Bachelor's thesis

Information Technology

Internet Technology

2016

Hugo Sastre

UNIX secure server

– a free, secure, and functional server example

BACHELOR'S THESIS │ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information Technology | Internet Technology

2016 | 38

Patric Granholm

Hugo Sastre

UNIX secure server

The purpose of this thesis work was to introduce UNIX server as a personal server but also as a start point for investigation and developing at a professional level.

The objective of this thesis was to build a secure server providing not only a FTP server but also an HTTP server and a cloud system for remote backups. OpenBSD was used as the operating system.

OpenBSD is a UNIX-like operating system made by hackers for hackers. The difference with other systems that might partially provide solutions is that OpenBSD requires a fully understanding of the configuration files in order to cover all the security holes, therefore, is needed to go through the documentation and examples of these files.

As a result we have a fully functional server with the next following features: anonymous FTP server, HTTP server including PHP, SSH server and of course a backup system in a cloud with cryptography enabled.

KEYWORDS:

UNIX, OpenBSD, Web, FTP, Security, OpenSSH, LibreSSL

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

CONTENTS

1 INTRODUCTION

Since Edward Snowden leaked classified information about global surveillance programs run by the NSA and the PRISM program, security and privacy have become a very important issue.

It looks almost impossible to stay away from surveillance, especially now that large companies and tools are compromised. OpenBSD is developed all over the world but its headquarters are located in Canada where it is possible to export cryptography within an operating system. In countries like the USA cryptography is considered a weapon by the law and has several restrictions.

The next sections discuss how to configure a secure server from scratch providing basic services as file transfer protocol (FTP), a web server, cryptography with LibreSSl and basic notions for backup and restore files.

The hardware used is a Lenovo ThinkPad T400, a regular internet connection and an external hard drive for the backups.

The software is the last stable OpenBSD, starting with version 5.7 and updated to 5.8 during the process.

An account on dot tk (http://www.dot.tk/en/index.html) was used in order to obtain the domains and subdomains used on the project.

As a result, there is an anonymous FTP server for sharing files and software, a ssh server for secure communications, a HTTP server with php enabled for dynamic contents and a cloud for remote backups. This server also provides secure connections with LibreSSL instead of OpenSSL for really secure cryptography.

2 THE OPENBSD SERVER

What is a server good for? Why are they needed? These are good questions and very difficult to answer. One of the more accurate answers may be “it depends on the situation”. However what is for sure is that the internet and the networks as we conceive them today would not be possible without servers. The act of making a query in a browser involves a server to provide the information required. Servers are needed to provide websites, to organize companies, for building applications, and many more functions.

Etymologically a server's task is to serve. If we are talking about a computer a good start would be doing it in a **secure** way, **functionality** would also be a desirable feature and the only way to achieve both is by **freedom**. At this point, trying to define freedom would be an act of futility. Fortunately this is about machines, software, and/or the combination of both, therefore the meaning of freedom in this context becomes more tangible. However, in English, the word “free” could be understood in economic terms of no cost or the possibility to be used in any way. Due to this ambiguity many hardware and software developers are using the Spanish word “**libre**” to explicitly refer to the second meaning of “free”.

Going deeper into the meaning of the concept lead us to the last term: **open**.

Applied to software, and even to hardware, an open system allows access to the source code in order to understand how it works. It is also possible to modify it according to the needs or desires of the user and share with other users the new version or “recipe”.

At this point we are talking about licenses and there is a great deal of them out there. As a starting point, we can describe the most famous of the permissive type licenses as:

- **GPL** and compatibles: grants all the rights, read, study, modify and share but is **viral** meaning that it will put any project, where GPL is included, under the same license. This is a good method for guaranteeing the future of the license and the software but scares possible investors or companies interested in these projects.

- **BSD** and compatibles: grants all the rights, read, study, modify and share but is **not viral**. This can be interpreted as an even freer level and which causes heated debates about this subject. In any case the software under this license can be used for any purpose.

In short, BSD License is a gift. The BSD community do not care what developers do with the code, but developers should not claim it is theirs. This might look obvious for the most of us but many companies had problems understanding that. In the words of Michael Lucas (2013), author of Absolute OpenBSD. "A lot of people have a very strong view points on things like Microsoft took the BSD tcp/ip stack and put in Windows back in the Windows 95 days […] but turn this around, imagine the cost in human suffering if Microsoft have written their own tcp/ip stack". This quotation is a perfect example of the advantages of Open Software even for a company like Microsoft which makes business with private software.

We are lucky that these terms have been invented and implemented now. The consequences of copyrights and licenses in a previous age, say the age of Newton and Leibniz, could have been devastating for our present and future. After all calculus could be interpreted as a piece of software in today's terms.

At the time this document was written the European Parliament had started to take seriously the idea of financing open source tools in order to protect the citizens instead of taking part in the Mass Surveillance game. OpenBSD is one of the three open source operating systems proposed. The other two are Qubes and TAILS. (Gamino et al. 2015, pp 52-53)

After this small briefing about licenses and servers lines there is no need to explain what OpenBSD stands for.

OpenBSD is a free “unix-like” operating system like Linux, Solaris or Mac OS among others. The BSD project is a straight descendant from the AT&T UNIX (tm) that originally was a closed source but in the late 70's it was gradually opened by the students of the University of California, Berkeley. Hence BSD stands for Berkeley Software Distribution. The main BSD variants are FreeBSD, NetBSD and OpenBSD among others. An important difference with another operating systems is that most of BSD systems generally use monolithic kernels so in our case OpenBSD is a complete operating system and not a distribution as most of the Linux are.

OpenBSD has the reputation of emphasizing on correctness, meaning “do it right or do not do it at all” and well-documented software therefore OpenBSD of manuals are very easy to follow and understand to the point that lack of this information is considered a bug and it has to be corrected in order to fix it. OpenBSD delivers a system with no security holes in the default install. Naturally, they are not responsible of all the 3rd party software that could be installed on it later, and “only two remote holes in the default install, in a heck of a long time!” according to their own website (OpenBSD, 2015).

As this thesis is about building a server based on OpenBSD and there is no need to install 3rd part applications on it for the services we need we can say that we are dealing with a very secure server by default. However, once we start to allow services, it is important to follow a good method and understand the configuration on which all the security and functionality of the project relies.

It has been said that OpenBSD is written by hackers for hackers but it is more precise to say that OpenBSD is written for the people who write it, who are basically system administrators who know what are they talking about. OpenBSD community claims that they “eat their own dog food” and they are very proud of running the system on their personal computers.

Another important issue is “correctness” in the OpenBSD project documentation and accuracy gets a real dimension to the point that there is people working and testing that all the documentation and the manual pages describe exactly how the programs work and there is no lack of information. Therefore even a masterpiece of code will not be approved and included in the system unless it is well-documented. The aim is not to need external help for any installation and the programs have to behave exactly as described in the instructions.

OpenBSD includes a manual with instructions to use all the programs and configuration files. The way to access the manual is to open a terminal and type “man” and the command or configuration file we want to consult.

3 HARDWARE SPECIFICATIONS

The selection of the machine is a very important link in the process of building a server. This is a good point for a small briefing about **hardware** and **software** and their relationship. The first one is easy to understand hardware is the physical part of the machine. In our previous example with pencils the hardware would be the pencil itself with its physical characteristics like length. The software is the programs that run on the hardware and these programs are methods of use, for example, the “recipe” for using the pencil as a hair stick. Normally the physical tools need a set of instructions. Even in our simple example a pencil could be shipped with a set of instructions for an optimal use. It could be about some specific compositions of the core, maybe the proportions of the graphite and clay so the pencil is designed for specific tasks or materials to work on it. Some pencil vendors provide this set of information. When we talk about computers this piece of information is called **firmware.** The firmware is a set of instructions for an optimal performance of the hardware. Because the firmware is provided by the vendors, it is their decision to provide **open firmware** or not. The reasons could be to hide the wonderful features of their hardware from the competitor companies, but it has been proved and admitted that sometimes the hardware has other instructions besides the main instructions, as in the case of “vulnerabilities” found in the BIOS of Lenovo laptops (Lenovo 2015).

OpenBSD runs in a large amount of hardware but they do not accept any firmware from the vendors that is not open. In case the companies do not provide open firmware they will use the open version of this firmware that has been “reverse engineered” to provide open instructions to that specific piece of hardware.

For a better experience, it is a good idea to choose hardware with open firmware.

In their website (OpenBSD 2015), OpenBSD list all the supported platforms. At the moment of writing this thesis (2016) the following architectures are being supported:

alpha, amd64, armish, hppa, i386, landisk, loongson, luna88k, macppc, octeon, sgi, socppc, sparc, sparc64, vax and zaurus.

The aim of OpenBSD is to support more platforms in the future if the open hardware requests are met.

In our case, we used a Lenovo ThinkPad version "7UET94WW (3.24 Intel(R) Core(TM)2 Duo CPU P8600 @ 2.40GHz, 2394.40 MHz)", 4 Gb RAM. As a home server, it is powerful enough. In addition, a laptop with a functional battery could be used as a UPS (uninterruptible power supply).

A simple script checks the status of the A/C adapter state and power off the machine when it is not connected to the power supply and the battery level is lower than 30%.

```
#!/bin/sh -

if [ `apm | grep “A/C adapter state” | awk -F”:” ' { print $2 }'` != “activated”] &&
[`apm | grep “Battery state” | awk -F”:” ' { print $2 }'` == “critical”]

        shutdown -h now

fi
```

This could be running in the cron of the server every 30 minutes avoiding power failures and the consequences of a sudden shut down.

If the power supply would be a critical point would be wise to write a script that sends a mail to the sysadmin in case of failure with a script like this:

```
#!/bin/sh -

if [ `apm | grep “A/C adapter state” | awk -F”:” ' { print $2 }'` != “activated”]

        mail -s “server power failure” sysadmin@domain.com        fi
```

4 SERVER SETUP

It has been already mentioned that OpenBSD puts a great effort on correctness and well-documentation, therefore, the FAQ of the project provides very accurate information for a standard installation.

4.1 Installation

What we need for an installation from scratch is a machine; internet connection would be desirable but not essential and the installation software. The last one could be a CD or a USB among others.

The best way to obtain an OpenBSD release and support the project is to buy a 3 CD set. Another way is to download it via HTTP/FTP. Choosing a close mirror like this can take a few minutes.

First we need to know the architecture of our hardware and choose an image from the mirror. It is a good idea to take a look at the README of the target architecture hosted in the FTP of each architecture for specific instructions.

If the plan is to install it from a CD there is two options: the file “install57.iso” or “cd57.iso”; the first option has all the code and the second one only the essential code for booting from the CD and fetching the source code from internet using FTP or HTTP. The first option grants the whole operating system for those cases when it is not possible to have the machine connected.

Lately the tendency of the hardware is to quit using the CD so it is also possible to build a boot USB. The easiest way is to get the file “install57.fs” from the same directory and use the utility “dd” in any Unix or Unix-like computer. This is as simple as this line script:

```
# dd if=”path to the install57.fs” of=”path to the usb stick”
```

It is very important to identify correctly the path to the USB, a confusion here will carry catastrophic results as the permanent deletion of the main hard drive!

Once the boot CD or USB is ready, we only have to restart the machine and change the BIOS to start from the CD drive or an external USB according to our choice.

The OpenBSD team have developed a very easy installation that goes over all relevant options like root password, adding users, partitioning of the hard drives and network configuration.

The upgrade process follows the same principles.

4.2 Partitioning

Partitioning a hard drive is always a good idea in order to improve the security and the speed of a hard drive versus a one partition installation. Luckily in this case OpenBSD automatically creates a partition according to the size of the machine hard drive and the RAM memory.

OpenBSD still needs a swap partition despite the fact that other Unix-like operating systems claim that is not necessary nowadays because it is supported to do the swapping to files.

In our case the use of a swap partition is a very interesting feature because it is a place where the kernel can store a copy of what is in core in the event of a system panic for later analysis. OpenBSD does this automatically if the swap partition is at least as large as the RAM and at least the same free space in the /var partition. OpenBSD will save a copy of the **dump** in /var/crash partition to assure this feature will work. The best practice would be to create a separated partition for /var/crash as large as the RAM.

A directory to take into special consideration is /tmp. Here is where all the temporal files are stored. It is world-writable therefore, it has its own partition. Depending on the use of the machine, it might need a big /tmp partition though most of the systems would manage with a very modest amount of storage (30M). For the sake of the life of the hard drive, it is a very good idea to mount this partition on the RAM memory. After all, the /tmp directory is purged at least every time the machine goes down so this place should not be used for long term storage. In this case the size of the /tmp could be around half of the RAM memory. This can be done by adding a line on the "/etc/fstab", the file where the partitions are defined after we have finished the installation:

```
swap /tmp tmpfs rw,nodev,nosuid,-s=1.8G 0 0
```

Other potential partitions to have under consideration are:

- /var/log: for the system logs
- /var/www: for the web server

In case we want a partition to keep over updates, it is a good idea to create it. For example if we are planning to have a data base server, say postgresql, we can create a "/var/postgresql" partition. That will make the backup tasks and the data preservation over updates very easy.

The /home partition is the place where the potential users of the system have their home directory with their own configuration files. This is the perfect example of the need for a partition in order to preserve those files over new installations. This can simply be done by ignoring this partition on the installation and mounting it manually after that. Everything would remain the same for the users.

OpenBSD provides an optional partition called "/altroot". The idea is to have a copy of the kernel and the /etc configuration files so that in case of the root (/) partition failure, it would be possible to gain access to the system and recover it. Obviously this partition would look nicer in a secondary hard drive. I also has to be at least as large as the root partition. To make this backup automatic the "/altroot" has to be created during the installation but never mounted. The "fstab" should have a line like this:

```
/dev/sd0d /altroot ffs xx 0 0
```

Moreover, the environment variable ROOTBACKUP must be set. For example, the following can be added to “/etc/daily.local”:

```
ROOTBACKUP=1
```

Of course, it is possible to create custom partitions for our own purposes or interests, like one with the name of the company or for some specific software we are planning to build or store there. Again, the advantage of this will be to make backups easier and a better performance in general but also add specific characteristics for that partition as making it read-only.

The characteristics of the partitions can be altered after the installation on the “/etc/fstab” file but for this to work the partitions must exist, therefore they should be created in advance.

4.3 Networking

The first step is to have a look at the manual for two programs:

- netstat(1): which shows the network status.
- ifconfig(8): which configures network interface parameters.

With the output of ifconfig(8), it is possible to have an idea of the network devices:

```
$ ifconfig
lo0: flags=8049<UP,LOOPBACK,RUNNING,MULTICAST> mtu 33200
        priority: 0
        groups: lo
        inet6 ::1 prefixlen 128
        inet6 fe80::1%lo0 prefixlen 64 scopeid 0x3
        inet 127.0.0.1 netmask 0xff000000
fxp0: flags=8843<UP,BROADCAST,RUNNING,SIMPLEX,MULTICAST> mtu 1500
        lladdr 00:04:ac:dd:39:6a
        priority: 0
        media: Ethernet autoselect (100baseTX full-duplex)
        status: active
        inet 10.0.0.38 netmask 0xffffff00 broadcast 10.0.0.255
        inet6 fe80::204:acff:fedd:396a%fxp0 prefixlen 64 scopeid 0x1
enc0: flags=0<>
        priority: 0
        groups: enc
        status: active
pflog0: flags=141<UP,RUNNING,PROMISC> mtu 33200
        priority: 0
        groups: pflog
```

In the example there are four network interfaces: a loopback (lo0), a physical ethernet card (fxp0), an encapsulating interface (enc0) and the packet filter logging interface (pflog0) for the OpenBSD firewall.

The hostname.if(5) is the interface-specific configuration file and should be created in /etc. We will create as many files as interfaces needed. An example would be:

```
$ cat /etc/hostname.fxp0
inet 10.0.0.38 255.255.255.0 NONE
```

To set the default gateway, we can create a file called /etc/mygate. It will be only one line with the gateway address.

The /etc/resolv.conf is the file to configure the DNS resolution. An example for a this file can be found in the manual of resolv.conf(5). The typical attributes of this file are:

- search: search list for hostname lookup.

- nameserver: IPv4 address (in dot notation) or IPv6 address (in hex-and-colon notation) of a name server that the resolver should query.

The last step is to set the host name under /etc/myname. Here is where the name of the machine and the domain are specified.

All these changes will be effective when rebooting the machine or running the netstart script as root:

```
# sh /etc/netstart

writing to routing socket: File exists

add net 127: gateway 127.0.0.1: File exists

writing to routing socket: File exists

add net 224.0.0.0: gateway 127.0.0.1: File exists
```

One of the common uses of OpenBSD is as a router. In this way, it is possible to take advantage of the Packet Filter firewall. The way to enable the forwarding gateway is to edit the file /etc/sysctl.conf by adding this line:

```
net.inet.ip.forwarding=1
```

If we choose to make a router out of our machine, it is very probable we are also interested to use OpenBSD as a DHCP server. First we have to edit the file /etc/rc.conf.local by adding the line dhcpd_flags="". This way we make sure the daemon starts after every reboot. The next step is to edit the file /etc/dhcpd.conf.

There are several examples of the configuration files under /etc/examples/, among them /etc/dhcpd.conf. These examples are very easy to understand and edit according to our needs.

Other important features to take into consideration at this point are:

- The possibility to create bridges in order to communicate and also filter to interfaces of the network.

- Using OpenBSD as a boot server for machines with PXE-capable NIC. In this way is possible to serve images of different operating systems.
- The Common Address Redundancy Protocol (CARP) to create redundancy and have two or more computers creating a single virtual network interface between them. This could also be a powerful load balancing system to make the traffic safer and smother.

4.4 Management

The first steps to take after a clean installation are tuning the /etc files. It is a good practice to backup the /etc files we want to preserve over different updates. A powerful tool for backup is tar.

```
# tar pcvzf /path/to/the/file.tar.gz /etc/{fstab,httpd.conf,rc.conf.local}
```

In the example:

- p: Preserve user and group ID as well as file mode regardless of the current umask(2). The setuid and setgid bits are only preserved if the user and group ID could be preserved. This is only meaningful in conjunction with the -x flag
- c: Create new archive, or overwrite an existing archive, adding the specified files to it.
- v: Verbose operation mode.
- z: Compress archive using gzip.
- f: Filename where the archive is stored.

The opposite action, to extract files from a tar file would be as follows:

```
# tar Ppxvzf /path/to/the/file.tar.gz
```

Where:

- P: Do not strip leading slashes (‘/’) from pathnames. The default is to strip leading slashes. This is very useful in our case when we want the backup files to replace the original files.
- x: Extract files from archive. If any files are named on the command line, only those files will be extracted from the archive. The file arguments may be specified as glob patterns, in which case tar will extract all archive members that match each pattern. If more than one copy of a file exists in the archive, later copies will overwrite earlier copies during extraction. The file mode and modification time are preserved if possible. The file mode is subject to modification by the umask(2).

For the first installations OpenBSD ships a directory (/etc/examples) with common configuration files to use as a guide when a new configuration is needed.

Another important step to take after a new installation is to add users to the system. The quickest and most suitable way to automatize this step useradd. According to the manual, 'The useradd utility adds a user to the system, creating and populating a home directory if necessary. Any skeleton files will be provided for the new user if they exist in the skel-directory directory (see the -k option). Default values for the base directory, the time of   password expiry, the time of account expiry, primary group, the skeleton directory, the range from which the UID will be allocated, and default login shell can be provided in the /etc/usermgmt.conf file, which, if running as root, is created using the built-in defaults if it does not exist”.

A more friendly way to add a user in OpenBSD is to use the adduser script:

```
# adduser

Use option ``-silent'' if you don't want to see all warnings and questions.

Reading /etc/shells

Check /etc/master.passwd
Check /etc/group
Ok, let's go.
Don't worry about mistakes. There will be a chance later to correct any input.
Enter username []: testuser
Enter full name []: Test FAQ User
Enter shell csh ksh nologin sh [ksh]: ksh
Uid [1002]: Enter
Login group testuser [testuser]: guest
Login group is ``guest''. Invite testuser into other groups: guest no
[no]: no
Login class authpf daemon default staff [default]: Enter
Enter password []: Type password, then Enter
Enter password again []: Type password, then Enter
Name:        testuser
Password:    ****
Fullname:    Test FAQ User
Uid:         1002
Gid:         31 (guest)
Groups:      guest
Login Class: default
HOME:        /home/testuser
Shell:       /bin/ksh
OK? (y/n) [y]: y
Added user ``testuser''
```

```
Copy files from /etc/skel to /home/testuser

Add another user? (y/n) [y]: n

Goodbye!
```

Removing users is a very simple operation with the ”userdel” command:

```
# userdel -r testuser
```

Notice that the -r option will also delete the user filesystem!

Administarting a system where there are many users with different administration tasks is a challenge and for that challenge OpenBSD introduces the tool ”doas”. This tool allows a user to temporarily run commands as the root user. To experienced Unix users this might sounds like ”sudo” that performs a similar task but ”doas” have a simpler and therefore more secure code base. The configuration file (/etc/doas.conf) also looks very simple and easy to understand:

```
$ cat /etc/doas.conf

permit nopass keepenv { PATH PS1 SSH_AUTH_SOCK } :wheel
```

where:

- permit: The action to be taken if this rule matches.
- nopass: The user is not required to enter a password.
- keepenv: The user's environment is maintained. The defaul is to reset the environment, except for the variables DISPLAY, HOME, LOGNAME, MAIL, PATH, TERM, USER and USERNAME.
- wheel: This directive affect all the users on the group wheel.

Once we have the users in the system, we have to deal with the mail. In the file /etc/mail/aliases we can redirect the mail to certain services to certain users in charge of those tasks. We just have to uncomment the lines we want to modify with the name of the user in charge with that service:

```
# Well-known aliases -- these should be filled in!
root:           hugo
manager:        hugo
dumper:         hugo


# RFC 2142: NETWORK OPERATIONS MAILBOX NAMES
abuse:          root
# noc:          root
security:       root


# RFC 2142: SUPPORT MAILBOX NAMES FOR SPECIFIC INTERNET
SERVICES
hostmaster:     hugo
# usenet:       root
# news:         usenet
webmaster:      hugo
ftp:            hugo
```

The next step is to run the command ”newaliases” as a root to make the changes effective.

We also want to automatize daemons to run at the start of after every reboot. The file ”/etc/rc.conf.local” is the place where we should put the series of variable assigments that are used to configure the system daemons:

```
$ cat /etc/rc.conf.local
```

```
ntpd_flags=""

httpd_flags=""

sshd_flags=NO

apmd_flags="-A"

pkg_scripts="php56_fpm"
```

In the example we start the daemons ntpd, HTTPD, apmd (with the option -A, for automatic) and php. We explicitly stop the sshd daemon to avoid external ssh connections.

4.5 LibreSSL

OpenBSD claims to focus on security and encryption. However, nothing is 100% secure and a bad practice can lead to great security holes. Especially when we start adding services and features to our machine.

OpenBSD is shipped with the latest version of libressl, a sub project of OpenBSD and according to their own website (OpenBSD, 2016):

“LibreSSL is a version of the TLS/crypto stack forked from OpenSSL in 2014, with goals of modernizing the codebase, improving security, and applying best practice development processes.

Primary development occurs inside the OpenBSD source tree with the usual care the project is known for.
On a regular basis the code is re-packaged for portable use by other operating systems (Linux, FreeBSD, Windows, etc).

LibreSSL is composed of four parts:

- The openssl(1) utility, which provides tools for managing keys, certificates, etc.
- libcrypto: a library of cryptography fundamentals
- libssl: a TLS library, backwards-compatible with OpenSSL

- libtls: a new TLS library, designed to make it easier to write foolproof applications

LibreSSL is supported financially by the OpenBSD Foundation and the OpenBSD Project."

After the disaster of "heartbleed" in April 2014 in the OpenSSL cryptography, OpenBSD decided to eliminate it from the operating system due to the difficulties on maintenance and keep the code clean. They created the LibreSSL project. For the sake of the usability and coherence with other programs that might want to use LibreSSL, the command keeps the name of OpenSSL.

Creating a key, for example, a HTTPD server, is very easy:

First we need to generate the private root key, for example a 2048 bit key. There is no need to say that this key should be kept in a secure place with restricted access. In case somebody has access to this key they could generate certificates that a browser will accept.

```
# openssl genrsa -out root.key 2048
```

It is also possible to generate a password-protected key by adding, among others, -des3. In this way, it will ask for the password every time we need to use it.

```
#  openssl genrsa -des3 -out rootCA.key 2048
```

Optionally we can also self-sign our certificate. With this we can create more certificates for our different services and provide secure connections via https.

We only need to type the command:

```
#  openssl req -x509 -new -nodes -key root.key -days 1024 -out root.pem
```

After that a script will ask for some information about us:

```
You are about to be asked to enter information that will be incorporated

into your certificate request.
```

```
What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

-----

Country Name (2 letter code) [AU]:FI

State or Province Name (full name) [Some-State]:Varsinais-Suomi

Locality Name (eg, city):Turku

Organization Name (eg, company) [Internet Widgits Pty Ltd]:Hugos

Organizational Unit Name (eg, section):IT

Common Name (eg, YOUR name):IT Hugos

Email Address:hugo@hugos.tk
```

5 REMOTE ACCESS

Once we have the server running we need an address to access remotely. The right and easiest way is to obtain a fixed ip, get a domain and point it to the destination ip. Then, it is possible to use the DNS server provided by the internet service provider or create a name server. Other cheaper option is to deal with a dynamic ip. That requires somebody to change the ip the domain is pointing every time the ip changes. There are some companies that provide this service on the internet.

5.1 Name server and domains

In case we have total control over our address, we can set up a name server. OpenBSD comes with “nsd” and “unbound” to solve this problem.

First, we have to edit the configuration file “/var/nsd/etc/nsd.conf” with our configuration. In the manual there is an example that we can modify to suit our needs:

```
server:

server-count: 1 # use this number of cpu cores

database: ""  # or use "/var/nsd/db/nsd.db"

zonelistfile: "/var/nsd/db/zone.list"

username: _nsd

logfile: "/var/log/nsd.log"

pidfile: "/var/nsd/run/nsd.pid"

xfrdfile: "/var/nsd/run/xfrd.state"


zone:

name: example.com
```

```
zonefile: /var/nsd/etc/example.com.zone


zone:

# this server is master, 192.0.2.1 is the secondary.

name: masterzone.com

zonefile: /var/nsd/etc/masterzone.com.zone

notify: 192.0.2.1 NOKEY

provide-xfr: 192.0.2.1 NOKEY


zone:

# this server is secondary, 192.0.2.2 is master.

name: secondzone.com

zonefile: /var/nsd/etc/secondzone.com.zone

allow-notify: 192.0.2.2 NOKEY

request-xfr: 192.0.2.2 NOKEY
```

The next step is to create the zone files. Here we need to create as many zone files as we defined on the nsd.conf file. In addition, we should place one under master and the rest of the zones under slave directories on “/var/nsd/zones/”. A zone file should look like this:

```
$ORIGIN example.com.

$TTL 86400


@               3600        SOA          a.ns.example.com.
hostmaster.example.com.          (

                    2014110502      ; serial
```

```
                        1800            ; refresh
                        7200            ; retry
                        1209600         ; expire
                        3600 )          ; negative

                NS      a.ns.example.com.
                NS      b.ns.example.com.

                MX      0 mail.example.com.

a.ns            A       108.xx.xxx.xx
b.ns            A       108.xx.xxx.xx
mail            A       108.xx.xxx.xx
```

The next step is to modify the unbound configuration file. OpenBSD provides an example on “/var/unbound/etc/unbound.conf”, which is very easy to implement.

At the end we have to modify the file “/etc/resolv.conf” which should look like this:

```
search example.com
nameserver 127.0.0.1
nameserver (ISP dns)
lookup file bind
```

In order to start these daemons automatically, we need to add the following lines to “/etc/rc.conf.local”

```
nsd_flags=
unbound_flags=
```

5.2 SSHD

According to the OpenSSH manual (OpenBSD, 2016), “ssh is a program for logging into a remote machine and for executing commands on a remote machine. It is intended to provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections, arbitrary TCP ports and UNIX-domain sockets can also be forwarded over the secure channel”.

When running a server, we need to set up and configure the ssh daemon (sshd) in order to control the connections to the server. The configuration file of the sshd is “/etc/ssh/sshd_config”. OpenBSD comes with a default configuration which is very easy to tune if we know what we are doing.

In case we have an OpenBSD box as a desktop, it would be wise to stop the ssh daemon on the boot adding one line to the “/etc/rc.conf.local”:

```
sshd_flags=NO
```

By default OpenBSD disables access by ssh to the root account. It is important to control who has access to the machine via ssh and make good use of doas to control the level of privileges of the ssh users.

To avoid man-in-the-middle attacks it is important to make sure that the daemon is only accepting protocol 2 by adding one line to “/etc/ssh/sshd_config”:

```
Protocol 2
```

It is possible and recommendable to discard the passwords and use keys instead.

Create keys:

```
$ ssh-keygen

Generating public/private rsa key pair.
```

```
Enter file in which to save the key (/home/hugo/.ssh/id_rsa):

Enter passphrase (empty for no passphrase):

Enter same passphrase again:

Your identification has been saved in /home/hugo/.ssh/id_rsa.

Your public key has been saved in /home/hugo/.ssh/id_rsa.pub.

The key fingerprint is:

SHA256:ndwVfndZ2xdGsldumJTkLDv5QSRd2p1lHlK8fzMzySg
hugo@fugu.my.domain

The key's randomart image is:

+---[RSA 2048]----+

|           .=OO*|

|          BB@%|

|            ..X=&|

|         o o *.++|

|        S + = + o|

|          E + Oo|

|              . . *|

|                 |

|                 |

+----[SHA256]-----+
```

Once we have the pair of keys, we can copy the public pair “id_rsa.pub” on the server:

```
$ cat id_rsa.pub >>.ssh/authorized_keys
```

Once we have the keys on the server, we can modify the “/etc/ssh/sshd_config”:

```
PasswordAuthentication no
```

5.3 FTPD

FTP stands for file transfer protocol and an anonymous FTP is the perfect tool for sharing software over the internet since most of the browsers support FTP connections.

Since HTTP protocol can also be used to transfer files, several discussions arose about the subject. Which is faster or more convenient? Or which will be abandoned in the future?

Why do we need to choose one when it is possible to have both? The solution is to put the anonymous FTP server home directory under the HTTP directory, for example “/var/www/ftp/”.

To start an anonymous FTP server requires having a FTP account on the system. Because we do not want to allow FTP users to have a shell we need to add a fake one:

```
#  echo /usr/bin/false >> /etc/shells
```

Then we can create the FTP account as described in the OpenBSD site:

```
# adduser

Use option ``-silent'' if you don't want to see all warnings and questions.

Reading /etc/shells

Check /etc/master.passwd

Check /etc/group

Enter username []: ftp

Enter full name []: anonymous ftp

Enter shell csh false ksh nologin sh [ksh]: false
```

```
Uid [1002]: Enter

Login group ftp [ftp]: Enter

Login group is ``ftp''. Invite ftp into other groups: guest no

[no]: Enter

Login class authpf daemon default staff [default]: Enter

Enter password []: Enter

Disable password logins for the user? (y/n) [n]: y


Name:       ftp

Password:   ****

Fullname:   anonymous ftp

Uid:        1002

Gid:        1002 (ftp)

Groups:     ftp

Login Class: default

HOME:       /home/ftp

Shell:      /usr/bin/false

OK? (y/n) [y]: Enter

Added user ``ftp''

Copy files from /etc/skel to /home/ftp

Add another user? (y/n) [y]: n

Goodbye!
```

Then we create the subdirectories etc and pub. They should be owned by root and the permissions have to be set to 555.

If we want the server to start on every boot we should add the following line to "/etc/rc.conf.local":

```
ftpd_flags="-llUSA"
```

Where:

- **`ll:`** detailed logging to syslog.
- **`U:`** log users so who(1) and similar programs can see them.
- **`S`** - log transfers to `/var/log/ftpd`.
- **`A`** - permit only anonymous FTP transfers.

5.4 HTTPD

The HTTP server stands for hypertext transfer protocol. That is the foundation of the world wide web. OpenBSD has his own server HTTPD, which is very similar to the NGINX server. Of course, the OpenBSD team has focused this server on security and correctness. It is chrooted, meaning it that can only deal with files under its home directory "/var/www/". To start the server on every boot we need to add the following line to "/etc/rc.conf.local":

```
httpd_flags=
```

OpenBSD has an example of the configuration file on "/etc/examples/httpd.conf":

```
# $OpenBSD: httpd.conf,v 1.14 2015/02/04 08:39:35 florian Exp $


#

# Macros

#

ext_addr="*"
```

```
#
# Global Options
#
# prefork 3

#
# Servers
#
# A minimal default server
server "default" {
	listen on $ext_addr port 80
}
# A name-based "virtual" server on the same address
server "www.example.com" {
	listen on $ext_addr port 80
	# Logging is enabled by default, but it can be turned off per server
	#no log
	location "/pub/*" {
		directory auto index
		log style combined
	}
	location "*.php" {
		fastcgi socket "/run/php-fpm.sock"
	}
	location "/cgi-bin/*" {
		fastcgi
		# The /cgi-bin directory is outside of the document root
		root "/"
	}
	root "/htdocs/www.example.com"
}
# An HTTPS server using SSL/TLS
server "secure.example.com" {
	listen on 127.0.0.1 tls port 443
	# Define server-specific log files relative to /logs
	log { access "secure-access.log", error "secure-error.log" }

	# Increase connection limits to extend the lifetime
	connection { max requests 500, timeout 3600 }
	root "/htdocs/secure.example.com"
}
# Another server on a different internal IPv4 address
server "intranet.example.com" {
	listen on 10.0.0.1 port 80
	directory { auto index, index "default.htm" }
	root "/htdocs/intranet.example.com"
}
# An IPv6-based server on a non-standard port
```

```
server "ipv6.example.com" {

        listen on 2001:db8::53f6:3eab port 81

        root "/htdocs/ipv6.example.com"

}

# Include MIME types instead of the built-in ones

types {

        include "/usr/share/misc/mime.types"

}
```

As we can read it is very easy to configure and tune.

6 BACKUPS

OpenBSD comes with two powerful tools for backup: dump and restore. Originally these tools were conceived to be used with a tape but since it is possible to dump to a file, and most importantly, restore it from a file the sky is the limit.

It is possible to make incremental backups in order to save time and avoid redundancy. For example, we can start with a total backup (level 0) and then start increasing the level every time we make a new backup so we have only the modifications since the previous level backup:

```
# dump -0au -f file.dump /dev/rsd0a
```

Where:

- **0:** is the level
- **a:** automatically determines the size (of the tape).
- **u**: makes an update the file /etc/dumpdates to record the backups
- **f:** the name of the file (or the device we use).

To restore our backup:

```
# restore rf file.dump
```

This will restore all our files in the same directory we are.

7 CONCLUSION

The main goal of this thesis was to create a personal server providing services, such as FTP, HTTP and a backup service via SSH and implement it in a secure way with a solid cryptography.

An additional goal of the thesis was to implement the server not only with free software but also open software, proving that it is possible to achieve professional results with this philosophy and tools.

The server was successfully implemented as specified and the learning experience has been very important to understand concepts that could be very useful to apply in other environments.

The set up and the maintenance of the sever requires a knowledge very important in the future of any junior System Administrator.

Definitely the project leaves the door open for future challenges, such a mail server which would only be possible with a static IP and a better control of the internet connection, a common feature in working environments.

REFERENCES

Berg, M. (2015). Mass Surveillance: Part 2 - Technology Foresight [Online]. Available at: http://www.europarl.europa.eu/RegData/etudes/STUD/2015/527410/EPRS_STU(2015)527410_REV1_EN.pdf [accessed 12.11.2015]

Gamino, A. (2015). Mass Surveillance: Part 1-Risks, Opportunities and Mitigation Strategies [Online]. Available at: http://www.europarl.europa.eu/RegData/etudes/STUD/2015/527409/EPRS_STU(2015)527409_REV1_EN.pdf [accessed 12.11.2015]

Lenovo Group Ltd (2016). SMM “Incursion” Attack [Online]. Available at: https://support.lenovo.com/fi/fi/product_security/smm_attack [accessed 20.01.2016]

LibreSSL. (2016). LibreSSL [Online]. Available at: http://www.libressl.org/ [accessed 15.10.2015]

Lucas, M. (2011). SSH Mastery. San Francisco, USA: Tilted Windmill Press.

Lucas, M. (2013). Absolute OpenBSD 2nd Ed. San Francisco, USA: No Starch Press.

Lucas, M. (2013). DNSSEC Mastery. San Francisco, USA: Tilted Windmill Press.

Lucas, M. (2013). An OpenBSD talk [Online]. Available at: https://www.youtube.com/watch?v=daT78L17Di8 [accesed 12.10.2015]

OpenBSD. (2016). OpenBSD [Online]. Available at: http://www.openbsd.org/ [accessed 12.10.2015]

OpenSSH. (2016). OpenSSH [Online]. Available at: http://www.openssh.com/ [accessed 15.11.2015]