrysherry.be
sshca-ca:/home/ca/hosts/postgresql.yerrysherry.be$
ssh-keyscan -t rsa postgresql.yerrysherry.be | awk '{print $2 " " $3}' > 2018_ssh_host_rsa_key.pub
```

For each server we make a folder with it’s FQDN and store the public key. The name of the public key starts with the year. Some people regenerates their server hosts multiple times a year. Add the month to the file name. As long as the file name is unique, we are fine.

2. Sign the public keys on the sshca server.

```
sshca-ca:/home/ca/hosts/postgresql.yerrysherry.be$
ssh-keygen -s /home/ca/host-sshca/2018-12-07-host-sshca-key \
-I "postgresql.yerrysherry.be" \
-n "postgresql.yerrysherry.be,postgresql,192.168.153.52" \
-V -1w:+52w \
-h \
/home/ca/hosts/postgresql.yerrysherry.be/2018_ssh_host_rsa_key.pub
```

-s: Sign the certificate with this private key.

-I: Certificate identity. The "key identifier" is logged by the server.

-n: Principals. It is a list of hosts we use for logging it. This means:

- ssh postgresql.yerrysherry.be
- ssh postgresql
- ssh 192.168.153.52

-V: How long is this certificate valid in the future. In this case -1 week and +52 weeks. Use “-V always:forever” if you have a boss who don’t like security.

-h: Only use this option for host authentication.

The output is a file “2018_ssh_host_rsa_key-cert.pub”. To get more info about this certificate:

```
sshca-ca:/home/ca/hosts/postgresql.yerrysherry.be$ ssh-keygen -Lf 2018_ssh_host_rsa_key-cert.pub
2018_ssh_host_rsa_key-cert.pub:
Type: ssh-rsa-cert-v01@openssh.com host certificate
Public key: RSA-CERT SHA256:mfcs7howNVrUpLo/wON/aG1EIpHHWDxnKE6770ITS0Y
Signing CA: RSA SHA256:Frz29Z+WV3zVZH7IfY21T27m0xXx/FulkQm4cRa9k9c
Key ID: "postgresql.yerrysherry.be"
Serial: 0
Valid: from 2018-12-06T23:26:08 to 2019-12-12T23:26:08
Principals:
    postgresql.yerrysherry.be
    postgresql
    192.168.153.52
Critical Options: (none)
Extensions: (none)
```

Copy the certificate key to the server “postgresql” in the folder “/etc/ssh/ca”. All CA configuration comes into a separate folder. Of course, we need root access. Open the “sshd_config” file and add the “HostCertificate” variable with the absolute path of the certificate key.

3. Update the “HostCertificate” on the server

```
postgresql# vi /etc/ssh/sshd_config
HostCertificate /etc/ssh/ca/2018_ssh_host_rsa_key-cert.pub
postgresql#
postgresql# rcctl restart sshd
sshd(ok)
sshd(ok)
```

You could have multiple “HostCertificate” keys, ex:

```
HostCertificate /etc/ssh/ca/2018_ssh_host_rsa_key-cert.pub
HostCertificate /etc/ssh/ca/2019_ssh_host_rsa_key-cert.pub
HostCertificate /etc/ssh/ca/2018_ssh_host_ed25519_key-cert.pub
```

4. Get the public CA key from the sshca server.

```
sshca-ca:/home/ca$ cat /home/ca/host-sshca/2018-12-07-host-sshca-key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDJmwBuMVq+oQQRvOpogu03xQ5HhLXy9G...
...f1stwrF1hRqY/ojp3Ku21KsEqZ CA host key create on 2018-12-07
```

The public key is part of the “known_hosts” file.

5. Edit the “known_host” file on the laptop.

We have the option doing it globally (“/etc/ssh/known_hosts”) or for each user. First we get rid of the old “known_host” file. The “known_hosts” file has multiple fields.

- @cert-authority. We want to use a CA.
- * The domain name this certificate is valid. We could use *.yerrysherry.be but then we can only use postgresql.yerrysherry.be for logging in. That’s why we use multiple principals for logging in.
- The public CA key.

```
mylaptop-john:/home/john/.ssh$ mv known_hosts known_hosts.old
mylaptop-john:/home/john/.ssh$ vi known_hosts
@cert-authority * ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDJmwBuMVq+oQQRvOpogu03xQ5HhLXy9Gf...
...4r6oRwreC0rbDpzzTkVsdHzf1stwrF1hRqY/ojp3Ku21KsEqZ CA host key create on 2018-12-07
```

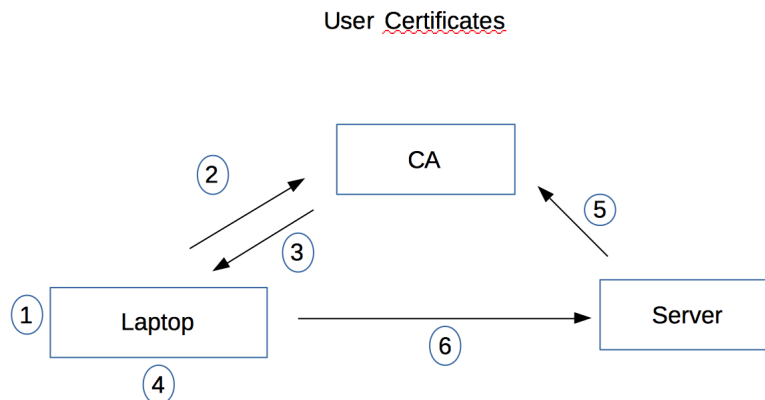
6. Make a ssh connection to the server.

Everything is setup.

```
mylaptop-john:/home/john$ ssh -v postgresql.yerrysherry.be
...
debug1: Authenticating to postgresql.yerrysherry.be:22 as 'john'
debug1: SSH2_MSG_KEXINIT sent
debug1: SSH2_MSG_KEXINIT received
debug1: kex: algorithm: curve25519-sha256
debug1: kex: host key algorithm: ssh-rsa-cert-v01@openssh.com
debug1: kex: server->client cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: kex: client->server cipher: chacha20-poly1305@openssh.com MAC: <implicit> compression: none
debug1: expecting SSH2_MSG_KEX_ECDH_REPLYdebug1: Server host certificate: \
    ssh-rsa-cert-v01@openssh.com SHA256:mfcs7howNVrUpLo/w0N/ag1EIpHHWDxnKE6770ITS0Y, \
serial 0 ID "postgresql.yerrysherry.be" CA ssh-rsa SHA256:Frz29Z+VW3zVZH7IfY21T27m0xXx/FulkQm4cRa9k9c \
valid from 2018-12-06T23:26:08 to 2019-12-12T23:26:08
debug1: Host 'postgresql.yerrysherry.be' is known and matches the RSA-CERT host certificate.
...
```

2.7 User Certificate Authority

The user certificate authenticates users to hosts. All our users keys are signed with the CA. The users are not borrowed putting there public keys in “authorized_key” files. This simplifies a lot when there are a lot of users.



1. Generate a key pair for the user.
2. Send the user public key to the CA server.
3. The CA server signs the user public key. We now have a user certificate key which we send back to the user.
4. Place the user certificate key into the “.ssh” folder of the user.
5. Get the public CA user key from the server. Update the “TrustedUserCAKeys” in the “sshd_config” file, remove the “authorized_keys”. Restart the ssh daemon.
6. Make a ssh connection to the server.

Ok, let’s try this scenario with the user “john”.

1. Generate a key pair for the user.

User “john” already creates a private and public key, otherwise “ssh-keygen -t rsa” creates a new one.

```

mylaptop-john:/home/john/.ssh$ ls -l
total 16
-rw----- 1 john john 1679 Nov 19 10:54 id_rsa
-rw-r--r-- 1 john john 410 Nov 19 10:54 id_rsa.pub
-rw-r--r-- 1 john john 400 Dec 12 23:21 known_hosts
-rw-r--r-- 1 john john 374 Nov 19 09:35 known_hosts.old
mylaptop-john:/home/john/.ssh$ cat id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDDgKfVQ7LWF7r....

```

2. Send the user public key to the CA server.

Like hosts, we put all the users in a separate folder on the CA server. Try to make all your users unique. Here, we copy & paste the file.

```

sshca-ca:/home/ca$ cd users
sshca-ca:/home/ca/users$ mkdir john
sshca-ca:/home/ca/users$ cd john
sshca-ca:/home/ca/users/john$
sshca-ca:/home/ca/users/john$ vi 2018_id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDDgKfVQ7LWF7r....

```

We save his public key with the year in front. Add the month in the filename for a better description. It is a good idea to save all the public keys of the user.

3. Sign the user public key on the CA server.

```
ssh-keygen -s /home/ca/user-sshca/2018-12-07-user-sshca-key \
-I "john" \
-n "john" \
-V -1w:+52w \
/home/ca/users/john/2018_id_rsa.pub
Signed user key /home/ca/users/john/2018_id_rsa-cert.pub: id "john" serial 0 for john valid from 2018-12-08T02:53:33 to 2019-12-14T02:53:33
```

Signing the key is nearly the same as signing the host key. We don't need the "-h" flag because this is only for signing host keys. We change the principal flag "-n" with the users login name. Then we send the user certificate key back to the user.

4. Place the user certificate key into the ".ssh" folder of the user.

Here we just open the file with vi and paste the user public certificate key. The name of the file must be "id_rsa-cert.pub". Let's see more information about the certificate.

```
mylaptop-john:/home/john/.ssh$ vi id_rsa-cert.pub
sh-rsa-cert-v01@openssh.com AAAAHhNzaC1yc2EtY2V...
mylaptop-john:/home/john/.ssh$ ssh-keygen -Lf ./id_rsa-cert.pub
id_rsa-cert.pub:
  Type: ssh-rsa-cert-v01@openssh.com user certificate
  Public key: RSA-CERT SHA256:KZJaJfY3HXPw4xGtjECAd7n5vcY1Vd+d2N7xQLKADLY
  Signing CA: RSA SHA256:G5vkBB/19wMN9D8zO9n0vaxXCwQwgfedadAr9DpQ9Lw
  Key ID: "john"
  Serial: 0
  Valid: from 2018-12-08T02:53:33 to 2019-12-14T02:53:33
  Principals:
    john
  Critical Options: (none)
  Extensions:
    permit-X11-forwarding
    permit-agent-forwarding
    permit-port-forwarding
    permit-pty
    permit-user-rc
postgresl-john:/home/john/.ssh$ mv c authorized_keys.old
```

5. Get the public CA user key from the server.

First, get the user public CA key from the CA server and store it in the folder "/etc/ssh/ca/" as "2018-12-07-user-sshca-key.pub" files. Do it as the "root" user.

```
sshca-ca:/home/ca/user-sshca$ cat 2018-12-07-user-sshca-key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDIu7...
postgresl-root:/etc/ssh/ca$ vi 2018-12-07-user-sshca-key.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDIu7...
```

Then update the "sshd_config" file. SSH needs to know that we use a user certificate. Add the path to the "TrustedUserCAKeys". Don't forget to restart ssh.

```
postgresl-root:/etc/ssh# vi sshd_config
TrustedUserCAKeys /etc/ssh/ca/2018-12-07-user-sshca-key.pub
postgresl-root:/etc/ssh# rcctl restart sshd
sshd(ok)
sshd(ok)
```

We don't use the "authorized_keys" file anymore.

```
postgresl-john:/home/john/.ssh$ mv authorized_keys authorized_keys.old
```

6. Ssh to our server

Now, the only thing to do is to log into the postgresql server.

```
mylaptop-john:/home/john$ ssh postgresql
Last login: Sun Dec 16 11:26:12 2018 from 192.168.153.50
OpenBSD 6.3 (GENERIC) #100: Sat Mar 24 14:17:45 MDT 2018
Welcome to OpenBSD: The proactively secure Unix-like operating system.
Please use the sendbug(1) utility to report bugs in the system.
Before reporting a bug, please try to reproduce it with the latest
version of the code. With bug reports, please try to ensure that
enough information to reproduce the problem is enclosed, and if a
known fix for it exists, include that as well.
postgresql-john:/home/john$
```

In the “authlog”, we see how users connect and disconnect.

```
postgresql-root:/root# tail -f /var/log/authlog
...
Dec 17 03:53:47 postgresql sshd[86645]: Accepted publickey for john from 192.168.153.50 port 38440 ssh2: RSA-CERT ID\john
(serial 0) CA RSA SHA256:G5vkBB/19wMN9D8zO9n0vaxXCwQwgfedadAr9DpQ9Lw
...
Dec 17 03:57:29 Postgresql Sshd[31896]: Received Disconnect From 192.168.153.50 Port 38440:11: Disconnected By User
Dec 17 03:57:29 Postgresql Sshd[31896]: Disconnected From User John 192.168.153.50 Port 38440
```

2.8 Logging in as root

A general rule is not logging straight into the “root” account. Always log in as a user and then use “su” or “sudo” to get root access. A mistake in the configuration file of “sudo” (/etc/sudoers), is hard to fix when rolled out to many servers.

When using a CA, it is ok for logging in as root. We need to do a few configuration steps for logging in as “root”. Those steps are done in a controlled way. Let’s see a few ways how this could be done.

2.8.1 With principals

A senior sysadmin “ann” needs access to the “postgresql” server as user “ann” and sometimes as “root”.

Configuration on her laptop. Feel free to give her a password or passphrase. Don’t forget to update her “known_hosts” file with the host certificate file. Otherwise, we need to type “y” for accepting the fingerprint.

```
mylaptop# useradd -m ann
mylaptop# su - ann
mylaptop$ ssh-keygen -t rsa
mylaptop$ cat /home/ann/.ssh/id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQz3PlmS8d13ePGeEgF95zx+6pAf....
vi /home/ann/.ssh/known_hosts
@cert-authority * ssh-rsa AAAAB3NzaC1yc2EAAA....
```

Configuration on the CA server. Copy “ann” public key (“/home/ann/.ssh/id_rsa.pub”) to “2019_id_rsa.pub”, signs the key with “ssh-keygen”. We now have a certificate user key, check it. We see that there are 2 principals “ann” and “root”.

```
sshca-ca:/home/ca/users$ mkdir ann; cd ann
sshca-ca:/home/ca/users/ann$ vi 2019_id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQz3PlmS8d13ePGeEgF95zx+6pAf....
sshca-ca:/home/ca/users/ann$ \
ssh-keygen -s /home/ca/user-sshca/2018-12-07-user-sshca-key \
-I "ann" \
-n "ann,root" \
-V -1w:+52w \
/home/ca/users/ann/2019_id_rsa.pub
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQDQz3PlmS8d13ePGeEgF95zx+6pAf....
Signed user key /home/ca/users/ann/2019_id_rsa-cert.pub: \
id "ann" serial 0 for ann,root valid from 2018-12-10T18:19:22 to 2019-12-16T18:19:22
sshca-ca:/home/ca/users/ann$ ssh-keygen -Lf ./2019_id_rsa-cert.pub
./2019_id_rsa-cert.pub:
Type: ssh-rsa-cert-v01@openssh.com user certificate
Public key: RSA-CERT SHA256:sikW0Xz8sRhKoz1c3K/HGCxHkkw42+SEhGm9pzav5u8
Signing CA: RSA SHA256:G5vkBB/19wMN9D8zO9n0vaxXCwQwgfedadAr9DpQ9Lw
Key ID: "ann"
Serial: 0
Valid: from 2018-12-10T18:19:22 to 2019-12-16T18:19:22
Principals:
```

```

ann
root
Critical Options: (none)
Extensions:
    permit-X11-forwarding
    permit-agent-forwarding
    permit-port-forwarding
    permit-pty
    permit-user-rc

```

Now, we send back the certificate user key to the laptop of “ann” and store it in her “ssh” folder.

```

sshca-ca:/home/ca/users/ann$ cat 2019_id_rsa-cert.pub
ssh-rsa-cert-v01@openssh.com AAAAHNzaC1yc2EtY2VydC12...
mylaptop$ vi /home/ann/.ssh/id_rsa-cert.pub
ssh-rsa-cert-v01@openssh.com AAAAHNzaC1yc2EtY2VydC12...

```

Now, the final configuration on the server “postgresql”. Of course “ann” needs an account on this server and most importantly people need to have the possibility to login as root. OpenBsd is very strict when logging in as root, it prohibits it completely.

```

postgresql-root:/root# useradd -m ann
postgresql-root:/root# vi /etc/ssh/sshd_config
#PermitRootLogin no
PermitRootLogin prohibit-password
...
postgresql-root:/root# rcctl restart sshd
sshd(ok)
sshd(ok)

```

If everything is done correctly, “ann” can login into the postgresql server from her laptop. Let’s check the log file on the postgresql server.

```

postgresql-root:/root# tail -f /var/log/authlog
Jan 18 09:48:52 postgresql sshd[93212]: Accepted publickey for ann from 192.168.153.50 port 39429 ssh2: \
RSA-CERT ID ann (serial 0) CA RSA SHA256:G5vkBB/19wMN9D8z09n0vaxXCwQwgfedadAr9DpQ9Lw
Jan 18 09:49:37 postgresql sshd[23237]: Accepted publickey for root from 192.168.153.50 port 48908 ssh2: \
RSA-CERT ID ann (serial 0) CA RSA SHA256:G5vkBB/19wMN9D8z09n0vaxXCwQwgfedadAr9DpQ9Lw

```

Very nice, first we see that “ann” logs in with her account and a little bit later she logs in as “root”.

2.8.2 With AuthorizedPrincipalsFile

In the last section, we add users to the principal argument in the user certificate. This is ok when we have a few users who need to login as a specific user. But it gets cumbersome very quickly and it doesn’t scale very well. Everytime a user needs to login as an other user, we need to change the user certificate. A better solution is to use the “AuthorizedPrincipalsFile” sshd option. We can do it globally or on a user basis.

Let’s suppose that the user “john” needs root access on the PostgreSQL server. First we need to change the “sshd_config” file. Add or change the option “AuthorizedPrincipalsFile”. Then we specify a folder name with contains “%u”, like “/etc/ssh/ca/%u_auth_principals”. The “%u” will be substitute with the login name of the user, in our case “root”. After changing the “sshd_config” file, restart the “sshd” daemon.

```

postgresql-root:/etc/ssh/# vi sshd_config
AuthorizedPrincipalsFile /etc/ssh/ca/%u_auth_principals
postgresql-root:/etc/ssh# rcctl restart sshd
sshd(ok)
sshd(ok)

```

For each user who needs “root” access, we enter a new line with a principal from the user certificate. In this case “john”.

```

postgresql-root:/etc/ssh/ca# vi root_auth_principals
john

```

Test if user “john” can login as root on the postgresql server.

```
mylaptop-john:/home/john$ ssh -l root postgresql
...
postgresql-root:/etc/ssh#
```

Now let's see if "john" still can login with his username.

```
mylaptop-john:/home/john$ ssh postgresql
no such identity: /home/john/.ssh/id_xmss: No such file or directory
john@postgresql's password:
postgresql-root:/root# tail -f /var/log/authlog
Jan 31 01:20:19 postgresql sshd[8155]: error: Certificate does not contain an authorized principal
```

Hmmm, this is not the case, a password prompt appears. The reason is that we use the "authorizedPrincipalsFile" option. The solution is to make a new file with the principal for "john" on the postgresql server.

```
postgresql-root:/etc/ssh/ca# vi john_auth_principals
john
mylaptop-john:/home/john$ ssh postgresql
...
postgresql-john:/home/john$
```

Because we are using the "authorizedPrincipalsFile" option, the user "ann" has the same problem. She can not login. We need to make auth_principals files for the user "ann" as we did for the user "john".

All the auth_principals files are centralized in the folder "/etc/ssh/ca" but as said before we can also put the file into the users home folder. Change the "sshd_config" file on the postgresql server.

```
postgresql-root:/etc/ssh# vi sshd_config
#AuthorizedPrincipalsFile /etc/ssh/ca/%u_auth_principals
AuthorizedPrincipalsFile .ssh/auth_principals
...
postgresql-root:/etc/ssh# rcctl restart sshd
sshd(ok)
sshd(ok)
```

Now put the principals in the "auth_principals" file in the .ssh of the user home folder. Now, user "ann" and "john" are happy and login as "root" on the postgresql server.

```
postgresql-root:/root/.ssh# vi auth_principals
ann
john
```

Using this method is a flexible way for giving "root" access or logging in as another user. We don't need to change the user certificate. We only add each user on a separate line in the principal file. We do it globally or in the user home folder. There is no best way but I prefer doing it more globally because this gives a centralized overview. Most of the time this method is sufficient for rolling out many cases.

We can extend this approach by using groups when making user certificates.

Suspose we have many sysadmins working on many different kind of mail servers.

```
exim
postfix
sendmail
qmail
```

2.9 With AuthorizedPrincipalsCommand

With the "AuthorizedPrincipalsFile" we create a static file. The same can be done in a dynamic way by using the "AuthorizedPrincipalsCommand" option. This option needs a program to run. This program must have an absolute path and only has user "root" permissions, no group or other permissions. (chmod 700 file.sh). The output of the program must be the same as we create with the "AuthorizedPrincipalsFile", a list of allowed certificate principals. This gives us a lot of possibilities. In the program, we can do ldap lookups, searching in a database, but be careful when using external resources. Doing this creates dependencies. Good luck when the LDAP server is down. We also need to specify "AuthorizedPrincipalsCommandUser", otherwise ssh won't start. The documentation arguments to use a user other than "root" for execution of the program but this doesn't work for me. We can specify arguments to use in the program. (man sshd_config)

- %% A literal ‘%’.
- %D The routing domain in which the incoming connection was received.
- %F The fingerprint of the CA key.
- %f The fingerprint of the key or certificate.
- %h The home directory of the user.
- %i The key ID in the certificate.
- %K The base64-encoded CA key.
- %k The base64-encoded key or certificate for authentication.
- %s The serial number of the certificate.
- %T The type of the CA key.
- %t The key or certificate type.
- %u The username.

AuthorizedPrincipalsCommand accepts following tokens %, %F, %f, %h, %i, %K, %k, %s, %T, %t, and %u. The most importance is “%u”. Whenever we change something in the “sshd_config” or in the program, we need to restart the “sshd” daemon. Let’s see how we implement this for our user “john” and “ann”. We create an ACL list file. When logging is as user “root”, only the user certificates with the principals “ann” and “john” can login. If we don’t find the user in the ACL file, means that the user is logging in as her/his user name. In that case we send the user name. Otherwise we must update our “auth_principals.acl” for each user who wants to login as her/his own account.

user	principal
root	ann
root	john

On the postgresql server it looks like this:

```
postgresql-root:/etc/ssh/ca# vi auth_principals.acl
#user      principal
root       ann
root       john
```

Change the “sshd_config” on the “postgresql” server. When somebody ssh to the server, the sshd executes the program “/etc/ssh/ca/auth_principals.sh” with the arguments “%u”, “%i” and “%h”. “%i” is what we see in the logging.

```
postgresql-root:/etc/ssh# vi sshd_config
#AuthorizedPrincipalsFile /etc/ssh/ca/%u_auth_principals
#AuthorizedPrincipalsFile .ssh/auth_principals
AuthorizedPrincipalsCommand /etc/ssh/ca/auth_principals.sh %u %h %i
AuthorizedPrincipalsCommandUser root
```

The “auth_principals.sh” accepts 3 parameters but we use only the first one, the user login name. The shell scripts executes a little awk script. It gets its input from “auth_principals.acl”. Lines starting with a “#” will be skipped. Then it checks the login user name with the first field in the “auth_principals.acl” file. If they match then the second field, the principal, will be written to the standard output. In the end, if nothing is written, then we write down the principal of the user name. Set the rights permissions on the “auth_*” files and restart the sshd daemon.

```
postgresql-root:/etc/ssh/ca# vi auth_principals.sh
#!/bin/ksh
# we get 3 parameters
# %u = username we login
# %h = home folder of the user
# %i = key ID of the user certificate
file="/etc/ssh/ca/auth_principals.acl"
/usr/bin/awk -v user=$1 '
BEGIN {count=0}
```

```

#!/ {next}
$1 ~ user {print $2; count=1}
END { if (count==0) {print user}}
' $file
postgresql-root:/etc/ssh/ca# ls -l auth*
-rw-r--r--  1 root  wheel   39 Jan 31 09:02 auth_principals.acl
-rwx-----  1 root  wheel  297 Jan 31 07:22 auth_principals.sh
postgresql-root:/etc/ssh/ca# rcctl restart sshd
sshd(ok)
sshd(ok)

```

Check if “john” can login as “root” and check the log file on the postgresql server.

```

mylaptop-john:/home/john$ ssh -l root postgresql
...
postgresql-root:/root#
postgresql-root:/root# tail -f /var/log/authlog
Jan 31 09:20:34 postgresql sshd[951]: Accepted publickey for root from 192.168.153.50 port 42237 \
  ssh2: RSA-CERT ID john (serial 0) CA RSA SHA256:G5vkBB/19wMN9D8z09n0vaxXCwQwgfedadAr9DpQ9Lw

```

The shell script can be extended, a few ideas:

- write information to a log file. I have no clue why but the log statements show duplicates in the log file.
- add a third column “yes” or “no” in the “auth_principals.acl” file if the user can authenticicated. This is a fast method for revoking a user.
- add more options to the “auth_principals.acl”.

2.10 Fine tuning privileges

In all the examples, the user “ann” and “john” have all the privileges when logging in. In most cases it’s ok. But sometimes we want to fine tune those privileges. In fact in all above cases, without knowing, we were creating an “authorized_keys” file format. For a complete reference take a look at the “sshd” man page, section “AUTHORIZED_KEYS FILE FORMAT”. Let’s make a “postgres” user on the postgresql server and a new user “accountant” on the laptop. On the “postgresql” server we need to run a special script “salary.sh” and only user “accountant” can run it with ssh.

```

postgresql-root:/root# useradd -m postgres
postgresql$ vi salary.sh
cat <<EOF
boss 7645
ann 4500
john 2877
EOF
postgresql$ chmod +x salary.sh

```

On our laptop:

```

mylaptop# useradd -m accountant
mylaptop# su - accountant
mylaptop# vi .profile
PS1="\h-\u:\$PWD$ "
mylaptop# exit
mylaptop# su - accountant
mylaptop-accountant:/home/accountant$ ssh-keygen -t rsa
...
mylaptop-accountant:/home/accountant/.ssh$ cat id_rsa.pub

```

On the sshca server. Copy the “mylaptop-accountant:/home/accountant/.ssh/id_rsa.pub” to “sshca-ca:/home/ca/users/accountant/id_rsa.pub”

```

sshca-ca:/home/ca/users$ mkdir accountant
sshca-ca:/home/ca/users$ cd accountant
sshca-ca:/home/ca/users/accountant$ vi 2019_id_rsa.pub
(copy mylaptop-accountant:/home/accountant/.ssh$ cat id_rsa.pub)

```

On the postgresql server disable following config parameters and restart the ssh daemon.

```
postgresl-root:/etc/ssh# cat sshd_config
...
HostCertificate /etc/ssh/ca/2018_ssh_host_rsa_key-cert.pub
TrustedUserCAKeys /etc/ssh/ca/2018-12-07-user-sshca-key.pub
#AuthorizedPrincipalsFile /etc/ssh/ca/%u_auth_principals
#AuthorizedPrincipalsFile .ssh/auth_principals
#AuthorizedPrincipalsCommand /etc/ssh/ca/auth_principals.sh %u %h %i
#AuthorizedPrincipalsCommandUser root
postgresl-root:/root# rcctl restart sshd
sshd(ok)
sshd(ok)
```

2.10.1 With principals

Create a user certificate for the user “accountant”. This time executes “ssh-keygen” with “-O” to set some options. With “-O clear”, we erase all the options. Then with “-O force-command” we add one option, execute the “salary.sh” command.

```
ssh-keygen -s /home/ca/user-sshca/2018-12-07-user-sshca-key \
-I "accountant" \
-n "accountant" \
-V -1w:+52w \
-O clear \
-O force-command="/home/postgres/salary.sh" \
/home/ca/users/accountant/2019_id_rsa.pub

sshca-ca:/home/ca/users/accountant$ ssh-keygen -Lf 2019_id_rsa-cert.pub
2019_id_rsa-cert.pub:
Type: ssh-rsa-cert-v01@openssh.com user certificate
Public key: RSA-CERT SHA256:jx5Z1Vc2buA4zs6MX0uMxSr84TkaQr3gwnQUbln39iM
Signing CA: RSA SHA256:G5vkBB/19wMN9D8z09n0vaxXCwQwgfedadAr9DpQ9Lw
Key ID: "accountant"
Serial: 0
Valid: from 2019-02-18T08:39:57 to 2020-02-24T08:39:57
Principals:
    accountant
Critical Options:
    force-command /home/postgres/salary.sh
Extensions: (none)
```

Don’t forget to update the “known_hosts” and/or the “.profile” file for the user “accountant”. Then we make a connection to the postgresql server with the user “postgres”.

```
mylaptop-accountant:/home/accountant$ ssh -l postgres postgresl
PTY allocation request failed on channel 0
boss 7645
ann 4500
john 2877
Connection to postgresql closed.
```

How about scp?

```
mylaptop-accountant:/home/accountant$ touch aa
mylaptop-accountant:/home/accountant$ scp aa postgres@postgresl:
boss 7645
mylaptop-accountant:/home/accountant$
```

The file “aa” won’t be copied to the postgresql server. What we get back is the first line of the “salary.sh” program. No idea why.

Privileges can be set on different places but if we want to be absolute sure, put it in the certificate. It overrules all other possibilities. This is also the safest way to give external persons, who work for a short period, some privileges.

2.10.2 With AuthorizedPrincipalsFile

In 2.8.2 we use the “AuthorizedPrincipalsFile” option. Let’s change “sshd_config” file, enable this option again. In the “/etc/ssh/ca/postgres_auth_principals” file, we specify the principals who have access to the postgres user. There is only one line with the content “accountant”. Restart the ssh daemon.

```
postgresql-root:/etc/ssh# vi sshd_config
...
HostCertificate /etc/ssh/ca/2018_ssh_host_rsa_key-cert.pub
TrustedUserCAKeys /etc/ssh/ca/2018-12-07-user-sshca-key.pub
AuthorizedPrincipalsFile /etc/ssh/ca/%u_auth_principals
#AuthorizedPrincipalsFile .ssh/auth_principals
#AuthorizedPrincipalsCommand /etc/ssh/ca/auth_principals.sh %u %h %i
#AuthorizedPrincipalsCommandUser root
postgresql-root:/etc/ssh/ca# vi postgres_auth_principals
accountant
postgresql-root:/root# rcctl restart sshd
sshd(ok)
sshd(ok)
```

Nothing is changed for the user “accountant”.

```
mylaptop-accountant:/home/accountant$ ssh -l postgres postgresql
PTY allocation request failed on channel 0
boss 7645
ann 4500
john 2877
Connection to postgresql closed.
```

Let’s redo the user “accountant”. We first need to remake the certificate.

```
sshca-ca:/home/ca/users/accountant$ mv 2019_id_rsa-cert.pub 2019_id_rsa-cert.command
sshca-ca:/home/ca/users/accountant$ ssh-keygen -s /home/ca/user-sshca/2018-12-07-user-sshca-key \
-I "accountant" \
-n "accountant" \
-V -1w:+52w \
/home/ca/users/accountant/2019_id_rsa.pub
```

And copy the contents from “sshca:/home/ca/accountant/2019_id_rsa.pub” to “mylaptop-accountant:/home/accountant/.ssh/id_rsa-cert.pub”.

```
mylaptop-accountant:/home/accountant/.ssh$ mv id_rsa-cert.pub id_rsa-cert.pub.command
mylaptop-accountant:/home/accountant/.ssh$ vi id_rsa-cert.pub
....
```

When somebody wants to login as user “postgres”, we check “/etc/ssh/ca/postgres_auth_principals” file. %u is substitute with “postgres”.

```
postgresql-root:/etc/ssh/ca# vi postgres_auth_principals
command="/home/postgres/salary.sh" accountant
```

Which options can be use in the “/etc/ssh/ca/postgres_auth_principals” file? Check the documantion with “man sshd” and the section “AuthorizedKeysFile”. The documentation is not 100% clear. Most options can be used. Separate the options with a comma, not with a space and the last field is the principal. There is only one principal.

```
command="/home/postgres/salary.sh",from="192.168.153.50" accountant
command="/home/postgres/salary.sh",no-agent-forwarding,no-port-forwarding,pty,no-X11-forwarding accountant
```

This works but is not 100% correctly. Ssh won’t give an error, it looks like the other options after a comma are neglected.

```
command="/home/postgres/salary.sh" from="99.99.99.99" accountant
command="/home/postgres/salary.sh" action="do_it_slow" accountant
```

Also the strange output when doing a scp.

```
mylaptop-accountant:/home/accountant$ scp aa postgres@postgresql:
boss 7645
mylaptop-accountant:/home/accountant$
```


2.10.3 With AuthorizedPrincipalsCommand

Again, we first need to change the “sshd_config” file on the “postgres” server to enable this option. Restart the daemon.

```
postgresl-root:/etc/ssh# vi sshd_config
...
HostCertificate /etc/ssh/ca/2018_ssh_host_rsa_key-cert.pub
TrustedUserCAKeys /etc/ssh/ca/2018-12-07-user-sshca-key.pub
#AuthorizedPrincipalsFile /etc/ssh/ca/%u_auth_principals
#AuthorizedPrincipalsFile .ssh/auth_principals
AuthorizedPrincipalsCommand /etc/ssh/ca/auth_principals.sh %u %h %i
AuthorizedPrincipalsCommandUser root
postgresl-root:/root# rcctl restart sshd
sshd(ok)
sshd(ok)
```

We need to change the “auth_principals.acl” file with a third option. When logging in as user “postgres” and the certificate has principal “accountant”, which is the user “accountant”, then only the “salary.sh” script can be executed.

```
postgresl-root:/etc/ssh/ca# vi auth_principals.acl
# user      principal      options
postgres    accountant    command="/home/postgres/salary.sh"
#root       ann
root        john
```

The only thing to do is to change one line in the “auth_principals.sh” script to print the options. ($\$1 \sim \text{user}$ {print $\$3$ " $\$2$; count=1})

```
postgresl-root:/etc/ssh/ca# vi auth_principals.sh
#!/bin/ksh
# we get 3 parameters
# %u = username we login
# %h = home folder of the user
# %i = key ID of the user certificate
file="/etc/ssh/ca/auth_principals.acl"
/usr/bin/awk -v user=$1 '
BEGIN {count=0}
/##/ {next}
$1 ~ user {print $3 " " $2; count=1}
END { if (count==0) {print user}}
' $file
```

That’s all and user “accountant” is happy again.

```
mylaptop-accountant:/home/accountant$ ssh -l postgres postgresl
boss 7645
ann 4500
john 2877
Connection to postgresql closed.
```

2.11 Revoking keys

We did a long story, with all kind of possibilities for creating users with principals. But what happens when people leave the organisation, when somebody’s laptop is stolen or we stop trusting the host. There are multiple ways for revoking user and host keys.

2.11.1 Revoking Users keys

We only need to change the option “RevokedKeys” in the “sshd_config” file. Let’s take a close look how this is done for the users keys. On the “postgres” server we have 3 users, “accountant”, “ann” and “john”. First we prepare the server for revoking users keys and then revoke the key of “ann”.

First, we add the “RevokedKeys” option to the “sshd_config” file and restart the sshd daemon.

```
postgresql-root:/etc/ssh# vi sshd_config
...
RevokedKeys /etc/ssh/ca/RevokedKeys
postgresql-root:/etc/ssh# rcctl restart sshd
```

First check of the user “ann” can still login from her laptop into the postgresql server.

```
mylaptop-ann:/home/ann$ ssh postgresql
no such identity: /home/ann/.ssh/id_xmss: No such file or directory
ann@postgresql's password:
```

It is important that the “RevokedKeys” file exists and that it is readable. Otherwise nobody can log in. Just, create the file. No need to restart the daemon.

```
postgresql-root:/etc/ssh/ca# > RevokedKeys
```

Now, we put the public key or the certificate key of the user “ann” is this file. That’s why it is important to keep those files on the “sshca-ca” server. Check the public keys of user “ann” on the “sshca” server and copy the contents of the “2019_id_rsa-cert.pub” or “2019_id_rsa.pub” to the “postgresql-root:/etc/ssh/ca/RevokedKeys”.

```
sshca-ca:/home/ca/users/ann$ ls -l
2019_id_rsa-cert.pub
2019_id_rsa.pub
```

On the postgresql server. For each user, we add one new line.

```
postgresql-root:/etc/ssh/ca# vi RevokedKeys
ssh-rsa AAAAB3NzaC1yc2EAAAADA...
```

And now user “ann” can not login anymore. A password prompt appear.

```
mylaptop-ann:/home/ann$ ssh postgresql
no such identity: /home/ann/.ssh/id_xmss: No such file or directory
ann@postgresql's password:
```

Let’s check the log file on the postgresql server

```
postgresql-root:/root# tail -f /var/log/authlog
...
Mar 13 12:59:53 postgresql sshd[42514]: error: Authentication key RSA SHA256:sik.. revoked by file /etc/ssh/ca/RevokedKeys
Mar 13 12:59:53 postgresql sshd[42514]: error: Authentication key RSA-CERT SHA256:si.. revoked by file /etc/ssh/ca/RevokedKeys
```

That is the fingerprint of the user “ann”. We can check this on the ssh-ca server.

```
sshca-ca:/home/ca/users/ann$ ssh-keygen -lf ./2019_id_rsa.pub
2048 SHA256:sikW0Xz8sRhKoz1c3K/HGCxHkkw4
```

The “RevokedKeys” is a text file. This file would become big. It is possible to compact this file and make a binary format, a Key Revocation List (KRL). We generate this file with “ssh-keygen” with the “-k” option. We make a new folder on the sshca server.

```
sshca-ca:/home/ca$ mkdir revoke
```

There are a few possibilities how we make a KRL file. Check the man page of “ssh-keygen” in the section “KEY REVOCATION LISTS”. The basic command is:

```
ssh-keygen -k -f krl_file [-u] [-s ca_public] [-z version_number] file ...

-k                Create or update a KRL file
-f                The KRL file
-u                Update/merge the KRL
```

```
-s ca_public          This is the CA public key. We created two CA some time ago. In our case:
                      sshca-ca:/home/ca/user-sshca/2018-12-07-user-sshca-key.pub
                      sshca-ca:/home/ca/host-sshca/2018-12-07-host-sshca-key.pub
                      Some more investigations has to be done with this parameter.
                      We can use whatever key we want, private, public. ssh-keygen try to revoke the key.

-z version_number    Add the signal number to the KRL
```

Let's revoke all the keys from user "ann" and "john".

```
sshca-ca:/home/ca/revoke$ ssh-keygen -k -f RevokedKeys /home/ca/users/ann/*.pub /home/ca/users/john/*.pub
Revoking from /home/ca/users/ann/2019_id_rsa-cert.pub
Revoking from /home/ca/users/ann/2019_id_rsa.pub
Revoking from /home/ca/users/john/2018_id_rsa-cert.pub
Revoking from /home/ca/users/john/2018_id_rsa.pub
sshca-ca:/home/ca/revoke$ ls
RevokedKeys
```

Now, we need to copy the "RevokedKeys" binary file to the postgresql server into the folder "/etc/ssh/ca". I copy this file through my real laptop with has the name "gerrit_laptop".

```
sshca-ca:/home/ca/revoke$ cp RevokedKeys /tmp
gerrit_laptop:Downloads $ scp -P 2501 captain@localhost:/tmp/RevokedKeys .
captain@localhost's password:
RevokedKeys
gerrit_laptop:Downloads $ scp -P 2502 RevokedKeys captain@localhost:/tmp
captain@localhost's password:
RevokedKeys
postgresql-root:/etc/ssh/ca# mv /tmp/RevokedKeys .
postgresql-root:/etc/ssh/ca# chown root RevokedKeys
```

Both the user "ann" and "john" can not login.

```
mylaptop-ann:/home/ann$ ssh postgresql
no such identity: /home/ann/.ssh/id_xmss: No such file or directory
ann@postgresql's password:
mylaptop-john:/home/john$ ssh postgresql
no such identity: /home/john/.ssh/id_xmss: No such file or directory
john@postgresql's password:
```

How do we check if a key is revoked? With the "ssh-keygen -Qf "

```
sshca-ca:/home/ca/revoke$ ssh-keygen -Qf RevokedKeys ../users/john/*.pub ../users/accountant/*.pub
../users/john/2018_id_rsa-cert.pub (john@mylaptop.yerrysherry.be): REVOKED
../users/john/2018_id_rsa.pub (john@mylaptop.yerrysherry.be): REVOKED
../users/accountant/2019_id_rsa-cert.pub (accountant@mylaptop.yerrysherry.be): ok
../users/accountant/2019_id_rsa.pub (accountant@mylaptop.yerrysherry.be): ok
```

How do we check all users and hosts?

```
sshca-ca:/home/ca/revoke$ ssh-keygen -Qf RevokedKeys ../hosts/*/*.pub ../users/*/*.pub
...
```

The first time we create the KRL. The next time use the "-u" option the append/merge the KRL file. For the user "accountant"

```
sshca-ca:/home/ca/revoke$ ssh-keygen -ku -f RevokedKeys /home/ca/users/accountant/*.pub
Revoking from /home/ca/users/accountant/2019_id_rsa-cert.pub
Revoking from /home/ca/users/accountant/2019_id_rsa.pub
```

An other possibility is creating a file. In this file we specify which keys we want to revoke. We can use different kind of keywords like serial, id, key, sha1, sha256, hash. Check the documentation, we need OpenSSH version 7.9 for using sha256 or hash keyword. Then we specify some information how we want to revoke this key. Let's have an example. We create the file "ids-to-revoke" but first let's check again the certificate of user "ann".

```
sshca-ca:/home/ca/users/ann$ ssh-keygen -Lf 2019_id_rsa-cert.pub
2019_id_rsa-cert.pub:
    Type: ssh-rsa-cert-v01@openssh.com user certificate
    Public key: RSA-CERT SHA256:sikW0Xz8sRhKoz1c3K/HGCxHkkw42+SEhGm9pzav5u8
    Signing CA: RSA SHA256:G5vkBB/19wMN9D8z09n0vaxXCwQwgfedadAr9DpQ9Lw
    Key ID: "ann"
    Serial: 0
    Valid: from 2018-12-10T18:19:22 to 2019-12-16T18:19:22
    Principals:
        ann
        root
    Critical Options: (none)
    Extensions:
        permit-X11-forwarding
        permit-agent-forwarding
        permit-port-forwarding
        permit-pty
        permit-user-rc
```

Now, it becomes clear which information we can use. For “id:”, we use the “Key ID” of the certificate. The “key:” is the CA certificate of the user “john”.

```
sshca-ca:/home/ca/revoke$ vi ids-to-revoke
id: ann
id: accountant
key: ssh-rsa-cert-v01@openssh.com AAAAHNNzaC1yc2EtY2...
```

Now, we create a new KRL.

```
sshca-ca:/home/ca/revoke$ ssh-keygen -k -f RevokedKeys -s /home/ca/user-sshca/2018-12-07-user-sshca-key ids-to-revoke
Revoking from ids-to-revoke
```

Let’s check what we revoke.

```
sshca-ca:/home/ca/revoke$ ssh-keygen -Qf RevokedKeys ../users/*/*pub
../users/accountant/2019_id_rsa-cert.pub (accountant@mylaptop.yerrysherry.be): REVOKED
../users/accountant/2019_id_rsa.pub (accountant@mylaptop.yerrysherry.be): ok
../users/ann/2019_id_rsa-cert.pub (ann@mylaptop.yerrysherry.be): REVOKED
../users/ann/2019_id_rsa.pub (ann@mylaptop.yerrysherry.be): ok
../users/john/2018_id_rsa-cert.pub (john@mylaptop.yerrysherry.be): REVOKED
../users/john/2018_id_rsa.pub (john@mylaptop.yerrysherry.be): REVOKED
```

Now, we copy the “RevokedKeys” file to the postgresql server. Most users keys are revoked. Not all. But the user “ann”, “john” and “accountant” can not login.

Serial Number

We create the user “ann” certificate with the following command and without a serial number.

```
sshca-ca:/home/ca/users/ann$ \
ssh-keygen -s /home/ca/user-sshca/2018-12-07-user-sshca-key \
-I "ann" \
-n "ann,root" \
-V -1w:+52w \
/home/ca/users/ann/2019_id_rsa.pub
```

It is possible to give a serial number to the certificate. It is not a good idea to give number + 1. Better is to create a random number, like:

```
$ echo 'ann' | cksum | cut -f 1 -d ' '
3476301178
```

Find an other way if the number is not big enough.

```
sshca-ca:/home/ca/users/ann$ \
ssh-keygen -s /home/ca/user-sshca/2018-12-07-user-sshca-key \
           -z 3476301178 \
           -I "ann" \
           -n "ann,root" \
           -V -1w:+52w \
           /home/ca/users/ann/2019_id_rsa.pub
```

Now, it is easier to revoke this certificate.

```
sshca-ca:/home/ca/revoke$ vi ids-to-revoke
serial: 3476301178
```

Pity, we only can use one “RevokedKeys” option in the `sshd_config` file. Otherwise we could use, for example one for dba, another for finance. Another problem is when KRL files are spread out on our servers, it is difficult to list which keys are revoke. Keep the KRL as small as possible and make sure you have the latest text file for making the KRL. This is also a reason for not signing keys too far into the future and regularly updating them.

2.11.2 Revoking hosts keys

The “RevokedKeys” option only works for users. In `sshd_config`, we can’t revoke hosts keys. Therefore we need to find a solution outside ssh. We have 2 options for blocking incoming ip addresses from other hosts. A firewall and TCP wrappers. The former we find in all unix OS but the latter it depends. In OpenBSD we don’t have TCP wrappers anymore but in FreeBSD, NetBSD we still do. So, check your OS for implementing blocking ip addresses.

2.12 Disable authorized_keys

If all your users authenticate with a CA then there is no need to use the “authorized_keys” file anymore. Change the “AuthorizedKeysFile” setting to “none” in the “sshd_config” file and restart the sshd daemon.

2.13 Conclusion

This is a long story for setting up a Certificate Authority (CA). You should have enough information for setting it up in production. Of course it would be crazy to set it up by hand. You need some kind of configuration management tool. There are many of them. Choose the one you most like, Ansible, Puppet are the best I know.

2.14 Resources

<https://code.fb.com/production-engineering/scalable-and-secure-access-with-ssh/>
<https://www.youtube.com/watch?v=exjorJyddok>
<https://ef.gy/hardening-ssh>
<https://www.digitalocean.com/community/tutorials/how-to-create-an-ssh-ca-to-validate-hosts-and-clients-with-ubuntu>
<https://blog.habets.se/2011/07/OpenSSH-certificates.html>
<https://framkant.org/2016/10/setting-up-a-ssh-certificate-authority-ca/>
<https://framkant.org/2017/07/scalable-access-control-using-openssh-certificates/>
<https://serverfault.com/questions/894507/how-to-give-temporary-access-with-ssh-using-certificate-authority>
<https://www.lorier.net/docs/ssh-ca.html>
<http://unixseclab.com/index.php/2017/01/09/ssh-start-to-finish-certificate-authority-basics/>
<http://unixseclab.com/index.php/2017/01/16/ssh-start-to-finish-architecture-standing-up-the-ca/>
<http://unixseclab.com/index.php/2017/01/23/ssh-start-to-finish-architecture-standing-up-the-ca-pt-2/>
<https://docs.fedoraproject.org/en-US/fedora/f28/system-administrators-guide/infrastructure-services/OpenSSH/>

<https://medium.com/leboncoin-engineering-blog/cassh-ssh-key-signing-tool-39fd3b8e4de7>

https://github.com/vedetta-com/vedetta/blob/master/src/usr/local/share/doc/vedetta/OpenSSH_Principals.md

<https://utcc.utoronto.ca/~cks/space/blog/sysadmin/SSHBroadKeyRevocation>

<https://www.lorier.net/docs/ssh-ca.html>

https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication#Mark_Public_Keys_as_Revoked

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/sec-revoking_an_ssh_c

https://www.ibm.com/support/knowledgecenter/en/SSLTBW_2.3.0/com.ibm.zos.v2r3.foto100/krl.html

https://docs.fedoraproject.org/en-US/Fedora/23/html/System_Administrators_Guide/sec-Revoking_an_SSH_CA_Certificate

https://en.wikibooks.org/wiki/OpenSSH/Cookbook/Public_Key_Authentication#Mark_Public_Keys_as_Revoked

Chapter 3

PostgreSql

```
checkpoint -> true in volgend backup
pg_start_backup('my_backup',true)
start backup
pg_stop_backup()
alter system set full_page_write=off;
zfs set compression=lz4 rpool
zfs set atime=off
zfs set recordsize=16K, door compressie
zfs set primarycache=metadata/all
zfs send -v -L -p -e pool | zfs receive -v pool
zfs get receive_resume_token
zfs snapshot pool
zfs send -v -L -p -e -i pool snapshot | zfs receive -v pool
zfs set logbias=throughput, txg 1 second
cat /sys/module/zfs/parameters/zfs_txg_timeout
echo 1 > /sys/module/zfs/parameters/zfs_txg_timeout
echo 'options zfs zfs_txg_timeout=1' >> /etc/..
alter system set synchronous_commit=off
zfs set logbias=throughput
https://jrs-s.net
```

Chapter 4

zfs snaphost

Als je een snapshot verwijdert maak een bookmark. `bsd now` 272, 214

<https://github.com/yboetz/pyznap>

<https://github.com/jimsalterjrs/sanoid>

<https://github.com/zfsonlinux/zfs-auto-snapshot>

<https://github.com/ewwhite/zfs-ha/wiki>