Introduction to the Paper.

This article is an excerpt of a much larger graduation thesis. To point this out, the reference marks from the original text will be kept (the chapters, the paragraphs and the captions will not be renumbered). This Paper aims at illustrating briefly what IENA is, how it was born and how it is possible to integrate it into Nagios.

Chapter 3.

"If you have an apple and I have an apple and we exchange these apples then you and I will still each have one apple. But if you have an idea and I have an idea and we exchange these ideas, then each of us will have two ideas." (George Bernard Shaw)

In this chapter the concept of IENA will be introduced, and the project together with its possible future developments will be discussed. The purpose of this chapter will not lie in offering a IENA implementation nor in describing its technical function.

3.1 Outlook.

The enterprises that project and implement a good computer system aim more and more frequently at simplifying and quickening the data administration, the transactions such as the financial ones, and the internal and external business communications.
Anyway security of the information shared within the system might be questioned by the fact that some people enjoy creating worms, virus, troyan for different reasons, and, last but not least, they try to take possession of "sensible" data belonging to the enterprise itself. For that reason during the history of computer science some security systems such as AntiVirus, Firewall, IDS (Intrusion Detection System), NIDS (Network IDS), and many more have been developed.
Every System listed above has the main purpose of protecting the business network by putting into practice the strategy of a "closed policy": encrypting the communication, getting rid of non-essential services, and so on.
The intent of this project consists in revising the logic of the paradigm attack/defence in today's security systems by introducing a pattern whose basis is an opposite approach if compared to the principles of the so-called "closed policy" used so far.
In practice the main purpose is creating a system whose aims are increasing and improving the business network security and the safeguard of the information exchanged within the enterprise.

3.3 Concept analysis.

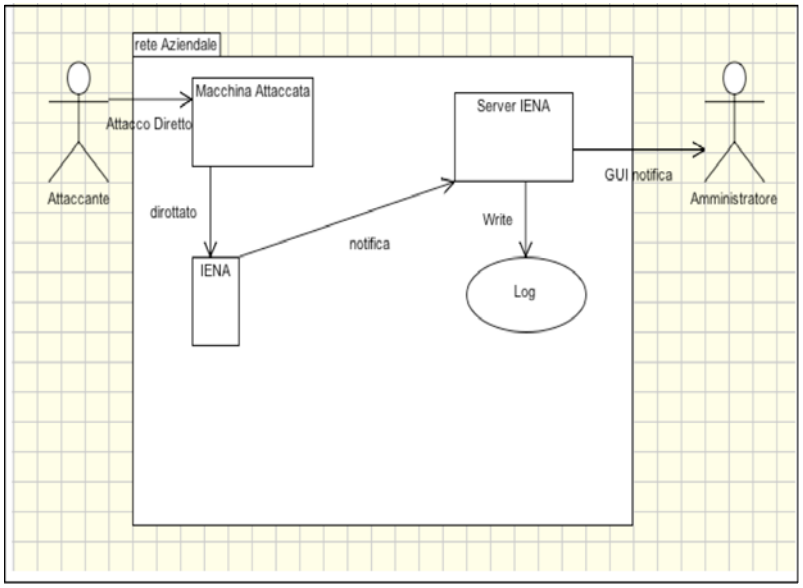
The fundamental objective is stopping and so identifying, the attacks even before they might bring about damage to the system; for that reason a good project has to operate at the source of the attack. Both IDS and IPS act during the attack, so sometimes they are ineffective. The instrument that will be designed must have the purpose of identifying any attack before it comes out. To make this sort of "magic prevention" possible, let's imagine we have to attack a system by taking on the attacker's point of view.
How is it possible to attack a system if we know nothing about it? The first step lies in coming into possession of the information; there are many techniques to get information: ripe registers, footprinting techniques, EDGAR research, searching on POC, MX record analysis; but now and for ever they will have to know how many and what services have been set up in the operative

machine. Thanks to this knowledge, we can state that a port mapping has a crucial importance for any attacker (social engineering permitting); why don't we make use of this knowledge?
Let's imagine we have just carried out a port scanning on a machine and we have intercepted the port 12345(a typical door of the famous troyan NetBus); for us that is a victory because we are sure that the machine has already been violated by NetBus, so we will ingenuously make a connection on that port to take possession of the system. Let's try to picture the attacker's expression when he has just realised that the port 12345 is not the NetBus door but it is a trap called IENA!
We have just discovered what IENA is: a trap into which any attacker will fall for only this reason: to attack a machine, knowing what services it provides is necessary but not sufficient.
Creating an instrument based on this scam to take the attackers in is our objective; that instrument must be designed with simple but effective technologies; it must be within everyone's reach because it will be easy to install and user-friendly; it will have to be as independent from the platform as possible and it will have a very high sensitivity level (identifying every single connection even if it might result as a permitted one, because the instrument sensitivity will be set expressly on every specific network).

3.3 Project.

Unlike an ids, IENA is not in search of "anomalous" requests of a specific service, but IENA itself creates false services on any machine.
Let's fix as a project prerogative the fact that a simple port-map for the internal business network is considered an Attack; setting IENE on every machine will make the attacker believe that he is dealing with a very vulnerable machine as regards different types of attack.
NOW and FOR EVER any attacker will have to make sure of which services are provided by the machine before putting into practice any type of attack. As soon as a request of a simple connection (such as a port-map) is sent in, IENE (false services) will spring as a trap sending messages of attempted (or effective) attack to the IENA server. IENA will catch the attacker's ip address and the right time of the attack (for a precise identification); moreover it will be able to create ban lists (with variable timing of one week) preventing the access to the network from that particular ip.
This way we are not trying to "close" the network, but we make use of the knowledge of the attacks to anticipate the attacker's move.
<u>This system will always be independent from the new vulnerabilities, cancelling the stressful race for the most up-to-date tool and for the system with the largest number of security paths.</u>
Preventing the attacker from carrying out a spoofing IP, with a correct hardening and a NIDS (such as arp watch) for the arp and icmp redirect attacks we are able to isolate the attacker and take action against him.

The UML diagram below lists the main characteristics of IENA.



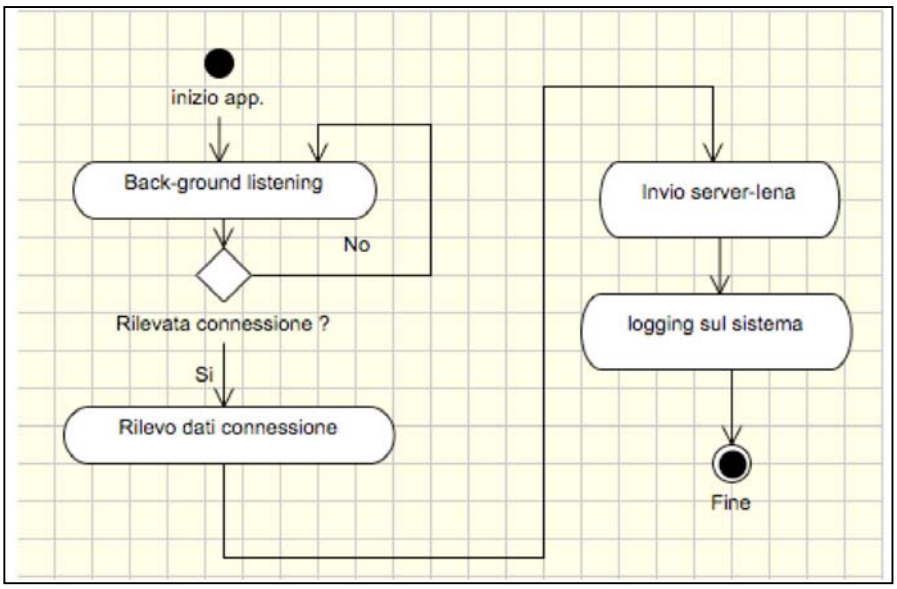
Picture 3.1: *Behavioural IENA UML*

In every machine one or more IENE will be installed; the installation of IENE can be manual or made through a script (platform dependent); every single IENA will hang on a particular port (at the user's discretion); as soon as IENA notices a connection on the "X" port, it will send a message to a server (IENA-server) which will deal with logging every access to the system log, including time and date of the presumed attack recording. Once the "capture" and the printing on the system log have been carried out, we can follow two different directions:

1) system administrator's warning (trough SMS or mail)
2) alteration in the net rules (IpTables modification and routing)

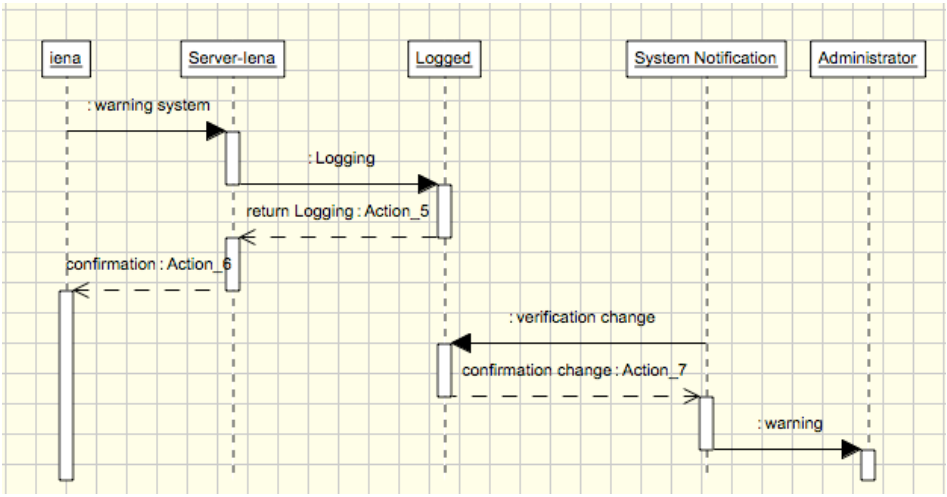
The system administrator's warning and/or the alteration in the system rules take place in a higher IENA logical level, so we could connect to the pre-existent "warning systems".

The Activity Diagram below sums up all the states of the IENA system.



Picture 3.2: *IENA Activity Diagram*

Every time that IENA notices a connection (or an attempt at connection) it intercepts all the respective data by sending them to the IENA-server. The communication between the IENA-server and (inside the net) and the single IENA (interface towards the external world) is carried out through a connected-oriented connection(TCP) by using the socket raw technology. When the IENA server has recorded the interception on the system log, another monitoring service will be operative; withdrawing this log cyclically, and the monitoring service will warn the administrator about what happened.
The following Sequence Diagram provides the details about the whole notification sequence.

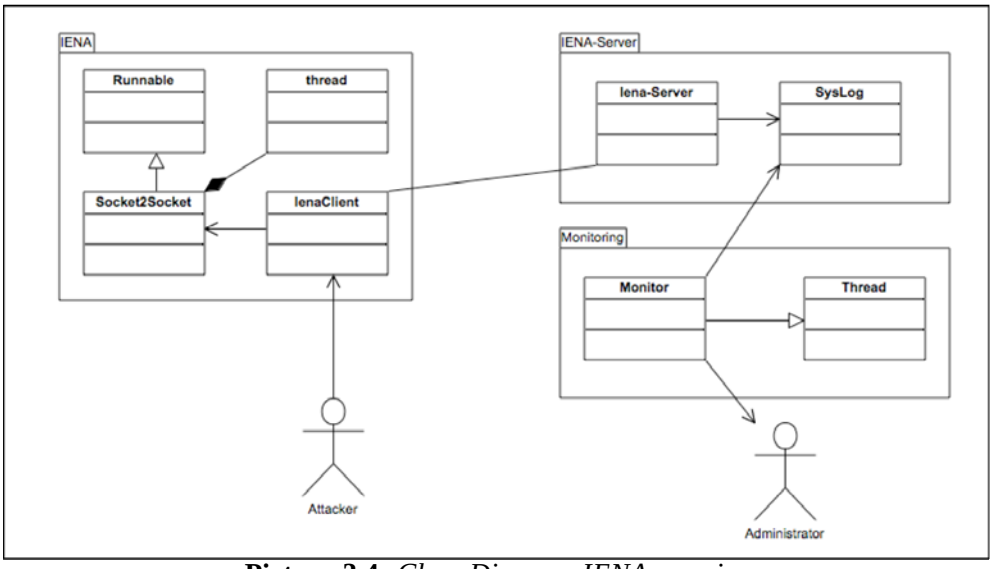


Picture 3.3: *Sequence diagram IENA notification*

From picture 3.3 we notice that there are two different control flows, the former (from the left to the right) comes out of the survey of a supposed IENA intrusion; the latter (from the right to the left) is repeated cyclically by the external System Notification. So the two processes are different; starting the former does not necessarily involve starting the latter. The stratification of this pattern comes out of the need of creating a system which is independent from the notification systems and the platform on which it has been installed. Afterwards an implementation written in Java (for IENA) and an implementation written in C-UNIX (for the IENA server) will be provided , since a “host” POSIX based system has been taken into account as a procedural choice.

3.4 Logical Architecture.

At the moment IENA logical architecture cannot be traced back to the typical Jacobson’s MVC model because at present it has not been designed to interact directly with the administrator: indeed it has the function of standing in the back-ground and warning through the system log. Below we report a ClassDiagram to explain the logical architecture in details.



Picture 3.4: *Class Diagram IENA warning*

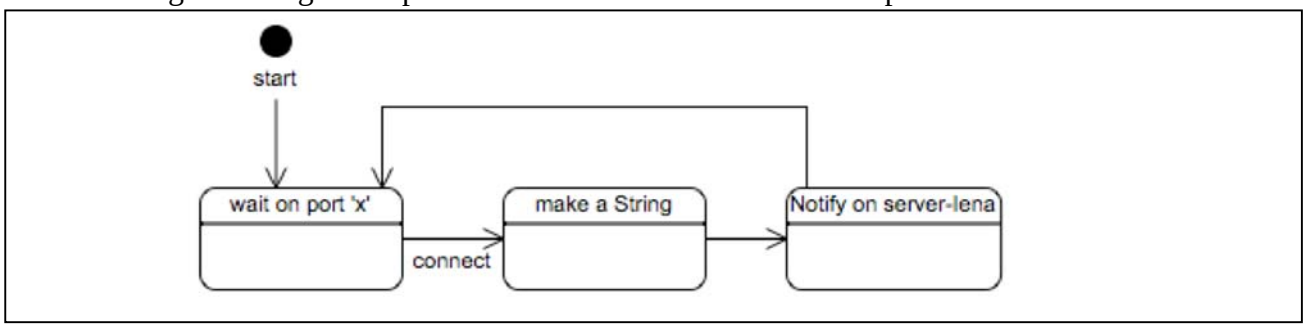
The three logical packages represent three different identities:

1) IENA: it represents the IENA-client, the class which has the concrete duty of intercepting the connections and notifying them to the IENA-server
2) IENA-server: this logical package represents the centralised logging method; it dialogues with "n" IENA-client and it is able to notify through system log
3) Monitoring: this logical package represents a logical entity (in this particular case it will be implemented by Nagios) which can notify to the net administrator what is happening on the net (in Nagios case, the warning can be carried out through e-mail or SMS).

At the moment of the connection (connection survey), the IENA-client class will evoke the Socket2Socket class which will distinguish the packages and shape a notification to be send to the IENA-server socket. These classes (implement Runnable) will have to stay in BackGround directly or indirectly, so they will extend general threads in "wait" state.

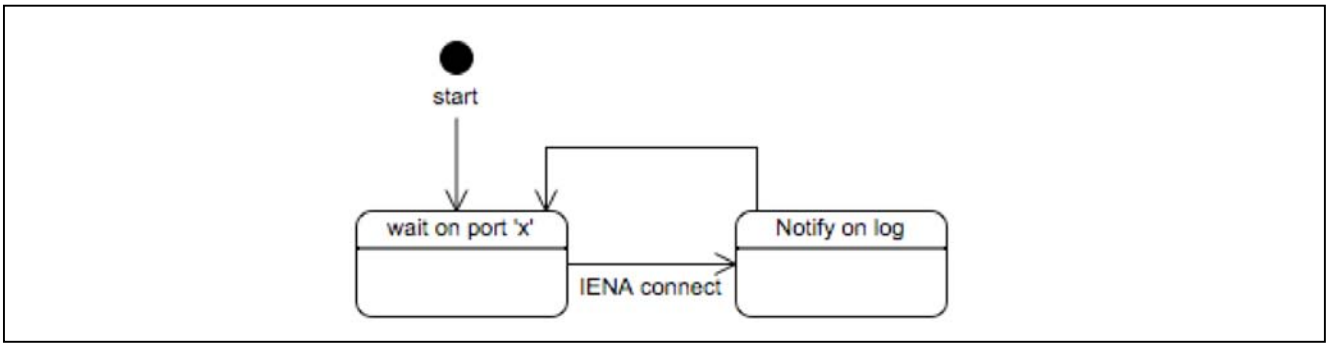
Once the connection has come (connect), there will be a shift in state, from the "wait" state to the "make String" state; afterwards, without any external interruption, the state will be changed into "Notify" state (notification) whose duty will be interacting with the IENA-server to communicate all the data referring to the connection.

The following state diagram explains how the state transactions take place.



Picture 3.5*: Client-IENA state diagram*

Once the notification has reached the IENA-server, it will change the state from "wait" to "Notify" to log the request on System-Log; the IENA-server will only have essentially these two states.



Picture 3.6: *Server-IENA state diagram*

The notification to the system administrator comes through a "known" system log, so there is no point in inserting a diagram state of the Notify system into the IENA project because it depends on the software used, not on the IENA project.

3.5 Why IENA is an effective model.

Let us stress the point according to which every attacker needs to know the services activated on a machine, and for doing that he might try different possible ways but in any case he will carry out a port-scan (it is possible that the attacker might use the social engineering techniques; in this case any technology would not be able to forestall it).
Regarding a port-scan implementation as permitted, some false services such as telnet, ssh or net-bios are created; this way, seeing these open services, ANY attacker will never hesitate to attack them.
The concrete concept can be compared with the typical HoneyPot concept. Unlike a HoneyPot, IENA does not aim at "entertaining" the intruder to take advantage (to learn new attack techniques), on the contrary it aims at being a "distributed" system to record and forestall possible attacks (we point out that HoneyPot is a system that works locally).
There are several types of HoneyPots (of low level such as BOF and Specter, or of high level such as HoneyNet), but in any case they are Host simulations that log locally what happens in specific connection conditions.
IENA is a light service, suitable for a distributed location (thanks to the remote logging); it is based on a very effective notification system such as Nagios and it has the duty of manipulating dynamically the net rules to "close the door" to the attacker.

3.6 Future developments.

This thesis project provides a IENA implementation that can be revised and readjusted for a remarkable development:

1) encrypting the communication between IENA and the IENA-server
2) the use of special socket-raw for the stealth scan detecting
3) the possibility of inserting a graphic interface to make the installation and the employment more user-friendly
4) the creation of particular rules to distinguish between the different types of attack
5) the implementation of a net based on IENA and Knock for the services that have been really activated

These are just some of the possible future developments of the IENA project.

Chapter 5.

"The value of an idea lies in the using of it" (Thomas Alva Edison)

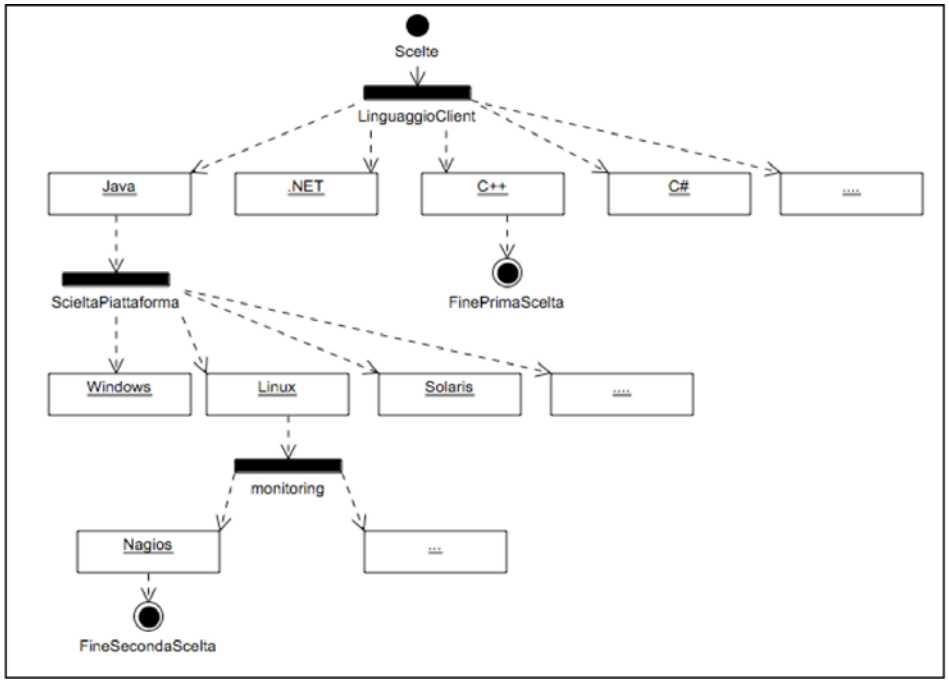
In this chapter we will deal with the problem of the concrete execution of IENA, one of the possible implementation will be introduced, expounding appropriately the decision tree.

Picture 5.0: *IENA logo*

5.1 The decision tree.

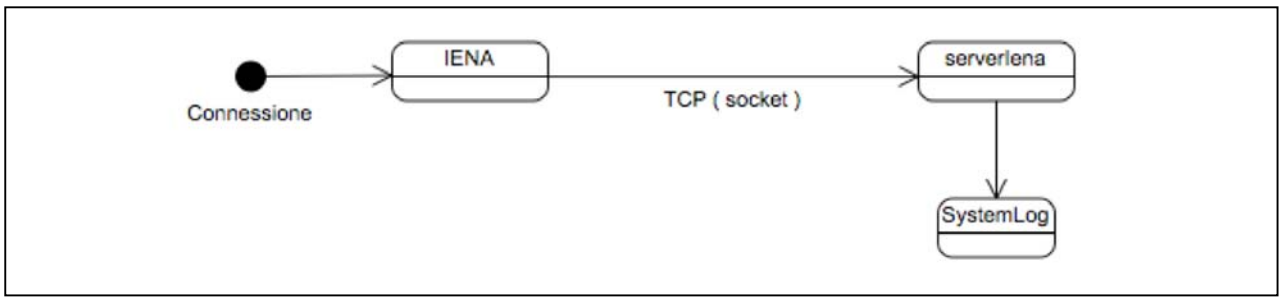
The project has been carried out thanks to Java and C-UNIX technology.

The choice of using an interpreted language such as Java has been brought about by the fact that the Java Virtual Machine is present almost on any platform, so writing a Client (IENA) with this technology assures a total reutilization of the code adapting it to every operating system. The use of Java has been considered more preferable than a Framework interpreted as .NET because the Java community is closer to the open-source world. Linux (fedora core) has been chosen as the IENA-server host system among the different available operating systems because it is free, reliable and very adaptable: it suits a business or an academic net of considerable dimensions. The language C-UNIX has been used to implement the IENA-server because it is remarkably fast and flexible; as it is not interpreted, it does not need any further tools (such as JVM) to be carried out on the host machine. After having taken into consideration several net monitoring systems in a Unix-based location, Nagios has been chosen; its easy installation procedure, the web-based dialogue system, the advisory system through sms or mail have strongly influenced the choice.
A first decision tree has been reported below.



Picture 5.1: *the decision tree*

The communication between IENA Client and IENA-server is carried out thanks to simple sockets TCP/IP; to the detriment of transport systems such as SSL, http, SOAP, this technology has been chosen because a large Framework is provided; according to the principle of the "code reutilization", several classes derived from previous projects have been used. This does not mean that in the next versions it will not be possible to change the communication protocol referring to well-known protocols such as SOAP or safe connections.



Picture 5.2: *communication between IENA and IENA-server*

IENA sensibility has been designed to be maximised because it is never too late to limit sensibility, on the contrary it could be difficult if the project was born with a lower sensibility. For this reason every IENA starts as soon as there is a connection on it.

To assure an easy "installation" (compilation and start) a script has been created: it meets the main requests of the Nagios standard; it has been written in Unix-shell technology and it has the duty of compiling both IENA-server and IENA-Client, and starting the server or the Client according to the transmitted entrance parameter; this script called "iena.script" must be inside the directorate in which there are the IENA-server and IENA-client files.

5.5 Start and IENA stand-alone check up.

Let us take into consideration the start script and compile the sources by writing:

```
-sh iena.script-compile
```

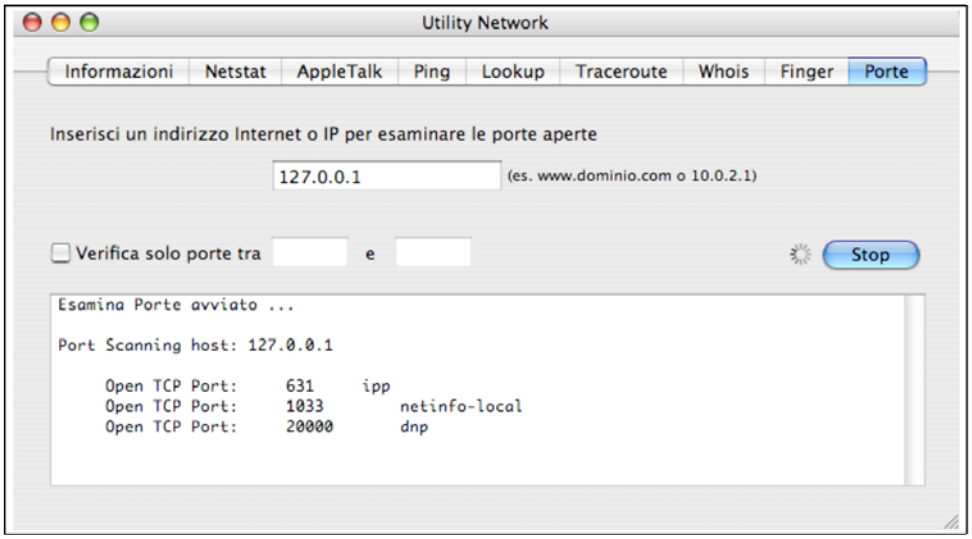
start the server and check up the effective functioning(stand-alone)

```
-sh iena.script start
```

Now let us check up the correct IENA-server start:

```
-ps –A | grep ienaServer
```

Let us check up the effective communication port opening through a simple port-scan; here is the result:



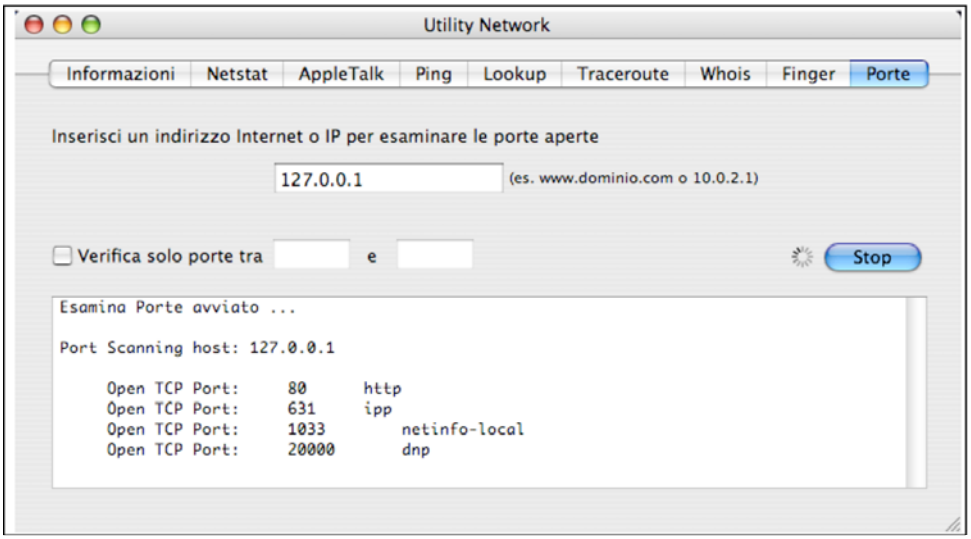
Picture 5.3*: port 20000 opening check up*

Q.E.D., the “port” 20000 (IENA-server default port) is open, so the logging service turns out to be operating on this machine. If we want to modify the logging server listening port, it is sufficient to edit the port number inside the start script (iena.script).
To globally check that the IENA service is functioning correctly, we have to start IENA (IENA Client) on the same machine and to test the double functioning by carrying out a port scanning locally (127.0.0.1); the result will have to be the following:

The port scanning result will also have to provide the port 80 (IENA default port) as an “open” port; in parallel, IENA will have to log the access attempt to the scanning port 80.

```
-sh iena.script startIENA
```

Let us check up the effective opening of the port 80 (IENA Client default port) by carrying out a port scanning locally, which will have to be considered as an attack and be monitored.
The scanning-port result has been reported below:



Picture 5.4: *IENA service check up*

As we can see in picture 5.4, both services are operating; carrying out a “cut” of /var/log/iena.log, we’ll obtain the system log with the notification of the occurred attack.

```
**** AttaccoAttacco da /127.0.0.1:53016 in 80  **** Date: Wed Jul 20 18:30:54 2005
 *****
**** AttaccoAttacco da /127.0.0.1:57522 in 80  **** Date: Wed Jul 20 18:32:34 2005
 *****
**** AttaccoAttacco da /127.0.0.1:59249 in 80  **** Date: Wed Jul 20 18:33:16 2005
 *****
**** Attacco  **** Date: Wed Jul 20 18:37:13 2005
 *****
PowerBook:/Users/marcoramilli/Documents/TESI_NAGIOS/src#
```

Picture 5.5: *log file*

It is interesting to check up the notification; through the tcpdump control it is possible to check up the effective interaction between the IENA-server and IENA Client; some logs referring to the communication during the “attack” have been reported below:

```
    L%.#l%.#
18:33:16.139877 IP (tos 0x0, ttl  64, id 21950, offset 0, flags [DF], length:
60, bad cksum 0 (->e6fb)!) localhost.59250 > localhost.dnp: S [bad tcp cksum
fe30 (->6e9c)!] 1511051608:1511051608(0) win 65535 <mss 16344,nop,wscale
0,nop,nop,timestamp 1814419235 0>
….E..<U.@.@.............rN Z..X…….0….?........

    L%.#....
18:33:16.139921 IP (tos 0x0, ttl  64, id 21951, offset 0, flags [DF], length:
60, bad cksum 0 (->e6fa)!) localhost.dnp > localhost.59250: S [bad tcp cksum
fe30 (->c948)!] 3370227991:3370227991(0) ack 1511051609 win 65535 <mss
16344,nop,wscale 0,nop,nop,timestamp 1814419235 1814419235>
….E..<U.@.@...........N .r….Z..Y…..0….?........
    L%.#l%.#
18:33:16.139930 IP (tos 0x0, ttl  64, id 21952, offset 0, flags [DF], length:
52, bad cksum 0 (->e701)!) localhost.59250 > localhost.dnp: . [bad tcp cksum
fe28 (->2f31)!] 1:1(0) ack 1 win 65535 <nop,nop,timestamp 1814419235 1814419235>
….E..4U.@.@.............rN Z..Y…….(…..
```

```
    l%.#l%.#
18:33:16.142961 IP (tos 0x0, ttl  64, id 21953, offset 0, flags [DF], length:
85, bad cksum 0 (->e6df)!) localhost.59250 > localhost.dnp: P [bad tcp cksum
fe49 (->cdf2)!] 1:34(33) ack 1 win 65535 <nop,nop,timestamp 1814419235
1814419235>
….E..UU.@.@.............rN Z..Y…….....I…..

    l%.#l%.#Attacco da /127.0.0.1:59249 in 80
18:33:16.143047 IP (tos 0x0, ttl  64, id 21954, offset 0, flags [DF], length:
52, bad cksum 0 (->e6ff)!) localhost.59250 > localhost.dnp: F [bad tcp cksum
fe28 (->2f0f)!] 34:34(0) ack 1 win 65535 <nop,nop,timestamp 1814419235
1814419235>
….E..4U.@.@.............rN Z..z…….....(…..
l%.#l%.#
18:33:16.143067 IP (tos 0x0, ttl  64, id 21955, offset 0, flags [DF], length:
52, bad cksum 0 (->e6fe)!) localhost.dnp > localhost.59250: . [bad tcp cksum
fe28 (->2f0f)!] 1:1(0) ack 35 win 65535 <nop,nop,timestamp 1814419235
1814419235>
….E..4U.@.@...........N .r….Z..{…..(…..

    l%.#l%.#
18:33:16.143128 IP (tos 0x0, ttl  64, id 21956, offset 0, flags [DF], length:
52, bad cksum 0 (->e6fd)!) localhost.http > localhost.59249: F [bad tcp cksum
fe28 (->3569)!] 1:1(0) ack 1 win 65535 <nop,nop,timestamp 1814419235 1814419235>
….E..4U.@.@.............P.q..;.0t.i.....(…..

    L%.#l%.#
18:33:16.143141 IP (tos 0x0, ttl  64, id 21957, offset 0, flags [DF], length:
52, bad cksum 0 (->e6fc)!) localhost.59249 > localhost.http: . [bad tcp cksum
fe28 (->3569)!] 1:1(0) ack 2 win 65535 <nop,nop,timestamp 1814419235 1814419235>
….E..4U.@.@.............q.P0t.i..;.....(…..

    L%.#l%.#
18:33:16.160587 IP (tos 0x0, ttl  64, id 21958, offset 0, flags [DF], length:
52, bad cksum 0 (->e6fb)!) localhost.59249 > localhost.http: F [bad tcp cksum
fe28 (->3568)!] 1:1(0) ack 2 win 65535 <nop,nop,timestamp 1814419235 1814419235>
….E..4U.@.@.............q.P0t.i..;.....(…..

    L%.#l%.#
18:33:16.160650 IP (tos 0x0, ttl  64, id 21959, offset 0, flags [DF], length:
52, bad cksum 0 (->e6fa)!) localhost.http > localhost.59249: . [bad tcp cksum
fe28 (->3569)!] 2:2(0) ack 2 win 65534 <nop,nop,timestamp 1814419235 1814419235>
….E..4U.@.@.............P.q..;.0t.j.....(…..
```

As we can notice from the logs, the communication with “Attack…” cord has been carried out only once during the port 80 check up made by the scanner.
The test through the port scanning, the log file and the communication between the two processes is tangible proof of the correct functioning of this first IENA implementation.

5.6 Installation and functioning with Nagios

IENA installation in Nagios is very easy. IENA is a stand-alone system which is independent from Nagios once it has started. It is sufficient to order Nagios to start IENA through the special script (iena.script); afterwards we will just have to let Nagios check up the changes in the IENA log file (/var/log/iena.log); once the two respective files Nagios and check_log have been shaped, the system can monitor the attacks on the hosts where IENA has been installed.

First of all, we have to insert the following lines into the configuration file “checkcommands.cfg”:

```
# 'check_iena' command definition
define command{
    command_name    check_iena
    command_line    $USER1$/check_log -F /var/log/iena.log -O /var/log/iena.old -q "Attacco"
    }
```

Picture 5.6: *modify checkcommands.cfg*

This way we “teach” Nagios what it has to do when the command “check_iena” is called; as can be noticed, we use a plug-in in present in the Nagios distribution: it compares the difference in log; in case the log changes with the passing of time, it means that an attack has been monitored by some IENA.
Now let us open the file “services.cfg”, configuration file of the Nagios services, and let us add the following lines:

```
# Service definition
define service{
    use             generic-service     ; Name of service template to use

    host_name           localhost
    service_description     IENA
    is_volatile         0
    check_period            24x7
    max_check_attempts
    normal_check_interval       1s
    retry_check_interval        1s
    contact_groups          linux-admins
    notification_interval       240
    notification_period     24x7
    notification_options        c,r
    check_command           check_iena
    }
```

Picture 5.7: *modify services.cfg*

This way we start the service “check_iena” (created above) every second, which means that every second Nagios will monitor if there are differences in the log files (iena.log). Once Nagios has been restarted, the system will be activated and ready for monitoring.

Author Marco Ramilli.
Translated by Michela Bertozzi.