Network packet capture with security requirements

8060155 - Jorge Machado

Redes e Serviços de Comunicação

Escola Superior de Tecnologia e Gestão de Felgueiras

A thesis submitted for the degree of

Master

Nov, 2012

1. Reviewer:

2. Reviewer:

Coordinator:

Day of the defense:

Signature from head of PhD committee:

Acknowledgements

Some words about me, I have studied Informatics Engineering until end 2010. I've started my Master degree in 2011, this is my final work that will end my Master degree on Network and Communication Services. I started working as SAP Administrator in Germany on the 1st of November of 2011. I hope you enjoy reading my report.

I will start to acknowledge the help and support from some persons. Firstly, to my parents because without them I would not be who I am today. Secondly, to my supervisor Professor António Pinto, he has helped me on the entire project and on reviewing this report. At the end I want to acknowledge the help of the company where I am currently working, has they provided the hardware to performs the tests. Thank you.

This part is easy, this Thesis is dedicated to my parents and my friends.

Abstract

Organizations tend to make an intensive use of computer networks and Internet connections. Examples of such organizations are the ones with web presence, with it's own fully qualified domain name, with it's own email service and more recently, organizations that use cloud-based services. The common technology being the IP.

The Internet has been growing since it's creation, new hosts being added frequently. Internet access technologies also have evolved in terms of quality and available bandwidth. As a result, millions of packets travel through the Internet every day. The probability of an host connected to the Internet being targeted for an computer attack attempt is very high. The most common network security technique is based on network traffic filtering, i.e. firewalls, with the purpose of blocking intruders out of the corporate networks.

The traditional small and medium-sized organization tend not to have a dedicated team, dedicated hardware or dedicated software to deal with attacks. The common attack will not be a attack that targets the organization, but an attack to a generic set of IP addresses. Here, the common reaction when in presence of an attack is no reaction, with most of the attacks going unnoticed for large periods of time.

For situations where attacks where noticed it would be useful to have a tool that allows the organization to prove that the attack had occurred. The idea is to collect network data, securing it in such a way that could be used, for example, in a court of law as proof of the criminal activity.

Keywords: Network packet capture, Security requirements

Resumo

Organizações tendem a fazer uso intensivo de redes informáticas e ligações á Internet. Exemplos de organizações deste tipo são por exemplo aquelas com presença na web, com o seu domínio, serviço de email e mais recentemente a integração de *cloud-based services*. A tecnologia comum nestes serviços normalmente é o protocolo IP.

A Internet tem vindo a crescer desde a sua criação, com novos dispositivos a serem adicionados frequentemente. A Internet também tem vindo a crescer ao nível de qualidade e de largura de banda. Como resultado milhões de pacotes atravessam a Internet todos os dias. A probabilidade de um dispositivo ligado á Internet ser alvo de um ataque é alta. A técnica mais usada para combater ataques na web é baseado em filtros, como por exemplo *firewalls*. Estas tem o objectivo de bloquear intrusos das redes corporativas.

Pequenas e médias organizações normalmente não possuem a capacidade de ter equipas, hardware e software dedicado para monitorizar e reagir a ataques informáticos. O ataque mais comum normalmente visa uma gama de IP's e não uma organização em específico. Portanto quando ataques deste género recaem sobre uma empresa a reacção é "não reagir" isto porque muitos dos ataques passam despercebidos por um longe período de tempo.

Para este tipo de situações seria útil ter uma ferramenta que ajudasse a organização a provar este tipo de ataques. A ideia é criar uma software que recolha a informação num dado ponto de uma rede, assegurando que a informação recolhida não pode ser alterada. Posteriormente deve ser possível usar este tipo de provas em tribunal para obter mais informações sobre o ataque.

Contents

List of Figures

IP Internet Protocol

PCap Packet Capture

FPCap Full Packet Capture

RAM Random Access Memory

NAPI New API

LVM Logical Volume Manager

NIDS Network Intrusion Detection Systems

RAID Redundant Array of Independent Disks

PKTGEN Linux Packet Generator

CPU Central Processing Unit

IDS Intrusion Detection System

ISP Internet Service Provider

RPM RPM Package Manager

SSD Solid State Drive

Chapter 1

Introduction

Organizations tend to make an intensive use of computer networks and Internet connections. Examples of such organizations are the ones with web presence, with it's own fully qualified domain name, with it's own email service and more recently, organizations that use cloud-based services. The common technology being used is the Internet Protocol (IP).

The Internet has been growing since it's creation, new hosts being added frequently. Internet access technologies also have evolved in terms of quality and available bandwidth. As a result, millions of packets travel through the Internet every day (1). The probability of an host connected to the Internet being targeted for an computer attack attempt is very high (2). The most common network security technique is based on network traffic filtering, i.e. firewalls, with the purpose of blocking intruders out of the corporate networks. On one side, additional tools exist, namely anti-virus software, Intrusion Detection System (IDS), and honey pots, that may help protect organizations against computer attacks. On the other side, such additional tools require more personnel, with specific technical training and more hardware resources, that limit its adoption in small and medium-sized organizations.

The traditional small and medium-sized organization may not be able to cope with the costs of maintaining a dedicated team, dedicated hardware and dedicated software to deal with such attacks. Moreover, the majority of the computer attacks will not be a direct attack to the organization, but an attack to a generic set of IP addresses. The point being, that a generic attack is not as harmful as a specific attack. Such scenario lead the authors to assume that, in the context of a small and medium-sized

organization, the common reaction when in presence of an attack is no reaction, with most of the attacks going unnoticed for large periods of time.

For situations where attacks where noticed it would be useful to have a tool that allows the organization to prove that the attack had indeed occurred. The idea is to have some kind of architecture (software / hardware) that would collect all or part of the network data. The next step is to secure this data in such a way that could be used, for example, in a court of law as proof of the criminal activity. To achieve this, the data record must be first encrypted to guarantee confidentiality and signed to guarantee Integrity. With this two steps performed we can archive the encrypted files and retrieved when needed in a safe way.

1.1 Reference Scenario

The adopted reference scenario has focus on small or middle size organization, that cannot afford to have a dedicated team to do network forensics analysis. They usually have one or two persons responsible for all technology that runs the organization. Small and medium organizations usually have one or two internet connections, and here is where the software can be implemented with relative small effort. This will allow organizations to register all connections from the outsize to the corporate network.

1.2 Objectives

The main objective of this work is to develop a solution that performs network packet capture that also considers security requirements. A list of the identified requirements is presented next.

1. Full packet capture
2. Confidentiality
3. Integrity
4. Continuous operation

The first requirement states that all network traffic must be captured and stored. The second addresses the issue of how to deal with the captured data in a safe way and

that confidentiality is not compromised. The third requirement is the integrity of the captured data, the meaning that the software must assure that the data is not change in anyway. The fourth requirement is the continuous operation, meaning that the software should perform space management automatically. There are several problems relative to Packet Capture (PCap) that we describe in Section 2.7.

The figure 1.1 is a possible scenario of the implementation of this software. The Internet Service Provider (ISP) is normally the entry point of most connections on a Organization, this connections them usually go through one or more firewall and them to the corporate network. The Network TAP is exactly between ISP and the first firewall, "cloning" every single packet that comes in that delivering it to the Network Traffic Register and Firewall. With this environment we can guarantee that that the packets entering the firewall are the same entering the Network Traffic Register

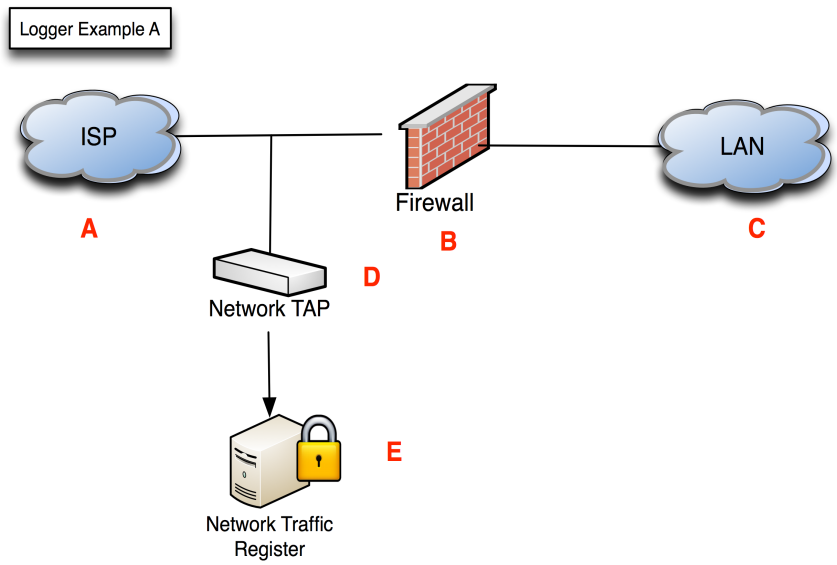

Figure 1.1: Reference scenario

1.3 Results

The expected outcome of this work is a functional prototype that performs network packet capture with security requirements to be adapted by Small and Mid-size Companies. The final solution is expected to operate on commodity hardware, cutting down the costs of its implementation.

1.4 Organization

The remaining of this report is structured has follows. On Chapter 2 you can find the State of the Art, objectives and related work. Still on Chapter 2 you can find some theoretical concepts that will be used later on in work for example what types of encryptions to use with big files. Chapter 4 is Validation and is located on page 25. Here you can find what Hardware, Operating System, Software and tests environments were used during this work. Still on Chapter 4 we perform some tests to analyze how the operating system our other tools react when the system is on full load.

Furthermore in Chapter 3 on page 12 is Proposed Solution - Network Packet CaptureSec. Here is proposed a solution to address the objectives / problems referred on Chapter 1, this is the main chapter of the report. At the end, you will find Chapter 5 page 33 with Conclusion, this Chapter is self explanatory and the reader will encounter a typical Conclusion for this type of work.

Chapter 2

State of the Art

Several applications provide packet capturing and analyses, examples from such applications are Snort (3), The Brof (4) and nIPS IntruPro Series (hardware appliance). They all make part of Network Intrusion Detection Systems (NIDS) (5). This applications are very Central Processing Unit (CPU) intensive as they need to read and analyze thousands of patterns over real time network traffic, such load can lead to packet loss, hampering traffic analysis. As this applications run away from the objective of this work some research was done, the authors find some other tools that deal with packet capture such: openFPC (6) and Daemonlogger (7) and off course the most basic tool tcpdump (8). This work will focus on the later ones.

2.1 Packet Capture

Full packet capturing, packet capturing by sampling and partial packet capturing are the three main techniques to perform Packet Capture.

2.1.1 Full Packet Capturing

In this technique the objective is to capture all data in the packet, including headers and payloads. This is the best way to record all network activity, but it comes with two main down sides. One is how to store the data to storage (disk bottleneck can occur) the other one is space in the storage to store all data. A medium sized network per day can generate from 10GB traffic up to one or two TB (9) per day. Lets say that you have a network with a 2TB traffic per day, your monitoring device most be able

to record constantly at 24,2 MB/s without counting with the high network usage time. The monitoring device need to have adequate storage if the objective is to perform full packet capture at wire speed's.

2.1.2 Packet Capturing by Sampling

With this approach what is done is perform PCap by samples, this means for example every 2 seconds record 30 seconds of activity and then sleep the process that performs the packet capturing. This clearly addresses the issue from space requirements reference previous. One of the problems with this approach is that applications like snort (3) or The Bro (4) will return inaccurate results, this happens because of the missing packets. It is worth to use sampling if it for simple network statistics.

2.1.3 Partial Packet Capturing

On (9) authors have opted to perform Partial packet capture to reduce several terabytes of information to just a few hundred gigabytes. This was accomplished by recording only the first 20 Kbytes of information for each connection. With this technique authors could actually get the most useful information from each connections (starting connections). Plus if an application like snort run analysis on the data collected, it would probably identify an attacker correctly. This happens because we are constantly capturing packets from the network and not with intervals. For example with 20 Kbytes of information for each connection is possible to determine: type of connection, source ip address, destination ip address, ports and better if we are in the presence of one network attack (for example brute force attacks will se here a lot of entries).

If full packet capture is not possible this technique should be user instead of sampling technique, as it more efficient.

2.2 Packet Capturing Tools

2.2.1 Tcpdump

Tcpdump(8) is one of the most basic tools for Packet capturing. It comes in almost all Linux/Unix distributions. It is very useful when administrators need a quick and dirty

solution to perform packet capture on a machine to posterior analysis. For example to check if the server is communication correctly with other systems. This tool is not very good in long term or continuos packet capture and with high traffic some packets are discarded.

2.2.2 Open FPC

Open FPC (6) is a very good set of tools that provide a lightweight full packet capture over the network. It has some requirements for install like apache2, php5. The main objective behind this application is to allow non-expert users to performs traffic analysis on network traffic. This set of tools are written in C and Perl and is based on tree components. Client device, ofpc-slave and ofpc-master. The Client device is used by the operator and it will run the client program (ofpc-client.pl). The ofpc-slave is the application that performs the packet capture. At last the ofpc-master, this is like a "proxy" that simple route or forward requests to the ofpc-slave and delivers the files to the client.

2.2.3 Daemonlogger

Daemonlogger (7) is an application developed by Martin Roesch, it performs Full Packet Capture (FPCap) and it his written in C. The author of this application is also developer for Snort, a well know open-source NIDS. Daemonlogger is based on the library libpcap (10) and can be run in two modes. One is the tap mode and the other is dump mode, in tap mode the application rewrites the packets to another interface acting like and proxy or an soft tap. For our work we used the dump mode, in this mode the application reads the packets and writes them to a PCAP file.

Daemonlogger is a open-source software that can be used and modify by everyone.

2.2.4 Comparassion of Packet Capure Tools

As Tcpdump does not fall in the minimum requirements of this project (continuous PCap) it will not be here referenced. As for Open FPC is a very complex set of tools that perform FPCap but written in perl. For our project we need a simpler tools that we could simply change without compromising oder parts of the software. As

Daemonlogger answer all this requirements it was chosen to our third party software to perform FPCap.

We opted to use third party software mainly for two reasons the first one is performance. Applications written in C are extremely fast. Tests performed during this work showed that Daemonlogger can capture and write files to disk with speeds up to 970 Mega bits per second. The second main reason is that we haven't have the time to write an python application that performs so good as Daemonlogger. Why re-invent the wheel if she already exists an can be used ? Thats we chosen Daemonlogger as our software for FPCap .

2.3 Cryptography

Cryptography is a very complex subject, it evolves random and mathematical expressions with several prerequisites. If you want extended details about Cryptography you should consult some book like this one Applied Cryptography (11). If you are a programmer and you are thinking "oh well i only need to generate a small random number and it will be fine then". No you won't, this is so a sensible subject that simple things like generating a random string can compromise all your Cipher work. The best advice is, if you are using some kind of Cryptography on your coding you should look for some framework that generates keys and randoms items for you like in (12). One simple example is the class RandomPool from Python it is not secure (12) and should not be used ! Moving on, other word that comes always along with Cryptography subjects is non-repudiation. This means if the person A sends a message to person B and this message uses some type of decent non-repudiation method the person A cannot say that he has not send that message.

2.4 Hash Functions

When implementing Hash functions the main objective is, given some arbitrary-sized data outputs a fixed-length hash value. Usually this type of objects (as MessageDigest from Java[1]) have an method that allows update the data already passed to the object upon creation. This Update method allows the creation of hash results of considerable

[1]For more information please consult: `http://docs.oracle.com/javase/1.4.2/docs/api/java/security/MessageDigest.html#MessageDigest(java.lang.String)`

sized files without the need to read the whole file. Hash functions are very especial because they guarantee that given a string A, it will always output the same result and if you have a string B it will never output the result from A. This is in theory, in real world there exist collisions and they must be corrected, that's why there are several hash functions algorithms like MD5, SHA, SHA256, SHA512 or HMAC.

2.5 Asymmetric Cryptography

Asymmetric keys are the more popular type of key(13). These keys come in pairs and are known as Public / Private keys. Nowadays there are used almost every where. They are relative safe (I'm saying relative safe because there are some people that already found a way to break this type of Ciphers if servers are using TLS 1.0, for more information you should search for *chaosradio* podcasts they are in German) and can be found in websites like Facebook, Gmail, Hotmail, your bank account and a lot more. This type of encryption allows for example two identities or hosts (like your computer and Facebook server) to exchange messages in a secure way without letting anyone see it. This concept is used to guarantee that no one can read one specific key unless you are in the possession of your private key. Private and Public keys are always generated in pairs, so public key dependes from the private key. Like the name says public keys can be shared and public know (think it like your home address) but in another hand private keys should be very well stored (think it like your home key or your credit card numbers), this pair keys have some especial characteristics. If you use your Public key to cipher a string it be only possibly to decipher it with the corresponding private key. In contrast if some string is ciphered with a private key all identities in a possession of the corresponding Public key can decipher and read it. The main problem with this method is that is high CPU intensive because it need to compute several mathematical operations and it only should be used for small chunks of data.

2.6 Symmetric Cryptography

The name of this concept is called Symmetric Ciphers and how the name say it is used to cipher text given a primary key and some random data. There are several Symmetric Ciphers algorithms available. This algorithms uses a smaller and random generated keys, it have the downside that it does not offer such a good security as Asymmetric Ciphers but the positive aspect is that they are way faster em comparison to Asymmetric Ciphers. The most known are AES, DES and DES3 all those algorithms perform symmetric ciphering in contrast with Public / Private keys as they perform asymmetric ciphering. In Our work we perform symmetric ciphering with AES encryption for better performance. The Cryptography kit that we use has a very nice Object that allows us to get an symmetric cipher with the input from a public certificate. Finally this types of ciphers are much faster them asymmetric ciphers but have the down side that can be quicker break with brute force attacks.

2.7 Related work

From all the literature investigated such as (14) (15) (16) (17) or (18). The main bottleneck identified by the authors was CPU zone. Meaning that there was not enough free CPU cycles to process all packets at wire-speeds. This issue was resolved by the implementation of device polling (15) and NAPI (19). Some groups have implemented device polling and NAPI (e.g. (15)) but there was still problems with PCap, this was due to the increasing speeds of the networks. To deal with the increasing bandwidth problems there are several attempts to completely bypass the kernel, in other words the packets travel almost directly from the network card to the userspace (20), with this solution is possible to capture packets at high speeds.

To address the problem for PCap in environments with 10Gbits connections, usually the main approach is to distribute the load to several links (e.g. 10 x 1Gbit links), and at the end of those links are devices that perform PCap (14) or (21). In last issue in the CPU scope is, the main techniques that we discussed are not taking benefit of dual core or quad core CPU's.

On (22) Mahdi et al., have proposed a novel solution that address this issue. The main idea behind is compose of to main modules, on DMA_MAP and another libDashCap. DMA_MAP is located in the kernel and libDashCap is located in the user space.

According to authors with the new kernels this is the optimal approach as the applications with interact with libDashCap and this library allows "queries for packets" from multi-processors. Tests have showed that this solution outperforms all others, they are able to perform PCap without packet loss in a Gbit environment and using multi-core processors with a CPU usage above 10 %.

Chapter 3

Proposed Solution - Network Packet CaptureSec

After defining all requirements for this software (Full packet capture, Confidentiality, Integrity and Continuous operation) we have verified some knowing issues for the requirements that we want to implement in Chapter 2. On this chapter we explain what was developed, how it was developed and why it was implemented in that way. In the section Concept (3.1) it is pointed out some considerations that we had taken in account to avoid typical problems like overloaded servers and uncontrolled number of encryptions. On the next section Implementation (3.2), the reader will find some code highlights that we think they are important to mentioning, still on the same chapter there are tree subsections. One for the changes that we need to perform on the software Daemonlogger (7), them a TCP/IP server that we call Message server and finally our Encrypt server. Furthermore we describe what options can be configured with this software and what are the results of each option.

3.1 Concept

As the reader has notice the final solution that we came up its based on tree components, and this works very well on our tests. Our initial approach was not so complex as these final version that we present in the next sections. On the initial approach we alter the source code from Daemonlogger and added a new thread every time that the maximal file size was reached, on this new thread we call an Python program that encrypts the

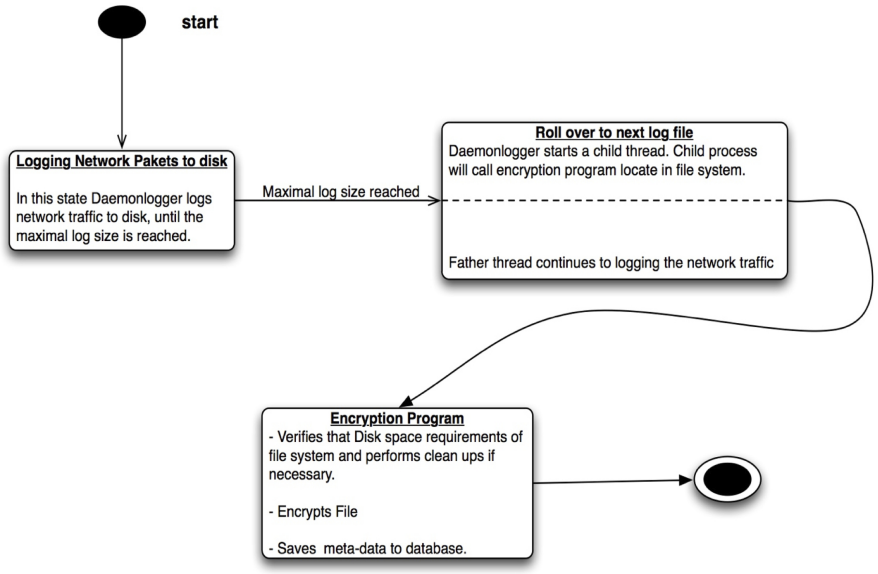

Figure 3.1: UML Diagram States of initial aproach

file, encrypts the "small" key with the public key from the loaded certificate and deletes the file that was not encrypted. On Figure 3.1 you can se an State Diagram of what i have described. The biggest problem that we had with this approach is on controlling the number of process that are perform parallel encryption. On tests that we performed we have moments that we see four encryption process plus the Daemonlogger process. This makes the server completely unusable and unreliable to perform continuos and seamless FPCap, further more we notice that the threads that were generate doesn't ended properly leaving zombies processes on the server. All this symptoms only occur when encryption threads are generate quickly enough leaving no time to older encryption threads to terminate, the threads are generated more quickly as more traffic comes to the Daemonlogger or the use of small files to store the network traffic.

Its was clear for us that this issue must be resolved as when traffic arrises the software become unstable. For that we need to aboard the problem with a completely other technology and we decided to use TCP sockets, creating one Message server that deals with all communication between Daemonlogger and the encrypt server. This new approach allow us to have several Daemonloggers running and several encryption server running.

3.2 Implementation

As we referred above the final prototype is based on TCP servers, this means that messages are exchanged between the "new Daemonlogger", the Message server and the Encrypt server. Now the process works as follows: First the message server need to be started, them when you start the Daemonlogger server it authenticates it self on the message server. When the encrypt server is started it processes in the same way as Daemonlogger and authenticates on the message server. During the runtime of Daemonlogger, when it reaches the maximal defined log size it spawns a new thread. This connects to the Message server and request an encryption. The message server adds this request and terminates the connection to Daemonlogger thread, the message server, now connects to encrypt server and gives the command to start encryption. When the encryption is done the un-encrypted file is delete. More detailed information will be provided on the next subsections 3.2.1, 3.2.2 and 3.2.3. In the Figure 3.2 you can find most of the messages exchanged between the tree components.

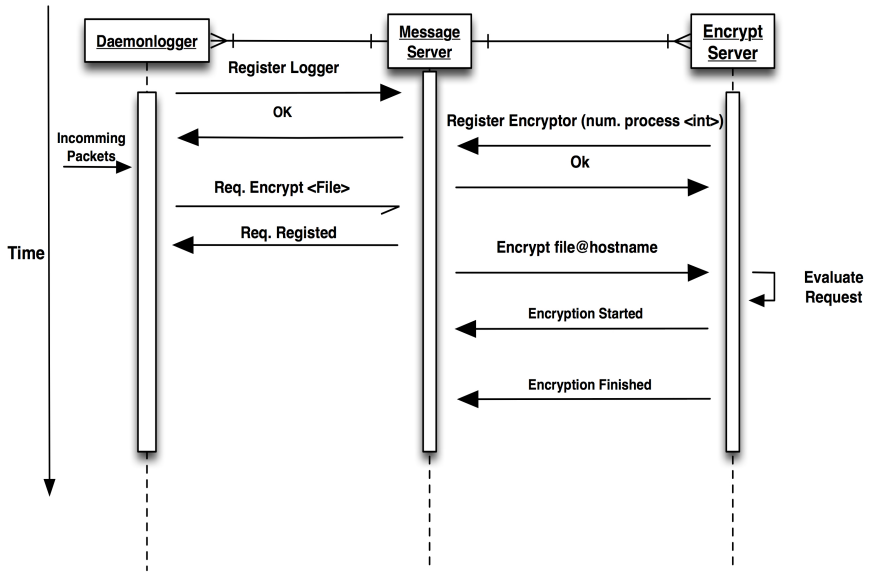

Figure 3.2: UML Sequence Diagram of messages exchanged

3.2.1 Changes to Daemonlogger

Software developed by Martin Roesch has no capabilities to perform connections to the Message server, so we need to perform two modifications on this program. The first one is located under the function main, here we added the part that this instance of the program registers it self on the message server. If the answer from the message server is positive the Daemonlogger becomes as answer on the connection an authentication code. If registry process does not return a sucessfull status form the message server, the Daemonlogger will not start. This authentication code is used to authenticate the requests when requesting an encryption. When the maximal log size is reached, the new created thread sends a request to the message server. This request contains the file that the encryptor should encrypt and the authentication code used on the registration from the Daemonlogger. Another point to notice is that is possible to have multiple instances running on the same host.

The code used to perform the request can be viewed in the following source code:

Listing 3.1: Connection code for Daemonlogger

```
struct sockaddr_in target;
int sock=socket(AF_INET,SOCK_STREAM,0);
int adl=sizeof(target);
char linha[81];
bzero((char *)&target,adl);
target.sin_family=AF_INET;
target.sin_addr.s_addr=inet_addr("10.0.0.2");
target.sin_port=htons(9250);
char hello[] = "HELLO\n";

if(-1==connect(sock,(struct sockaddr *)&target,adl)) {
        printf("Connect to msg_server failed\n");
        exit(-1);
}
else{
        write(sock,hello,strlen(hello));
        read(sock,linha, 300);
        strcpy(hello,"REQ_ENCRYPT:");
        strcat(hello,my_key);
        strcat(hello,":");
        strcat(hello,my_hostname);
        strcat(hello,":");
        strcat(hello,current_file);
```

```
        strcat(hello,":");
        strcat(hello,file_size);
        strcat(hello,"\n");
        write(sock,hello,strlen(hello));
        read(sock,linha, 300);
        close(sock);
}
```

3.2.2 Message server

This component is the where all communication goes trough. It's one of the most complex components on all the project, with this type of architecture it allows high scalability in a way that it can be easy added a new encrypt server or an Daemonlogger server, it will register on start up with the running Message server. Now when the Daemonlogger requests one encryption to the Message server it will add that request on one queue. This queue will be automatic distribute trough all active Encrypt servers. This distributions will automatic occur when a request to encrypt is requested to the Message server. Another feature is that distribution can be triggered every x (where x is read from a configuration file) seconds. On Figure 3.2 you can see how important this component is.

This component was completely developed in python. On our coding we use some standard classes like *SocketServer.StreamRequestHandler* allowing us to code much quicker and efficient. Another advantage of using python is that it reduces the number of code lines developed, we have circa 500 lines of code for the complete Message server. Every time that the server is stopped the queue will be write to an local file so when the message server starts again it will already know what to distribute. As we are using threads we decided to use the class Lock, it allow performing dictionaries locks in an easy way and this is thread safe. On Appendix A the interested reader can find the code used to create the message server (configuration files not included).

When you start the message server simply execute *python msg_server.py* the program will start and perform the following tasks by the following order :

1. Initialize global queues and global variables;

2. Read queue that was saved to disk before shutdown of message server;

3. Read the configuration file *logger.cfg*, this file has configurations like IP address and port that will be use to perform the binds to the TCP/IP ports.

After all this small steps are performed, it will perform the bind[1] to the ports and the ip address and listen to connections from the network.

3.2.3 Encrypt Server

The Encrypt Server is the third component of this software. This program is again completely written in Python and here is where the heavy encryption occurs to encrypt the network dumps files recorded from the Daemonlooger.

To start the Encrypt Server simply execute *python encr_ server.py*, the program will start and perform the following tasks by the following order :

1. Read the configuration file *encrypt_ srv.cfg*, this file has path's to other critical files like public-key certificate and maximal encrypt processes;

2. Initialize the Pool of processes;

3. Register it self on the Message server and perform the network bind, becoming avalable to respond to encryption requests.

The reader should be now asking what does is mean "Initialize the Pool of processes", As Python is a relative young programing language it implements very useful classes that can be used to performs thats in Multitasking. The Pool class comes with the Module multiprocessing. For the initialization of this object it needs a parameter that is the number of Processes in the Pool, after that you can submit tasks to the Pool. Some advantages of using this type of Objects are :

- Automatically distribution of tasks between Processes;
- Controling all Processes in the pool centraly;
- Communications objects are avaible between Processes, like Pipes;
- Asynchronous commands to the pool, increases performs and reduces bottlenecks;

[1]Bind is a request that the program does to the operating system, and request a range of ports or ip address. After they are "binded" to a program it is not possible to assing the same port again to another program. This is common use in programs that must be listing on a network port.

- Call Back functions for each command submited to the pool, this are used for example when a encryption is done on the Process.

Notice that we are actually using new Processes and not threads[1], this allows better performance and is the recommend way to perform Multitasking with Python.

Now when the message server has a request on queue, it searches for an available encryptor and sends the encryption task via TCP/IP connection. In The Encrypt Server side when receiving a task it evaluates if the task is already registered on the Encrypt server queue and if the request is authenticated. If everything is ok, the program acquires an lock (Process and thread safe), adds the request to the local queue, increments the actual work in progress, calls the function from the pool Object *apply_async* and releases the lock. At this point an encryption is started and simultaneous the TCP/IP connection to the Message Server is ended.

After the fucntion *apply_async* the process of the Pool start execution the target function, in our case is the function *do_the_work*. This function check if we are on a distributed environment or not, this will be explained on Section 3.4. If we are in a local installation we move the file to a temporally location and perform a space validation. If the required space to encrypt the file is not available, them the oldest file is delete by querying the meta information on the database.

Finally we use the class *RSA* from the module *Crypto.PublicKey* that allows to read an public certificate from file to memory, after that the function *encrypt_file* is called. These function accepts the following parameters: key (random key), in_filename (file to encrypt), out_file (name of encrypted file) and chunk-size. The chunk size is customizable, and it controls how much of the file will be read. For example if the file to encrypt has 1GB and Chunk file is 512MB the function will in practice read the first 512MB of the file perform the encryption of those 512MB, them release the memory and them reads the rest of 512MB of file. This function was not developed in this work. It was developed by Eli Bendersky, and it can be found at http://eli.thegreenplace.net (23) and is free to use. After the file is encrypted we perform an encryption of the random key used to encrypted the file. At the end the temporarily file is delete at this point the only way to get access to the network packets is to have the private key from the certificate. And it should be store on an external device not in the server.

[1]If you need more information why we used Processes and not Threads the reader you consult Python documentation and search for Multitasking

When the function ends the same Process still performs the call back functions that notices the Message server that encryption is done and updates status variables.

3.3 File Encryption from Files with Hundred of Mega Bytes

So, how should be approach encryption from files with several hundred Mega Bytes of data? We start with generating a Symmetric key (with our especial Object PKCS from the toolkit), the next step we perform encryption from the file using that key, at the end you encrypt your symmetric key with your public certificate (That's a Asymmetric Key). With this method only the identity in a possession of the private pair certificate will have access to decrypt the file.

To end this section some historical information could be useful, when Cryptography started to have leverage on informatics environments the rule that most identities adopted when they developed a new Cipher algorithm was to keep the code so secret as possible denning freelancers experts to analyze the actually source code.
Now a days the process is complete in the oder way, before one Cipher algorithm become a standard to the industry in most cases it will be release as open source so that freelancers experts can perform source code analyzes. In this way you don't have a "black box" algorithm but something that was tested from the community. This was only a brief introduction to Cryptography, as I reference before if you need more details on this matter please take a look in the specialized literature.

3.4 Deploy methods and restrictions

This section the is intended to the reader that wants to implement this software with multiples Encryptors and multiples Daemonloggers. This section is not a guide but a roadmap with good practices that should be taken account when deploying such software.

For starting as you notice all tree components perform communication between with TCP/IP protocol, so if you want to use this software in a distributed environment you explicated need dedicate network. The next step is to decide where you will install the Message server. There is no need to have a dedicate server for this, you can installed

it on a Daemonlogger server or in a Encrypt server. Again please make sure that all messages trade between the tree components goes trough the dedicated network.

There are two folders with configurations files. One folder is the *msgsrv_setting* and it should be place on the same folder as the *encr_server.py*. Inside this folder you can find the configuration file for the Message server. The other folder is named *encrsrv_settings* and it should be place on the same folder as the *encr_server.py*. Inside this folder you have the configuration file for the Encrypt Server (The configuration options will be explained on Section 3.5). The only relevant configuration that you should pay attention is on the Encrypt Server settings. Here you will find a parameter named *all_in_one*, if this file is set to True the logic from the Encrypt server will think that there is in a presence from an simply or non-distributed installation. If the installation is a simply install them, it will simply move the file to *tmp_encry_location* and them start the encryption. If there is a distribute installation it will first perform a secure copy (scp) from the Daemonlogger to the *tmp_encry_location* and them start the encryption.

At last but not least, you must configure your servers so that is possible to login from the Encrypt Server to the Daemonlogger servers **without any** password. This step is extreme important or it will not perform any encryption on a distribute installation.

3.5 Configurations options

This chapter is relative easy to understand and quick to read. This chapter is relevant to who need to know how does each parameter work. To perform all configurations there is a small script that builds the configuration file by asking some questions that require user input. We will split this chapter in two parts, the first is dedicate to configuration options about the Message server and the other part is dedicated to the Encrypt Server. The reader should be now asking: "what about a configuration file for the Daemonlogger?" Until now there is no configuration file for Daemonlogger, every configuration is passed as argument when the Daemonlogger is started.

3.5.1 Message Server Settings

These configuration file is intended to be simple and so there are only five configurations possibly to the Message server. They are :

- msg_server_ip - Message server ip address where it will responde to request (Remember dedicated network);
- msg_server_port - Port of the Message server where the connections will occur;
- max_encryt_processes - Maximal number of processes that the Message server accepts when a Encrypt server registers;
- automatic_distri_queue - Automatic distribute the queue by time (Default behavior is to distribute the queue when a new encrypt request is registered);
- sleep_time_for_automatic_distri - Time between automatics imports (Only used if automatic_distri_queue is set to True).

On the next extract you can see an example of the configuration file for the Message server.

Listing 3.2: Message Server configuration file example

```
[General]
msg_server_ip = 10.0.0.2
msg_server_port = 9250
max_encryt_processes = 1000
automatic_distri_queue = True
sleep_time_for_automatic_distri = 10
```

3.5.2 Encrypt Server Settings

This configuration file is a little more complicate because as normal it need some information from the Message server. To better understand it, the file is spitted in two sections, one named General and other Encrypt_server. The General section is intended to General configuration like IP address and port to the message server. Let's dive in to the configuration.

- msg_server_ip - IP adrress from the Message server;
- msg_server_port - Port from the Message server;
- db_host - Database host for storing meta-data information;
- db_user - Database user;

- db_passwd - Database password;
- db_name - Database name;
- chunksize - Chunk size to use in the encryptions (Explained on Section 3.2.3);
- pubkey_location - Full path to public certificate;
- decrypt_files_location - Location where to put decrypted files (Not in use);
- encrypted_files_location - Location where the encrypted files are stored;
- tmp_encry_location - Temporally file location to perform encryption
- all_in_one - Type of installation (Explained on Section 3.4);
- files_before_encrypt - Files location before encryption (Only for single install all_in_one is set to True);
- encryp_server_port - Port where the message server will listen;
- encryp_server_ip - Encrypt server IP adrress;
- max_processes - Maximal number of parralel processes that the server should have (We recomend for normal hardware 2 processes);
- file_for_work_in_progress - Location where the to save the file from the work in progress.

Again for the better understand how this file work on the next extract the reader can see a configuration file for Encrypt server.

Listing 3.3: Encrypt Server configuration file example

```
[General]
msg_server_ip = 10.0.0.2
msg_server_port = 9250
db_host = 10.0.0.2
db_user = root
db_passwd = toor
db_name = http_logger
[Encrypt_server]
chunksize = 134217728
```

```
pubkey_location =/install/certificate/pubkey.key
decrypt_files_location = /install/decrypt/
encrypted_files_location = /install/encrypted/
tmp_encry_location = /tmp_encrypt/
all_in_one = True
files_before_encrypt = /log_ramdisk/
encryp_server_port = 9260
encryp_server_ip = 10.0.0.2
max_processes = 2
file_for_work_in_progress = /install/encry_work_in_progress
```

The encrypt server need a database connection. In this work we decided to use the Web Framework Django, (24) on this database we save some information about the encryption that was performed. And can be later used to retrieve metadata, and if more developed to decrypted files. The database has now twelve tables, most of the tables comes with the standard installation of Django. On figure 3.3 the reader can view a database schema for this application (without the standard tables from Django).

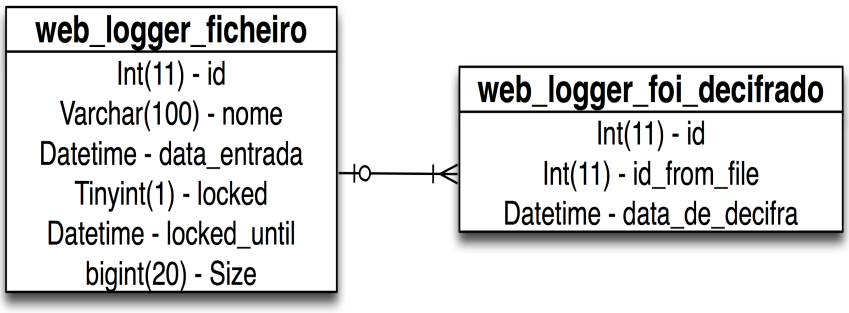

Figure 3.3: Database Scheme

3.6 Chapter Conclusion

On this chapter we present to the reader the main work developed. After some initial tests we came up with an scalable architecture that allows multiple Encryptors and Daemonloggers. The key component to the success of this prototype software is the developed Message Server, he has the responsibility of exchanging messages between Daemonloggers and Encryptors. The second main functionality is to distribute and

control the number of parallel encryptions on each Encryptor. Further more we had the need to change the Daemonlogger source code (3.2.1) in a way that it register to the developed Message Server.

Another component in this prototype software is the Encrypt Server (3.2.3), this python program uses the most recent methods to create, control and use processes in python. We use a pool of processes to start asynchronous encryptions without having performance impacts on the main process that still need to receive connections from the Message Server. The initial encryption of one network capture is optimized to maximize the performance, as the second performance is optimized on security issues leaved by the initial encryption. The Encrypt Server still connects to one mysql database and saves some meta-data that can be used later. There are more functionality's that this Encrypt Server performs and they are explained on 3.2.3. In the end of the Chapter we explain what are the best ways to deploy this work, which configurations is possible and what are the consequences if changed. For a better understanding of the configuration options available to use we still provide and configuration file example for each component. This work is not finished yet, it will need some work on some areas and some more tests. We will enumerate and explain this on **??**.

Chapter Validation, it will be presented a validation of this work. Here it will he showed the results of the tests with the final software version. On Chapter Conclusion is presented the final conclusion of this work. We opted to include a section work in progress, in this section it will be pointed the work that needs to be done to get a final release of this software.

Chapter 4

Validation

As a small reminder to the reader, the objective of the Thesis is to develop an scalable and secure Network Traffic Register. This means that the software should be used to capture all traffic circulating on an network. As reference to the reader two examples are provided on where to install Network Traffic Register on the network.

Example A on figure 4.1 shows the usage of the software at the entrance of connection from an private corporate network to the internet, as here can be useful to proof external attacks. In Example B on figure 4.2 the software is applied on Backbone connection from an corporate network. This are just examples, the reader can deploy the software anywhere in the network for that you only need some type of passive or active network tap.

When developing an software that need to be stable the reader should ask his self at least one main question. How much budget do I have and second how stable or available should it be. For the case of this work the Budget was really limited as i have no availability to buy hardware, but from other point of view it was a good change because the developed software would have to run in relative old or commodity hardware. In this Chapter the reader can expect a presentation of several components used for our work, components like Hardware, Network and Operating System. In the end of the Chapter we explain and analysis tests performed to evaluate the performance of Hardware and Operating System. This chapter is useful to fully understand the next Chapter Work Evaluation 3.

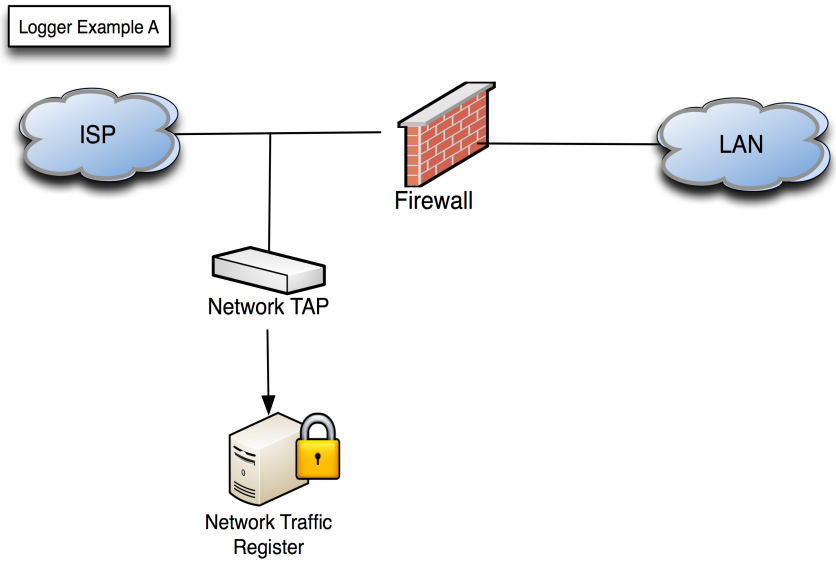

Figure 4.1: Example A from where to install Network Traffic Register on the network

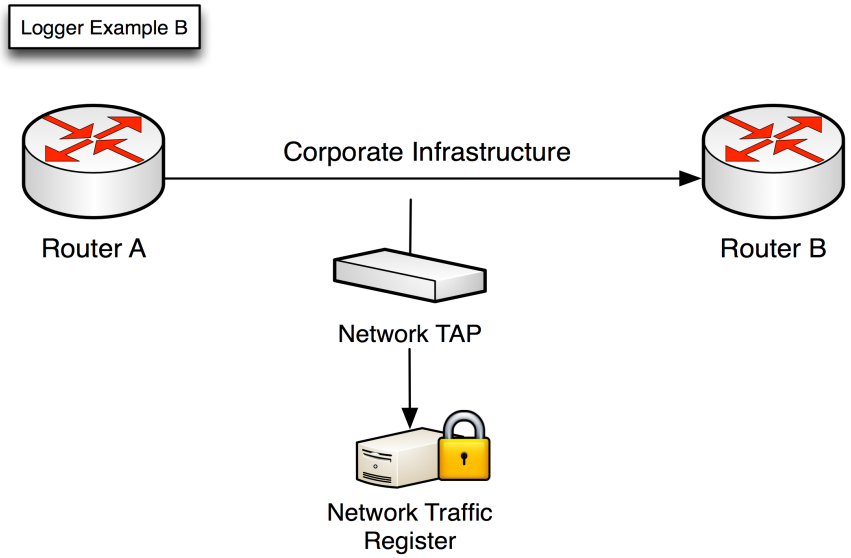

Figure 4.2: Example B from where to install Network Traffic Register on the network

4.1 Hardware and Network configuration

In this small project we used two HP Compaq dc770 Small Form Factor this are available on Amazon for 77€this means that with this kind of hardware almost every companies can implement this Software without the need of high end servers. One of the servers is used to generate packets and the other is used to perform PCap, both servers have two Giga Bytes of Random Access Memory (RAM), they are connected directly by Ethernet interface and have two cores. The server that generate network packets is named Sender, this server has an Wireless interface installed that allows to perform administration and configuration on both servers. The server on where we perform coding and tested the developed software his named Leech.

4.2 Operating System and their tools

For Operating System we opted for Ubuntu Server version 11.10 at the start of this project it was the most actual stable version that was released. We chosen this operating system because it has a great community support and the most packages (RPM files) can be install on this type of system. We could not choose any type of operating system that comes with licenses cost like Windows, but for a truly answer this work would be a lot difficult to implement if our choice was Windows server.

One tool that we used to allow space flexibility was Logical Volume Manager (LVM), this is a well used concept that basically allows on the fly the expansion and reduction of file system, for more information about this subject you should consult a book like Understanding the Linux Kernel, Third Edition (25).

4.2.1 Linux Kernel

Old linux kernels have some problems with how they deal with high packets rates(15) (16). To avoid the problems referred above we decided to use the latest stable Linux Kernel released. Ubuntu Server 11.10 comes already with Kernel 3.00, at the time of developing this work the available Linux Kernel was the version 3.36. To install the kernel we download the referred version and them the main commands execute to install the new Linux Kernel were:

Listing 4.1: Encrypt Server configuration file example

```
make modules
make install
make modules_install install
cd /boot/
mkinitramfs -o initrd.img-3.3.6 3.3.6
```

After this steps you need to reboot the system and boot with the new Kernel. We were able to capture all traffic without having packet lost as showed in 4.3.

4.2.2 Ramdisk

Ramdisk is actually a relative old method used when applications are very disk intensive. With this concept what is done is allocate a block of RAM on the server, let's say 1 Giga Byte. This allocate space in them mapped directly on file system. The Operating System them write files directly on RAM increasing drastically the reads and writes times of applications that use this filesystem. The downside of using this technique is that if the mapped directory on the filesystem get full them the application is in really trouble and would not be to operate in a stable mode. We have performed several tests with this type of concept, but as showed in Figure 4.3 the difference between using Ram Disk concept or not at the Encryption speed level are not relevant and it is not worth to implement. Another point notice during the tests is that running Daemonlogger to capture packets at 180 Kb/s or 380 Kb/s does not make really difference on the encryption processes. Still in Figure 4.3 you can notice that time to encrypt and file size are well-proportioned, this means that when the file to encrypt is bigger it takes more time to encrypt.

4.2.3 Redundant Array of Independent Disks

Redundant Array of Independent Disks (RAID) is a technology that uses multiple physical disks combining them into one faster or redundant (failure tolerant) disk. After configuration a file that is usually stored in one physical disk, now depending on the RAID configuration the file is stored on many disks (allowing higher writing speeds) or mirrored (allowing disk failures). Now a days in possible to implement RAID based on software or in dedicated hardware. You can implement this kind of technology by mixes

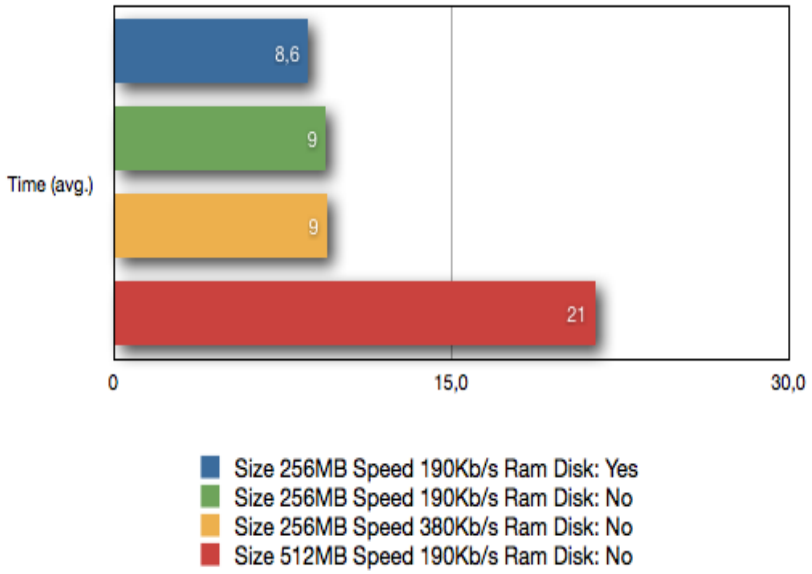

Figure 4.3: Encryption speed with Ram Disk

grades of RAID, usually the most implemented is the RAID 0 or RAID 1. [1] In this project we could not afford the implementation of this kind of technology, but it if we had this opportunity we will have an increase of overall all performance on the server.

4.3 Testing Kernel and Full Packet Capture on Work enviroment

At this point we have explain to the reader how test and development environment was setup. After we perform the installation of Linux Kernel and Daemonlogger software we need to assure that PCap will be not affected wen the server is with high load. We perform several tests to verified that the new Linux Kernel and Daemonlogger were functioning with the desired performance. As showed in Figure 4.4 we could achieve capture speeds up to 883 Mega bytes per second without any packet lost. This is a very good result, knowing that we use a commodity on board network interface (One Giga byte interface). To carry out this network load tests we used a tool named

[1] We found a really god explanation about this subject on wikipedia on `http://en.wikipedia.org/wiki/RAID`, normaly we would not recomend the reader to visit this kind of sites. But in this case it is a good documentation and explanation.

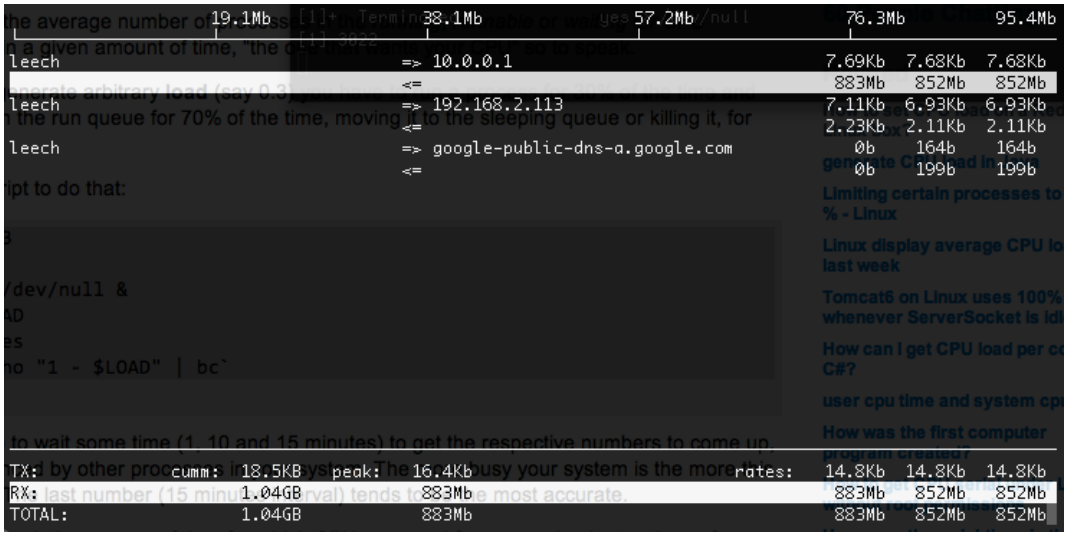

Figure 4.4: Packet Capture speed rates

Linux Packet Generator (PKTGEN). There are other network traffic generators like Nemessis or Scapy but we opted for PKTGEN because there is no need to install any software on our sender server, PKTGEN comes integrated on the Linux Kernel. The chosen network traffic generator is a highly aggressive tool and you can quickly reach the limits of routers, switches or any server receiving this kind of traffic.[1] At this point we fully tested the network performance and we are satisfied, but this implementation allow further growth up to 10 Giga bytes per second. Another point that needed to be tested is CPU, as this is one of critical aspects referred on (15) and could significantly compromise this work. Until now when PCap is running at maximal speeds (over 750 Mb/s) we actually only have one CPU busy and the load of the system is completely normal (about 1.3). For testing the CPU load on our server the process was simple, start PCap with Daemonlogger and them generate some load on the server with simple cycle scripts as show on Figure 4.5. We achieve systems loads more them 5, and parallel we measure packet lost or packet errors, in this set of test we didn't find any packet lost.

Again, in the in of this project we achieve with no problems capturing speeds of 650 Mega Bytes per second plus encryption up to two processes in parallel. If this is not enough

[1]For more information about this tools you should consult the manual page at `http://www.linuxfoundation.org/collaborate/workgroups/networking/pktgen`

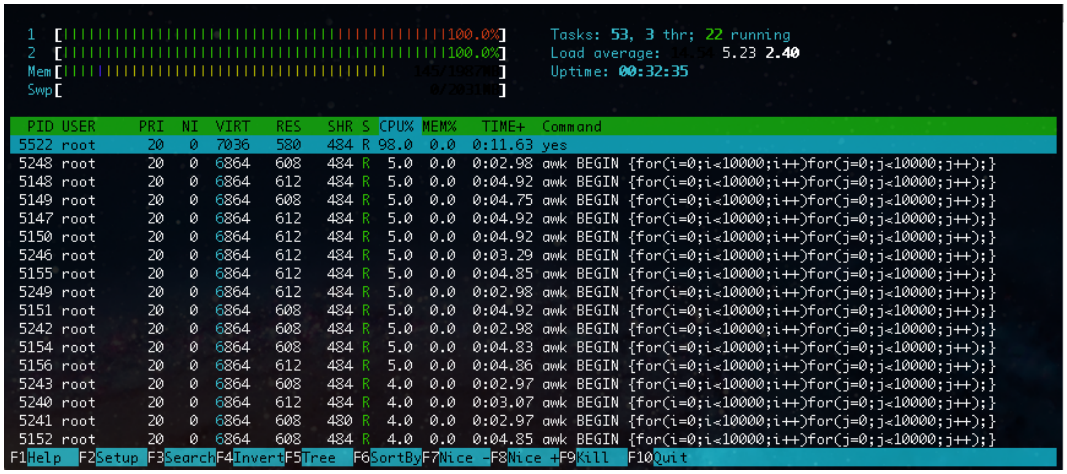

Figure 4.5: Test of CPU Load

to the common organizations it is possible with small effort to add a new Daemonlogger server and a new Encrypt server. Fabian Schneider et al on (14) showed one possible approach using an switch that is capable of 10 Giga bytes speeds and the another gold functionality is that allows load balancing connections to another ethernet ports. With this technique we can install several servers that deal with the PCap and encryption without losing packets. As our budged was limited we could not make tests with this kind of hardware, but we are certain that it will work.

4.4 Performance vs Number of encryption processes

The performance of this application (when in single installation) is directly depend on the number of parallel Encryption process. While performing tests on the presented commodity hardware, we notice that using more than two encryptors processes (parallel encryption, the processes are actually performing encryption) leads to an unstable system.

In this way for the hardware that we mentioned we recommend the usage of two encryptors. If the queue of the message server is getting to long, it means that the capturing data from the network is being faster them encryption. In this point you should check your hardware and them increment the number of parallel encryptors. At least if upgrading to an faster hardware is not possible you can always deploy a new encryptor instance on a different hardware.

4.5 Chapter Conclusion

In this Chapter the reader have learned about what type of hardware we used and the main configuration of it. We performed tests to evaluate the performance of Ram Disk and showed that it is not worth to implement it, at the end of the Chapter we showed to the reader tests that were performed to assure that the basic software, hardware and configuration where up to the challenge.

Chapter 5

Conclusion

Chapter 1 presents the motivation that lead the authors to pursue this work. Next, the reference scenario and the work objectives are detailed. The objectives being the full packet capture of network traffic, its confidentiality and integrity, and the ability to continuously operate be rotating stored data. The Chapter ends by identifying the results expected in the end of the work.

Chapter 2 identifies and describes the different types of packet captures that more commonly used. Some tools that perform packet capture are presented next (Section 2.2). A basic description of Cryptography is also presented. Ending the Chapter, the work, performed by others that is related to the presented work, is detailed.

Chapter 3 details the proposed solution, enumerting it's three software modules. The interaction between modules is depicted in a sequence chart and textually detailed. In particular, the interaction between the the Encrypt server, or the Daeminlogger, with the Message server is described. Next, the problematic of the implementation of the proposed solution is a real scenario is discussed and several suggestions are made. Finally, all configuration options of the Message server and Encrypt server modules are detailed.

Chapter 4 presents some validation tests performed on the installed system that lead to the conclusion that the combined use of Daemonlogger and the most recent Linux Kernel, the problems referenced on the literature were overcome.

5.1 Relevant Results

The prototype of the proposed solution constitutes the key result of the present work. The prototype was built on top of the Daemonlogger in order to add to its FPCap capability, the required functionalities to cope with the security requirements, such as confidentiality and integrity, but also to support continuous operation and to be scalable. The use of RAM disks to boost encryption performance was evaluated, but it resulted in no significant performance. The prototype was able to capture and encrypt at speeds of 650MB/s, using two processes in parallel. After the encryption is done, the only way to access to original file is to possess the private key.

5.2 Future Work

The securing of the communications between modules is secured, has it is assumed to happen in a separated and dedicated network. Nonetheless, it is easily envisaged that some implementations scenarios may require this communications to be secured. Future work can then include the use of TSL/SSL to secure the communication between modules, using the certificates for bidirectional authentication.

Bibliography

[1] Cisco, "The zettabyte era," May 2012. 1

[2] D. Belson, "The state of the internet," *Akamai*, vol. Volume 5, no. 2, p. 40, 2012. 1

[3] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the 13th USENIX conference on System administration*, ser. LISA '99. Berkeley, CA, USA: USENIX Association, 1999, pp. 229–238. [Online]. Available: http://dl.acm.org/citation.cfm?id=1039834.1039864 5, 6

[4] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Proceedings of the 7th conference on USENIX Security Symposium - Volume 7*, ser. SSYM'98. Berkeley, CA, USA: USENIX Association, 1998, pp. 3–3. [Online]. Available: http://dl.acm.org/citation.cfm?id=1267549.1267552 5, 6

[5] J. R. Vacca, Ed., *Computer and Information Security Handbook*. Morgan Kaufmann Publishers, 2009, ch. 18, pp. 302, 305. 5

[6] E. B. Leon Ward, "Openfpc," 03 2012. [Online]. Available: http://www.openfpc.org/documentation/about 5, 7

[7] M. Roesch, "Daemonlogger," 09 2012. [Online]. Available: http://www.snort.org/snort-downloads/additional-downloads 5, 7, 12

[8] TCPDUMP.org, "Tcpdump," 10 2012. [Online]. Available: http://www.tcpdump.org/ 5, 6

[9] S. Kornexl, V. Paxson, H. Dreger, A. Feldmann, and R. Sommer, "Building a time machine for efficient recording and retrieval of high-volume network traffic," 2005. 5, 6

[10] TCPDUMP.org, "Packet capture with libpcap and other low level network tricks," 3 2008. 7

[11] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*, 2nd ed., P. Sutherland, Ed. New York, NY, USA: John Wiley and Sons, Inc., 1995. 8

[12] A. Kuchling, "Python cryptography toolkit," p. 16, 2008. 8

[13] S. Oaks, *Java Security, 2nd Edition*, ser. Java Series. O'Reilly & Associates, 2001. [Online]. Available: http://books.google.de/books?id=EhX9BjHj9M4C 9

[14] F. Schneider, J. Wallerich, A. Feldmann, S. Uhlig, K. Papagiannaki, and O. Bonaventure, "Packet capture in 10-gigabit ethernet environments using contemporary commodity hardware," 2007. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-71617-4_21 10, 31

[15] L. Deri, N. S. P. A, V. D. B. Km, and L. L. Figuretta, "Improving passive packet capture: Beyond device polling," 2004. 10, 27, 30

[16] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," *ACM Trans. Comput. Syst.*, vol. 15, no. 3, pp. 217–252, Aug. 1997. [Online]. Available: http://doi.acm.org/10.1145/263326.263335 10, 27

[17] O. Peter and S. Tamas, "Software-based packet capturing with high precision timestamping for linux," 2010. 10

[18] C. Morariu and B. Stiller, "Dicap: Distributed packet capturing architecture for high-speed network links," 2008. [Online]. Available: http://www.zora.uzh.ch/7163/ 10

[19] C. Benvenuti, *Understanding Linux network internals.* O'Reilly, December 2005. 10

[20] L. Deri, "ncap: Wire-speed packet capture and transmission," in *in Proceedings of the IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON*, 2005, p. 2. 10

[21] L. Schaelicke, K. Wheeler, and C. Freeland, "Spanids: A scalable network intrusion detection loadbalancer," 2005. 10

[22] M. Dashtbozorgi and M. A. Azgomi, "A high-performance software solution for packet capture and transmission," *Computer Science and Information Technology, International Conference on*, vol. 0, pp. 407–411, 2009. 10

[23] E. Bendersky, "Aes encryption of files in python with pycrypto." [Online]. Available: http://eli.thegreenplace.net/2010/06/25/aes-encryption-of-files-in-python-with-pycrypto/ 18

[24] A. Holovaty and J. Kaplan-Moss, *The Definitive Guide to Django: Web Development Done Right, Second Edition*, 2nd ed. Berkely, CA, USA: Apress, 2009. 23

[25] D. P. Bovet and M. C. Ph, *Understanding the Linux Kernel Third Edition*, third edition ed. O'Reilly Media, Nov. 2005. [Online]. Available: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0596005652 27

Appendix A

Annexes

Listing A.1: Message Server source code

```
import SocketServer, ConfigParser, sys, signal, socket, thread, time
from multiprocessing import Process, Lock
from Crypto.Hash import SHA
from multiprocessing import Queue
global configuration_file
configuration_file=None
global conn
conn = None
global encryptors
global queue_to_encrypt
global loggers
global server
global lock
global automatic_distri
class Encrypt_server:
    name = None
    ip_adresse = None
    port = 0
    key  = None
    processes = 0
    running = 0
    encryption_running = []
    lock = Lock()


    def get_running_processes(self):
        return self.running
```

```
def unresgister_from_my_self(self):
    global encryptors
    received = ""
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.connect( ( self.ip_adresse , int(self.port) ))
        sock.settimeout(4)
        #sock.settimeout(4)
        sock.sendall("HELLO"+"\n")
        received = sock.recv(1024)
        if received == "ENCRYPT_SRV":
            sock.sendall("UNREGISTER:"+self.key+"\n")
            received = sock.recv(1024)
    except socket.error:
        pass
    counter = 0
    for a in encryptors:
        if a.key == self.key and a.name == self.name:
            del encryptors[counter]
        counter = counter +1
    if received =="OK":
        print "Returning True for unresgister..."
        return True
    elif received == "WORKING":
        print "Returning False for unresgister... reason working"
        return False
    else:
        print "I cannot contact my encryptor"
        return True




def add_work(self, request):
    global queue_to_encrypt
    #Request is like:
    #   L:10.0.0.2:ficheiro1:5000
    diff = int(self.processes) - int(self.running)
    if (diff < 1):
        return False
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(4)
    try:
        # Connect to server and send data
```

```
sock.connect( ( self.ip_adresse , int(self.port) ))
sock.sendall("HELLO"+"\n")
received = sock.recv(1024)
splitted_request = request.split(":")
#Request is like:
#   L:10.0.0.2:ficheiro1:5000
data = ""
try:
    data = "ENCRYPT:%s:%s:%s"%(splitted_request[1],
        splitted_request[2],splitted_request[3])
except IndexError:
    print "ERROR !!!!!!!!"
    sock.close()
    return False
sock.sendall(data+"\n")
received = sock.recv(1024)
received = received.split(":")
if received [0] =="ENCRYPT_STARTED":
    self.lock.acquire()
    self.running=self.running+1
    print "Appending to running encryption : %s"%(request)
    self.encryption_running.append(request)
    self.lock.release()
    sock.close()
    return True

elif received[0] == "IAM_BUSY":
    print "I am busy "+self.name
    self.lock.acquire()
    self.running = self.processes
    sock.close()
    return False

elif received[0] == "REQ_ALREADY_IN_QUEUE":
    queue_to_encrypt.remove(request)
    if self.encryption_running.count(request) == 0:
        self.encryption_running.append(request)

    print "Request already in Queue from encryptor"
    return True

else:
    print "Error on register_work. add_work()"
    sock.close()
    return False
```

```
        except (IndexError):
            print "Index Error lin 76"
        except socket.error:
            #Cannot get connection... unregister
            self.unresgister_from_my_self()

        finally:
            sock.close()
    def get_work_in_progress(self):
        return self.encryption_running

    def remove_work(self,request):
        try:
            self.running = int(self.running) - int(1)
            if self.running < 0:
                self.running = 0
            self.encryption_running.remove(request)

        except ValueError:
            print "Error on Remove work"
            pass
        return True
    def generate_self_key(self):
        h = SHA.new()
        lixo=self.name+self.ip_adresse
        h.update(b'%s'%(lixo))
        self.key=h.hexdigest()
    def get_self_key(self):
        return self.key
    def is_free(self):
        diff = int(self.processes) - int(self.running)
        if diff > 0:
            return True
        return False

    def __str__(self):
        return self.ip_adresse+":"+self.name

    def __init__(self,name,ip_adresse,port, processes):
        self.name=name
        self.ip_adresse = ip_adresse
        self.processes = processes
        self.running = 0
        self.port = port
        self.generate_self_key()
```

```
class Log_server:

    name = None
    ip_adresse = None
    key = None


    def generate_self_key(self):
        h = SHA.new()
        lixo=self.name+self.ip_adresse
        h.update(b'%s'%(lixo))
        self.key=h.hexdigest()

    def get_self_key(self):
        return ""+self.key

    def __init__(self,name,ip_adresse):
        self.name=name
        self.ip_adresse=ip_adresse
        self.generate_self_key()

    def __str__(self):
        return self.ip_adresse+":"+self.name


def get_free_encrytor(encryptors):
    for a in encryptors:
        if a.is_free():
            return a
    return None


def distribute_queue(queue,encryptors, lock, loop=False, sleep=60):
    #Racing function, muss perform locking....  and check loopings
    if loop ==True:
        print "Sleep in function %i"%(sleep)
        while True:
            lock.acquire()
            distributed_queue = []
            for a in queue:
                free_encryptor=get_free_encrytor(encryptors)
                if free_encryptor != None:
                    if free_encryptor.add_work(a):
                        distributed_queue.append(a)
```

```
                else:
                    print "Nothing is free or registed cannot add work : %
                        s"%(a)
            #Remove work distributed from queue...
            for a in distributed_queue:
                queue.remove(a)
            lock.release()
            time.sleep(sleep)
    else:
        lock.acquire()
        distributed_queue = []
        flag = True
        for a in queue:
            free_encryptor=get_free_encrytor(encryptors)
            if free_encryptor != None and free_encryptor.add_work(a):
                distributed_queue.append(a)
            else:
                print "Nothing is free"
                flag = False
        #Remove work distributed from queue...
        for a in distributed_queue:
            queue.remove(a)
        print "Queue left : %s"%(queue)
        lock.release()
        return flag


def register_encryption(ip_adresse,file_to_encrypt,size,queue_to_encrypt):
        #verify if request already in queue
        element="L:"+ip_adresse+":"+file_to_encrypt+":"+size
        for a in queue_to_encrypt:
            if a==element:
                return False
        queue_to_encrypt.append(element)
        return True

def check_if_logger_registed(logger, ip,loggers):
    for a in loggers:
        if a.name == logger and a.ip_adresse == ip:
            return a
    return False

def check_if_encryptor_registed(encryptor, ip,encryptors):
    for a in encryptors:
        if a.name == encryptor and a.ip_adresse == ip:
```

```
            return a
    return False

def is_authenticated(key,name,ip_adresse,loggers):
    for a in loggers:
        if a.key == key and a.name == name and a.ip_adresse == ip_adresse:
            return True
    return False

def is_callback_authenticated(key,encryptors):
    for a in encryptors:
        if a.key == key:
            return a
    return False


class MyTCPHandler(SocketServer.StreamRequestHandler):
    """
    The RequestHandler class for our server.

    It is instantiated once per connection to the server, and must
    override the handle() method to implement communication to the
    client.
    """
    def handle(self):
        global encryptors
        global configuration_file
        global lock
        # self.rfile is a file-like object created by the handler;
        # we can now use e.g. readline() instead of raw recv() calls
        try :
                print "I have RECIVED a request"
                self.data = self.rfile.readline().strip()
                if self.data =="HELLO":
                    self.wfile.write("MSG_SRV")
                    data_from_sender=self.rfile.readline().strip().split("
                        :")
                    if data_from_sender[0]=="" or data_from_sender[1]=="":
                        print "No data"
                        return
                    if data_from_sender[0]=="REGISTER":
                        if data_from_sender[1] =="L":
                        #Quero registar um logger
                            is_registe_logger = check_if_logger_registed(
                                data_from_sender[2],self.client_address
```

```
                    [0], loggers)
                if is_registe_logger:
                    print "Already Registered"
                    self.wfile.write(data_from_sender[2]+":
                        ALREADY_REGISTED:%s\0"%(
                        is_registe_logger.get_self_key()))
                    print "Data to send: "+data_from_sender
                        [2]+":REGISTED:L:%s\0"%(
                        is_registe_logger.get_self_key())
                else:
                    a = Log_server(data_from_sender[2], self.
                        client_address[0])
                    loggers.append(a)
                    self.wfile.write(data_from_sender[2]+":
                        REGISTED:L:%s\0"%(a.get_self_key()))
                    print "Data to send: "+data_from_sender
                        [2]+":REGISTED:L:%s"%(a.get_self_key()
                        )
                    print "Logger Registered: {}".format(
                        data_from_sender[2])
            elif data_from_sender[1] == "E":

                is_encrytor_registed =
                    check_if_encryptor_registed(
                    data_from_sender[2], self.client_address
                    [0], encryptors)
                if is_encrytor_registed:
                    print "Already Registered"
                    self.wfile.write(data_from_sender[2]+":
                        ALREADY_REGISTED:"+
                        is_encrytor_registed.get_self_key())
                    return False
                else:
                    #verificar se foi fornecido o numeor de
                        processos para o servidor de
                        encryptacao

                        if data_from_sender[4].isdigit() and
                            data_from_sender[4]>0 and int(
                            data_from_sender[4]) <=
                            configuration_file.getint('General
                            ','max_encryt_processes'):
                            a = Encrypt_server(
                                data_from_sender[2], self.
                                client_address[0],
```

```
                                    data_from_sender[3],
                                    data_from_sender[4])
                            encryptors.append(a)
                            self.wfile.write(data_from_sender
                                [2]+":REGISTED:E:%s"%(a.
                                get_self_key()))
                        else:
                            print "Wrong header, probaly
                                max_encryt_processes parameter
                                "
                            self.wfile.write("
                                WRONG_REGISTER_HEADER")
                    else:
                        print "Wrong Encrtor Header"
                        self.wfile.write("WRONG_REGISTER_HEADER")
                        #quero registar um encrpytor
                elif data_from_sender[0] == "REQ_ENCRYPT":
                    #Check if request is authenticate
                    b = is_authenticated(data_from_sender[1],
                        data_from_sender[2], self.client_address[0],
                        loggers)
                    if b == True:
                        #Register encryption on queue, check if
                            something is free and write the anwerser
                            in the socket.
                        #String of data must be sometinh like :
                            REQ_ENCRYPT:secretkey:name:file_to_encrypt
                            :size
                        if register_encryption(self.client_address[0],
                            data_from_sender[3],data_from_sender[4],
                            queue_to_encrypt):
                            self.wfile.write("REQ_REGISTED:%s"%(
                                data_from_sender[3]))
                            p = thread.start_new_thread(
                                distribute_queue,(queue_to_encrypt,
                                encryptors,lock,))
                            print "Distributing Queue in Progress..."
                            for a in encryptors:
                                print "------ work in progress from
                                    encryptors"
                                print "      %s"%(a.name)
                                print "      %s"%(a.
                                    get_work_in_progress())

                        else:
```

```
                self.wfile.write("DUPLICATED_REQUEST")
                print "Faild to register encryption"
        else:
            print "NOT_AUTHORIZED"
            self.wfile.write("NOT_AUTHORIZED")
    #Call_back from encrypt_server
    elif data_from_sender[0] == "DONE":

        print "RECIVED CALLBACK {}".format(
            data_from_sender)
        a = is_callback_authenticated(key=data_from_sender
            [1], encryptors=encryptors)
        print "Before remove work from the object : %s"%(a
            .get_work_in_progress())
        if a != False:
            a.remove_work("L:"+data_from_sender[3]+":"+
                data_from_sender[4]+":"+data_from_sender
                [5])
            self.wfile.write("OK")
            print "After remove work from the object : %s"
                %(a.get_work_in_progress())
            thread.start_new_thread(distribute_queue,(
                queue_to_encrypt,encryptors,lock,))
            print "Distributing Queue in Progress..."
            for a in encryptors:
                    print "----- work in progress from
                        encryptors"
                    print "     %s"%(a.name)
                    print "     %s"%(a.
                        get_work_in_progress())
            return True
        else:
            print "NOT_AUTHORIZED"
            self.wfile.write("NOT_AUTHORIZED")
    elif data_from_sender[0] =="FILE_DOES_NOT_EXIST":
        print "RECIVED CALLBACK FILE DOES NOT EXIST {}".
            format(data_from_sender)
        a = is_callback_authenticated(key=data_from_sender
            [1], encryptors=encryptors)
        if a != False:
            a.remove_work("L:"+data_from_sender[2]+":"+
                data_from_sender[3]+":"+data_from_sender
                [4])
            self.wfile.write("OK")
```

```
                            p = thread.start_new_thread(distribute_queue,(
                                queue_to_encrypt,encryptors,lock,))
                            print "Distributing Queue in Progress..."
                            for a in encryptors:
                                    print "----- work in progress from
                                        encryptors"
                                    print "      %s"%(a.name)
                                    print "      %s"%(a.
                                        get_work_in_progress())
                        else:
                            print "NOT_AUTHORIZED"
                            self.wfile.write("NOT_AUTHORIZED")
                            return False
                    elif data_from_sender[0] == "UNREGISTER":

                        for a in encryptors:
                            if a.key == encrytor.key and a.client_address
                                == encrytor.ip_adresse:
                                a.unresgister_from_my_self()
                                self.wfile.write("OK")
            except IndexError as e:
                print "Sorry from Register Header index: {}".format(
                    data_from_sender)
                print e
                self.wfile.write("WRONG_HEADER")

            print "Encryptors"
            for a in encryptors:
               print "Encryptor: %s %s %s"%(a.ip_adresse, a.key, a.name)
            print "Queue to encrypt: %s"%(queue_to_encrypt)

            #print "\nQueue\n"
            #for a in queue_to_encrypt:
            #    print a
            #print "{} wrote:".format(self.client_address[0])
            #print self.data
            # Likewise, self.wfile is a file-like object used to write back
            # to the client
            #self.wfile.write(self.data.upper())

def load_config():

    #load and apply configs from file
    try:
        global configuration_file
```

```
        configuration_file = ConfigParser.RawConfigParser()
        configuration_file.read('./msgsrv_setting/logger.cfg')
    except ConfigParser.NoSectionError:
        print "I cannot find the logger.cfg file. Make sure that is in ./
            msgsrv_setting/logger.cfg"
        sys.exit()

def signal_handler(signal, frame):
        print 'You pressed Ctrl+C!'
        global encryptors
        global server
        global automatic_distri
        positions_for_delete = []
        counter_for_up_encryptions = 0
        counter = 0
        for a in encryptors:
            if len(a.get_work_in_progress()) == 0:
                unresgister_rsp = a.unresgister_from_my_self()
                if unresgister_rsp == True:
                    positions_for_delete.append(counter)
                elif unresgister_rsp == "WORKING":
                    counter_for_up_encryptions =
                        counter_for_up_encryptions +1
            else:
                counter_for_up_encryptions = counter_for_up_encryptions +1
            counter = counter + 1

        if counter_for_up_encryptions >0:
            print "My encrytors are still processing please wait until
                they finished, i will terminate in max 60 secs"
            time.sleep(5)
            counter_for_up_encryptions = 0
            counter = 0
            for a in encryptors:
                if len(a.get_work_in_progress()) == 0:
                    unresgister_rsp = a.unresgister_from_my_self()
                    if unresgister_rsp == True:
                        positions_for_delete.append(counter)
                    elif unresgister_rsp == "WORKING":
                        counter_for_up_encryptions =
                            counter_for_up_encryptions +1
                else:
                    counter_for_up_encryptions =
                        counter_for_up_encryptions +1
                counter = counter + 1
```

```
        else:
            print "No pending encryption and encrytors where notifyed"

        if automatic_distri != None:
            automatic_distri.terminate()
        #delete encrytors...
        #check if queue is empty before write
        if len(queue_to_encrypt) != 0 :
            print "Writing queue to file ./pending_work.work..."
            with open("pending_work.work","w") as pending_work_file:
                for a in queue_to_encrypt:
                    pending_work_file.write(a+"\n")

        thread.start_new_thread(server.shutdown,())
        sys.exit(0)
def automatic_distribution(time_sleep):
        global queue_to_encrypt
        global encryptors
        global lock
        time.sleep(int(time_sleep))
        while True:
            print "automatic_distribution working..."
            distribute_queue(queue_to_encrypt,encryptors,lock)
            print "Ended automatic_distribution..."
            time.sleep(int(time_sleep))

if __name__ == "__main__":
    load_config()
    global loggers
    loggers=[]
    global encryptors
    encryptors=[]
    global queue_to_encrypt
    queue_to_encrypt=[]
    global server
    global lock
    lock = Lock()
    global automatic_distri
    automatic_distri = None
    #encrypt_in process are in objects from encryptors, in the queue are
        only the objects that are not distributed
    signal.signal(signal.SIGINT, signal_handler)
    HOST, PORT = configuration_file.get('General','msg_server_ip'),
        configuration_file.getint('General','msg_server_port')
    try:
```

```
        print "Reading ./pending_work.work..."
        with open("pending_work.work","r") as pending_work_file:
            for line in pending_work_file:
                queue_to_encrypt.append(line.rstrip())
            pending_work_file.truncate(0)
    except IOError:
        pass

    if configuration_file.getboolean('General','automatic_distri_queue'):
        print "Starting Automatic Queue Distribuing with time of %s..."%(
            configuration_file.getint('General','
            sleep_time_for_automatic_distri'))
        thread.start_new_thread(automatic_distribution,(configuration_file
            .getint('General','sleep_time_for_automatic_distri'),))
    print "Starting Message Server on {}:{}".format(HOST,PORT)
    # Create the server, binding to localhost on port 9999
    server = SocketServer.TCPServer((HOST, PORT), MyTCPHandler)

    # Activate the server; this will keep running until you
    # interrupt the program with Ctrl-C
    server.serve_forever()
    print "Exiting..."
```

Listing A.2: Message Server source code

```
import SocketServer, ConfigParser, sys, signal, socket, os, sys, thread,
    time, MySQLdb
from multiprocessing import Pool, Lock
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_v1_5
from encryptfunction import encrypt_file,decrypt_file
from Crypto.Hash import SHA
from Crypto.Cipher import AES
from Crypto import Random
from datetime import datetime


#Global Configurations
global configuration_file
configuration_file=None
global lock
lock = Lock()
global my_key
my_key = None
global my_processes_running
```

```
my_processes_running = 0
global max_processes_running
max_processes_running = 0
global work_in_progress
pool = None
global server

#this import need to be here because of the global variables !
def disk_manager(file_name):
    #global configuration_file
    #file_name = configuration_file.get('Encrypt_server','
        encrypted_files_location')+os.path.basename(file_name)
    configuration_file = load_config_localy()
    print "Start disk_manager at %s"%(datetime.now())
    space_needed=os.path.getsize(file_name)
    print "I need this Space: %s"%(space_needed)
    conn = MySQLdb.connect(host=configuration_file.get('General','db_host'
        ),user=configuration_file.get('General','db_user'),passwd=
        configuration_file.get('General','db_passwd'),db=
        configuration_file.get('General','db_name'))
    cursor = conn.cursor()
    flag = True

    while(flag):
        print "I'm on the cycle from disk_manager for {}".format(file_name
            )
        disk = os.statvfs(configuration_file.get('Encrypt_server','
            encrypted_files_location'))
        available_space = ((disk.f_frsize * disk.f_bavail) / 1024) /1024
        space_needed = (space_needed / 1024 ) / 1024

        if (available_space  < (space_needed + 1024)):
            print "Releasing space..."
            #need to clean up. delete file on db and file system
            #print "Cleanning up %s"%(file_name)
            cursor.execute(u'SELECT nome, data_entrada FROM
                web_logger_ficheiro WHERE locked = "0" ORDER by
                data_entrada asc limit 500')
            row = cursor.fetchone()
            if row is not None:
                file_name = row[0]
            else:
                #check if is space available in filesystem if not exit
                if (available_space  < (space_needed + 1024)):
```

```
                    print "Something really bad happend, filesystem is
                        full i think..\n"
                    sys.exit(-1)
            enc_file_to_rm = "%s.enc"%(os.path.basename(file_name))
            sign_file_to_rm = "%s.signature"%(os.path.basename(file_name))
            os.system('rm %s%s'%(configuration_file.get('Encrypt_server','
                encrypted_files_location'),enc_file_to_rm))
            os.system('rm %s%s'%(configuration_file.get('Encrypt_server','
                encrypted_files_location'),sign_file_to_rm))
            cursor.execute(u'DELETE from web_logger_ficheiro where nome="%
                s"'%(file_name))

        else:
            #Add a new meta-data in database
            date_today = "%s"%(datetime.now())
            date_today = date_today[:-7]
            file_name=os.path.basename(file_name)
            try:
                cursor.execute("""
                    INSERT INTO web_logger_ficheiro (nome, data_entrada,
                        locked, size)
                    VALUES
                        ('%s','%s','%s','%s')

                    """%(file_name,date_today,0,space_needed))
            except Exception as e:
                #log exception !
                print "Panicccc line 64"
                print e
                return False

            #print "I have space"
            flag = False
    cursor.close()
    conn.close()
    print "Finish disk_manager at %s"%(datetime.now())

def save_meta_data(file_to_encrypt):
    pass

def save_signature(KeyToSave,Public_Cipher,FileName):

    with open(FileName+".signature",'wb') as writefile:
        #ever time returns an 20 bytes long hash
        #h = SHA.new(writefile.read())
```

```
        writefile.write(KeyToSave)

def start_ciphering(file_to_encrypt):

    pubkey = RSA.importKey(open(configuration_file.get('Encrypt_server', '
        pubkey_location')).read())
    random_bytes = os.urandom(32)
    #pid = os.fork()
    #if pid == 0 :
    disk_manager(file_name=file_to_encrypt)
    #sys.exit()
    #encrypt file with 256MB chunks and 512Mb file size takes about 17
        seconds
    print "Start cifer for %s at %s"%(file_to_encrypt, datetime.now())
    encrypt_file(key=random_bytes, in_filename=file_to_encrypt,
        out_filename=configuration_file.get('Encrypt_server','
        encrypted_files_location')+os.path.basename(file_to_encrypt+'.enc'
        ), chunksize=configuration_file.getint('Encrypt_server','chunksize
        '))
    #cifer random key to post decrypt
    public_cipher=PKCS1_v1_5.new(pubkey)
    ciphertext = public_cipher.encrypt(random_bytes)
    #doing this on a fuction so i can perform saves on databases to (later
         on the project)
    save_signature(KeyToSave=ciphertext, Public_Cipher=public_cipher,
        FileName=configuration_file.get('Encrypt_server','
        encrypted_files_location')+os.path.basename(file_to_encrypt))
    print "Ended cifer for %s at %s\n"%(file_to_encrypt, datetime.now())
    os.system('rm %s'%(file_to_encrypt))
    return "DONE:%s"%(file_to_encrypt)

def check_if_logger_registed(logger, ip,loggers):
    for a in loggers:
        if a.name == logger and a.ip_adresse == ip:
            return True
    return False

def do_the_work(ip_adresse, file_name, file_size):
    #load local confif
    configuration_file = load_config_localy()
    #need this variable because on msg if this name registed
    old_return_name = file_name
    file_name = os.path.basename(file_name)
    #get the file from logger
    if not configuration_file.getboolean('Encrypt_server','all_in_one'):
```

```
        #muss transfer the file to the server
        disk = os.statvfs(configuration_file.get('Encrypt_server','
            tmp_encry_location'))
        available_space = ((disk.f_frsize * disk.f_bavail) / 1024) /1024
        space_needed = (file_size / 1024 ) / 1024
        if space_needed + 100 < available_space:
            a =os.system("scp -B -q %s:%s %s"%(ip_adresse,file_name,
                configuration_file.get('Encrypt_server','
                tmp_encry_location')))
            if a !=0:
                print "PANIC ! I cannot get the remote file "
                return False
        else:
            print "PANIC ! !  No File System   Reasonn : SPACE"
            return False
    else:
        #File already in server only need to move it from daemonlloger
            location to tmp encrypt location
        file_to_move = configuration_file.get('Encrypt_server','
            files_before_encrypt')+file_name
        #print "File to move : "+file_to_move
        #a = os.system("mv %s %s >/dev/null  2>&1"%(file_to_move,
            configuration_file.get('Encrypt_server','tmp_encry_location'))
            )
        a = os.system("mv %s %s"%(file_to_move,configuration_file.get('
            Encrypt_server','tmp_encry_location')))
        if a != 0:

            return_str = "FILE_DOES_NOT_EXIST:%s:%s:%s"%(ip_adresse,
                old_return_name,file_size)
            print "PANIC ! ! I cannot move the file Return string is: %s"%(
                return_str)
            return return_str
    #File is now in tmp_location
    output = start_ciphering(configuration_file.get('Encrypt_server','
        tmp_encry_location')+file_name)
    #Request is like:
            #   ENCRYPT:10.0.0.2:ficheiro1:5000
            #return from start_ciphering is like : DONE:<file_name>
    expected_output = "DONE:%s"%(configuration_file.get('Encrypt_server','
        tmp_encry_location')+file_name)
    if output == expected_output:
        return_str = "ENCRYPT:%s:%s:%s"%(ip_adresse,old_return_name,
            file_size)
```

```
        print "[info] EXTERN_PROCESS_WILL_RETURN_WORK_DONE: %s"%(
            return_str)
        return return_str

    else:
        print "EXTERN_PROCESS_FAILED!"
        return "FAILED"

def callback_function(self):
    global work_in_progress
    global my_processes_running
    global lock
    global my_key

    data = self.split(":")
    data_to_send = ""
    if data[0] == "FILE_DOES_NOT_EXIST":
        print "Call Back File does not exist"
        print data
        #a.remove_work("L:"+data_from_sender[3]+":"+data_from_sender
            [4]+":"+data_from_sender[5])
        data_to_send = "FILE_DOES_NOT_EXIST:"+my_key+":"+data[1]+":"+data
            [2]+":"+data[3]+"\n"

    if self == False or self == None or self =="":
        print "Something whent wrong with the extern process"
        return False

    if data[0] == "ENCRYPT":
        data_to_send = "DONE:"+my_key+":"+":".join(data)+"\n"
    #Sending information back to msg_server

    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.settimeout(2)
    try:
        sock.connect((configuration_file.get('General','msg_server_ip'),
            configuration_file.getint('General','msg_server_port')))
    except socket.error:
        print "Cannot connect to msg_server but work is done..."
    sock.sendall("HELLO"+"\n")
    received = sock.recv(1024)
    print "Data to send in Callback: %s"%(format(data_to_send))
    print "Free: %s"%(my_processes_running)
    sock.sendall(data_to_send)
    if sock.recv(1024) =="OK":
```

```
        #ENCRYPT:10.0.0.2:/log_ramdisk/daemonlogger.pcap.1346518553:1000
        lock.acquire()
        try:

            work_in_progress.remove("ENCRYPT:"+data[1]+":"+data[2]+":"+
                data[3])
        except ValueError:
            print "ERROR on CALLBACK"
        finally:
            if my_processes_running >0:
                my_processes_running = my_processes_running - 1
            else:
                my_processes_running = 0
        lock.release()
        print "Response to msg_srv sended, work_in_progress: %s"%(
            work_in_progress)

def check_if_request_registed(request):
    global work_in_progress
    a = work_in_progress.count(request)
    if a >0:
        return True
    else:
        return False

def stop_server():
    global pool
    global lock
    global my_processes_running
    global work_in_progress
    pool.close()
    lock.acquire()
    my_processes_running = max_processes_running + 1
    lock.release()
    if len(work_in_progress) ==0:
        server.shutdown()
        return True
    print "Waiting 30 seconds..."
    time.sleep(30)
    with open(configuration_file.get('Encrypt_server','
        file_for_work_in_progress'),'w') as work_in_progress_file:
        for a in work_in_progress:
            work_in_progress_file.write(a)
    server.shutdown()
    sys.exit(0)
```

```
class MyTCPHandler(SocketServer.StreamRequestHandler):
    """
    The RequestHandler class for our server.

    It is instantiated once per connection to the server, and must
    override the handle() method to implement communication to the
    client.
    """


    def handle(self):
        # self.rfile is a file-like object created by the handler;
        # we can now use e.g. readline() instead of raw recv() calls
        global pool
        global max_processes_running
        global my_processes_running
        global work_in_progress

        try:

            self.data = self.rfile.readline().strip()
            if self.data =="HELLO":
                self.wfile.write("ENCRYPT_SRV")
                data_from_sender=self.rfile.readline().strip()
                if data_from_sender is "":
                    print "No data"
                    return False
                #Request is like:
                #   ENCRYPT:10.0.0.2:ficheiro1:5000
                if check_if_request_registed(data_from_sender):
                    self.wfile.write("REQ_ALREADY_IN_QUEUE")
                    return False
                data_from_sender = data_from_sender.split(":")
                if  data_from_sender[0] == "ENCRYPT":

                    try:
                        socket.inet_aton(data_from_sender[1])
                        print "Before encry max_processes_running: %s"%(
                            max_processes_running)
                        print "Before encry my_processes_running: %s"%(
                            my_processes_running)
                        if max_processes_running > my_processes_running:
```

```
                            global lock
                            #Don't need now locking but if the server will
                                 go multithreaded them i will need it.
                            lock.acquire()
                            my_processes_running = my_processes_running +1
                            print "Work in progress %s"%(work_in_progress)
                            work_in_progress.append(":".join(
                                data_from_sender))
                            #def do_the_work(ip_adresse, file_name,
                                file_size):
                            pool.apply_async(do_the_work, args = (
                                data_from_sender[1],data_from_sender[2],
                                data_from_sender[3],), callback =
                                callback_function)
                            lock.release()
                            self.wfile.write("ENCRYPT_STARTED")
                        else:
                            print "Already Busy"
                            self.wfile.write("IAM_BUSY")
                        #Copy file (add the local installation or not...)
                        #Check I can run processes
                        # legal
                    except socket.error:
                        self.wfile.write("WRONG_IP")
                elif data_from_sender[0] =="UNREGISTER" and my_key ==
                    data_from_sender[1]:
                    print "Recived UNREGISTER"
                    self.wfile.write("OK")
                    thread.start_new_thread(stop_server,())
                    return

                else:
                    print "Wrong Header"
                    self.wfile.write("WRONG_HEADER")
                    return
            except IndexError:
                print "Index Error"
                self.wfile.write("WRONG_HEADER")

def load_config():

    #load and apply configs from file
    try:
        global configuration_file
        configuration_file = ConfigParser.RawConfigParser()
```

```
        configuration_file.read('./encrsrv_settings/encrypt_srv.cfg')
    except ConfigParser.NoSectionError:
        print "I cannot find the encrypt_srv.cfg file. Make sure that is
            in ./encrsrv_settings/encrypt_srv.cfg"
        sys.exit()

def load_config_localy():
    #Loads the same configs as loadconfig but for the Processes
    #load and apply configs from file
    configuration_file = None
    try:
        configuration_file = ConfigParser.RawConfigParser()
        configuration_file.read('./encrsrv_settings/encrypt_srv.cfg')
    except ConfigParser.NoSectionError:
        print "I cannot find the encrypt_srv.cfg file. Make sure that is
            in ./encrsrv_settings/encrypt_srv.cfg"
        sys.exit()
    return configuration_file

def signal_handler(signal, frame):
        print 'You pressed Ctrl+C!'
        global pool
        pool.close()
        global server
        server.server_close()


if __name__ == "__main__":
    load_config()
    global server
    global work_in_progress
    work_in_progress = []
    max_processes_running = configuration_file.getint('Encrypt_server','
        max_processes')
    pool = Pool(processes=max_processes_running)
    signal.signal(signal.SIGINT, signal_handler)
    print "Starting...\n"
    try:

        with open(configuration_file.get('Encrypt_server','
            file_for_work_in_progress'),'w') as work_in_progress_file:
            for a in work_in_progress_file:
                work_in_progress.append(a)
            work_in_progress_file.truncate(0)
    except IOError:
```

```
    pass

print "Registing this encry_server on msg_server(%s)... "%(
    configuration_file.get('General','msg_server_ip'))
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.settimeout(1)
try:
    # Connect to server and send data
    sock.connect((configuration_file.get('General','msg_server_ip'),
        configuration_file.getint('General','msg_server_port')))
    sock.sendall("HELLO"+"\n")
    # Receive data from the server and shut down
    received = sock.recv(1024)
    if received[:7] == "MSG_SRV":
        print "MSG_SRV ok !"
    else:
        print "Sorry Wrong response"
        sys.exit(-1)

    data = "REGISTER:E:%s:%s:%s\n"%(socket.gethostname(),
        configuration_file.get('Encrypt_server','encryp_server_port'),
        configuration_file.get('Encrypt_server','max_processes'))
    print data
    sock.sendall(data)
    received = sock.recv(1024)
    received = received.split(":")
    if received [0] == socket.gethostname() and received[1]=="REGISTED
        " and received[2]=="E":
        print "{} sucessefull registed on Message Server".format(
            socket.gethostname())
        my_key = received[3]
    elif received[1] == "ALREADY_REGISTED":
        print "Already Registed , moving on..."
    else:
        print "Sorry I cannot register in Message Server...\n Quitting
            !"
        sys.exit(-1)
finally:
    sock.close()
signal.signal(signal.SIGINT, signal_handler)
HOST, PORT = configuration_file.get('Encrypt_server','encryp_server_ip
    '), configuration_file.getint('Encrypt_server','encryp_server_port
    ')

# Create the server, binding to localhost on port 9999
```

```
    server = SocketServer.TCPServer((HOST, PORT), MyTCPHandler)
    print "Starting Encrpyt Server on {}:{}".format(HOST,PORT)
    # Activate the server; this will keep running until you
    # interrupt the program with Ctrl-C
    server.serve_forever()
    print "Exiting..."
```