

---

---

**ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA**  
**SEDE DI CESENA**  
**II FACOLTA' DI INGEGNERIA CON SEDE A CESENA**  
**CORSO DI LAUREA IN INGEGNERIA INFORMATICA**

TITOLO DELL'ELABORATO

**STUDIO E IMPLEMENTAZIONE DEI  
MODULI DEL PROGRAMMA DI  
MONITORAGGIO DI RETE NAGIOS**

Elaborato in

**LABORATORIO DI RETI DI TELECOMUNICAZIONI  
L-A**

Relatore  
**Cerroni Walter**

Presentata da:  
**Bianchini Stefano**

Sessione I  
Anno Accademico 2004/2005

---

---



*A Romina, Nonno Renato e Nonna Albertina*



# Indice

<b>1 Introduzione.....</b>	<b>1</b>
<b>2 Uno sguardo a Nagios.....</b>	<b>5</b>
2.1 Nagios.....	5
2.2 Interfaccia Web.....	6
2.3 Requisiti di sistema.....	7
2.3.1 Modifica della configurazione di Apache.....	9
2.4 Strumentazione utilizzata.....	11
2.5 Plugin.....	11
2.5.1 Installazione dei plugin.....	11
2.6 NRPE.....	12
2.6.1 Configurazione di NRPE.....	13
2.6.2 Avvio di NRPE.....	15
2.7 Configurazione di Nagios.....	15
2.7.1 Configurazione dei contatti.....	16
2.7.2 Configurazione dei comandi.....	17
2.7.3 Time Periods.....	19
2.7.4 Configurazione degli Host.....	19
2.7.5 Gruppi di Hosts.....	21
2.7.6 Configurazione dei servizi.....	22
2.8 Risoluzione dei problemi comuni.....	23
2.8.1 Verifica della validità della configurazione.....	23
2.8.2 Errore di scrittura dei file log.....	23
2.8.3 Errore numero 127.....	23
2.9 Consigli sulla sicurezza.....	24
<b>3 Panoramica sui plugin.....</b>	<b>27</b>
3.1 Check_breeze.....	27
3.2 Check_disk.....	27
3.3 Check_swap.....	29
3.4 Check_users.....	29
3.5 Check_icmp.....	30

3.6 Check_dhcp.....	31
3.7 Check_dns.....	32
3.8 Check_ssh.....	33
3.9 Check_ups.....	33
3.10 Check_disk_smb.....	35
3.11 Negate.....	36
3.12 Check_smtp.....	37
3.13 Check_file_age.....	38
3.14 Check_flexlm.....	39
3.15 Check_ftp.....	40
3.16 Check_mrtg.....	41
3.17 Check_pop.....	43
3.18 Check_imap.....	45
3.19 Check_nagios.....	46
3.20 Check_nrpe.....	47
3.21 Check_tcp.....	47
3.22 Check_udp.....	49
3.23 Check_time.....	49
3.24 Check_nt.....	50
3.25 Check_http.....	52
3.26 Check_log.....	54
3.27 Check_ping.....	55
<b>4 Linee guida per lo sviluppo di plugin.....</b>	<b>57</b>
4.1 Nozioni basilari.....	57
4.1.1 L'Output dei plugin.....	57
4.1.2 Scrivere solo un linea di testo come Output.....	58
4.1.3 Verbose output.....	58
4.1.4 Output guida a video.....	59
4.1.5 Generazione dello stato del servizio.....	59
4.1.6 Codici di ritorno (return codes).....	59
4.2 Comandi di sistema e gestione dei file ausiliari.....	60
4.2.1 Comandi di sistema esterni.....	60

4.2.2	Usa di spopen().....	61
4.2.3	File temporanei.....	61
4.2.4	Gestione dei links.....	62
4.2.5	Gestione dell'input.....	62
4.3	Opzioni dei plugin.....	62
4.3.1	Parametri.....	62
4.3.2	Le opzioni standard.....	63
4.4	Invio di nuovi plugin.....	63
4.5	Un po' di esempi pratici.....	64
4.6	Scheduling di un plugin.....	68
<b>5</b>	<b>Creazione di un plugin.....</b>	<b>71</b>
5.1	Requisiti.....	71
5.2	Analisi dei Requisiti.....	71
5.2.1	Ip Spoofing.....	71
5.2.2	Indirizzo Mac.....	72
5.2.3	Protocollo ARP.....	73
5.2.3.1	Scopo del protocollo ARP.....	73
5.2.3.2	Funzionamento.....	73
5.2.3.3	Sicurezza.....	74
5.3	Implementazione.....	74
5.3.2	Definizione degli stati di uscita.....	76
5.3.3	Funzione print_revision() e print_help().....	76
5.3.4	Parametri d'ingresso.....	77
5.3.5	Il cuore del programma.....	78
5.3.6	Un ultimo aggiustamento.....	80
5.4	Inserimento in Nagios.....	80
5.5	Verifica di funzionamento.....	81
<b>6</b>	<b>Un'interfaccia Web per Nagios.....</b>	<b>83</b>
6.1	Motivazioni.....	83
6.2	Implementazione.....	83
<b>7</b>	<b>Conclusioni.....</b>	<b>87</b>



# Capitolo 1: Introduzione

All'aumentare del grado di complessità della rete e del numero di macchine collegate, anche le attività più semplici diventano un problema.

Ad esempio, vorremo sicuramente sapere se i servizi sono tutti regolarmente in funzione o se la nostra banda comincia ad essere satura di traffico... ma non è certo una buona idea collegarsi una per una alle varie macchine e controllare, oppure mettersi davanti ad una console dove scorrono i dump degli analizzatori di rete!

Per questo, un'architettura di rete sicura deve prevedere anche dei **sistemi di monitoraggio centralizzato**, in grado di fornire rapidamente una sorta di "istantanea" dello stato della rete.

Grazie ad essi, è molto più semplice rilevare eventuali anomalie ed intraprendere le opportune azioni correttive, indagando nel contempo sulle cause dell'anomalia.

In questa tesi verrà preso in considerazione il programma più diffuso, Nagios, disponibile solo per piattaforma UNIX; in particolare si incentrerà sull'analisi, sulla configurazione e sullo sviluppo dei mattoni fondamentali di questo programma, i plugin.

Si deve tenere ben presente però che Nagios non è l'unico programma che fornisce un monitoraggio centralizzato: al di là delle soluzioni a pagamento (IBM Tivoli e OpenView della HP), esistono ottime alternative nel campo dell'open-source e del software libero in generale.

Ad esempio, un valido sostituto di Nagios può essere *Cacti*, strumento potente e flessibile, pur avendo un nucleo di funzionamento diverso basato su protocollo SNMP. L'interfaccia Web deve il suo funzionamento a script php al posto dei CGI di Nagios. Il sorgente è liberamente scaricabile dal sito <http://www.cacti.net>; nella figura sottostante è possibile vederlo all'opera.

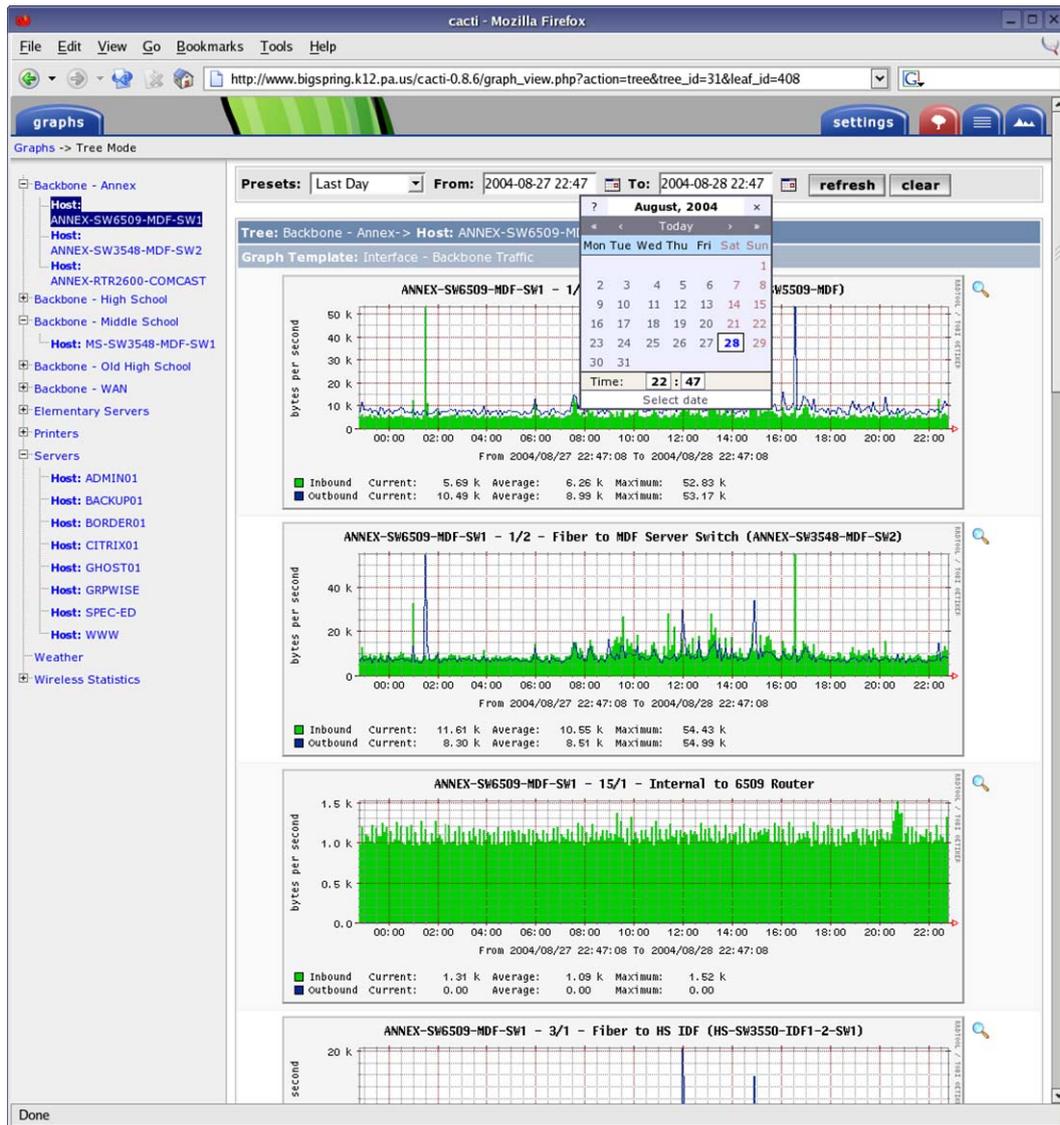


Figura 1. La schermata Web di Cacti

Cito anche, a titolo informativo, altri strumenti essenziali per il monitoraggio di rete:

- **Remote Filesystem Checker (RFC)**, set di script per automatizzare il controllo dei filesystem delle macchine remote da una console centralizzata, disponibile al sito <http://rfc.sourceforge.net>, dall'utilizzo alquanto ostico (completamente testuale da linea di comando, ma con possibilità di notifiche via email).

```
claudio@monster.roma2.infn.it: /home/claudio/RFC-NG-1.0.0 - Shell - Konsol
Sessione Modifica Visualizza Segnalibri Impostazioni Aiuto
#####
#                               USAGE:                               #
#                               #                                     #
# ssh-key-gen                    (create new ssh keys)             #
# ssh-key-del                    (delete the ssh keys)            #
# set-perm                       (set right permissions)         #
# node-add                       (add a node to RFC list)        #
# node-chk <node>                (check <node>)                  #
# node-update <node>             (update <node> - ask)           #
# node-force-update <node> <node> (update <node> - force)       #
# node-remove <node>            (remove <node> - ask)           #
# node-force-remove <node> <node> (remove <node> - force)       #
# mass-chk                       (check all nodes)               #
# mass-update                    (update all nodes)              #
# mass-remove                    (remove all nodes)              #
# list-nodes                     (list all RFC nodes)            #
# run                            (check all and send report)     #
# mail [<address>]               (send a report)                 #
# chk-config                     (check RFC config)              #
# version                        (show RFC version)               #
#                               #                                     #
#####
[claudio@monster RFC-NG-1.0.0]$
```

Figura 2. Una schermata dell'help di RFC

- **Multi Router Traffic Grapher (MRTG)**, statistiche di utilizzo della banda, scaricabile dal sito <http://people.ee.ethz.ch/~oetiker/webtools/mrtg> e disponibile anche per sistemi Win32.

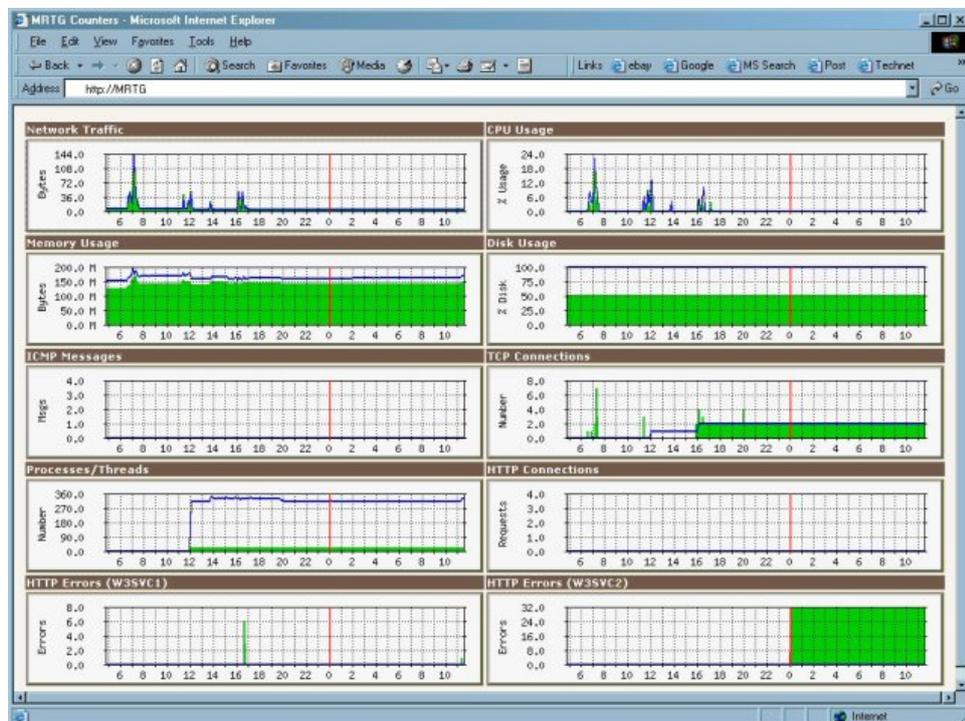


Figura 3. L'interfaccia Web di MRTG



## Capitolo 2: Uno sguardo a Nagios

### 2.1 Nagios

Nagios è un programma open-source per il monitoraggio delle reti locali[1]. Un sistema di monitoraggio controlla le risorse dei terminali (processi, spazio libero su disco) e la disponibilità dei servizi che offrono (Web server, server FTP, server DHCP). Nasce dalle ceneri di NetSaint, e il suo acronimo significa “*Nagios Ain't Gonna Insist On Sainthood*”, specificando quindi la diversità dal suo predecessore (non insisterà più sulla santità)[2]. Il tutto è corredato da un sistema di notifiche di malfunzionamenti, utilizzabile via posta elettronica o SMS.

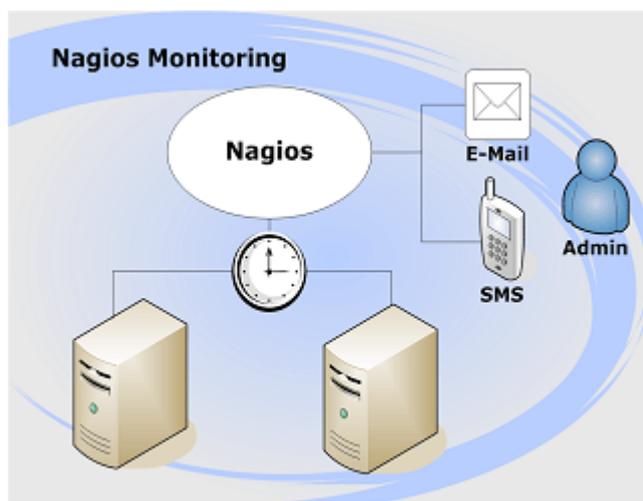


Figura 1. Schema del funzionamento di Nagios

I componenti principali di Nagios sono:

- **Monitoring daemon:** è un processo lato server che implementa le funzionalità di monitoraggio e di notifica degli allarmi;
- **Web interface:** è un'applicazione Web, installabile su un qualsiasi Web Server che supporti lo standard CGI (usualmente viene utilizzato il programma Apache), che consente di visualizzare lo stato degli oggetti monitorati anche da postazioni diverse dal server Nagios.

## 2.2 Interfaccia Web

L'interfaccia Web di Nagios mostra lo stato degli oggetti monitorati secondo diversi livelli di dettaglio.

Le viste disponibili principali sono:

- **Status Map**, che visualizza la tipologia e lo stato della rete;
- **Status Detail**, che mostra invece lo stato di ciascun servizio monitorato
- **Tactical Overview**, che sintetizza in un'unica schermata lo stato della rete e dei servizi;
- **Status Overview**; che mostra lo stato dei servizi raggruppati secondo un criterio stabilito (tipologia, ubicazione, ecc.);
- **Alert History**, che mostra lo storico dei problemi rilevati con possibilità di applicare filtri;
- **Trends**, che mostra l'andamento nel tempo dello stato di un servizio.

Nagios consente di rendere sicuro l'accesso all'interfaccia Web, definendo utenti con differenti permessi di visualizzazione sugli oggetti monitorati.

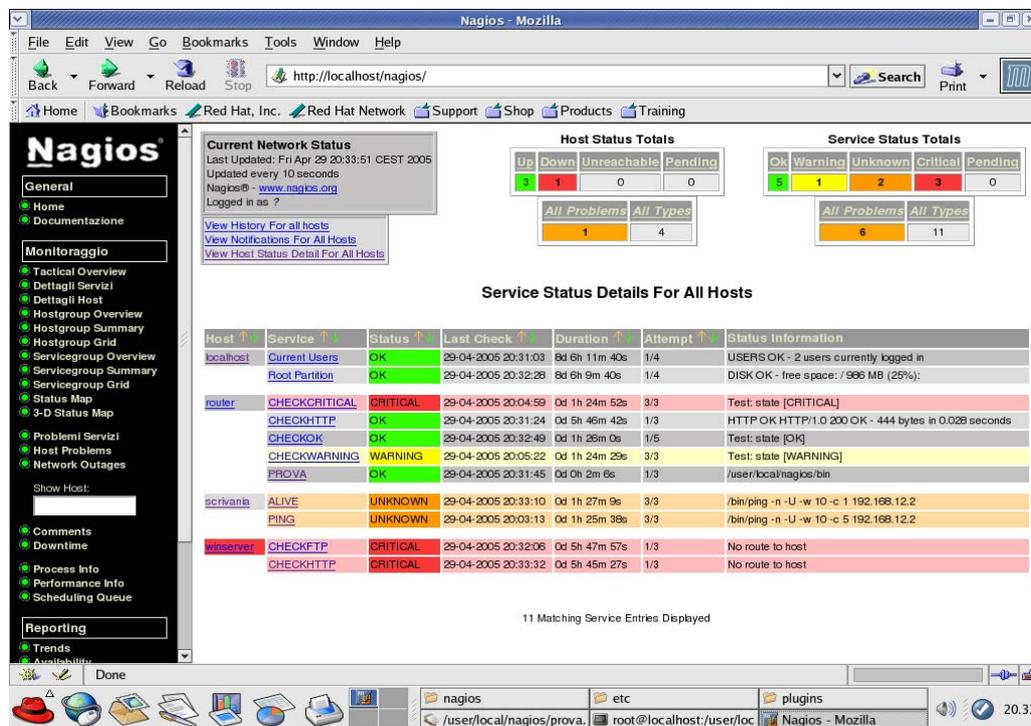


Figura 2 – Una schermata di Nagios

## 2.3 Requisiti di sistema

Il solo requisito principale è quello di utilizzare una macchina su cui giri GNU/Linux e nella quale sia installato un compilatore gcc. Opzionalmente, ma personalmente caldeggiato, l'uso delle CGI, delle librerie gd di Thomas Boutell e conseguentemente del web-server Apache.

Andiamo ad elencare i file necessari per installare Nagios, facilmente reperibili su internet, al fine di ottenere una configurazione base funzionante ed espandibile in funzione di qualunque esigenza.

- nagios
- nagios plugin
- gd library

### Compilazione delle librerie “gd”

Eeguire

```
# ./configure
# make check
# make install
```

### Installazione e compilazione di Nagios

Per prima cosa è necessario creare l'utente ed il gruppo *nagios*:

```
# adduser nagios
```

Spostiamoci nella directory in cui abbiamo salvato i sorgenti e decomprimiamo il file di nagios:

```
# tar -zxvf nagios-2.0b3.tar.gz
```

Spostiamoci nella directory appena creata:

```
# cd nagios-2.0b3
```

e lanciamo il file di configurazione:

```
# ./configure --prefix=/usr/local/nagios --with-  
cgiurl=/nagios/cgi-bin --with-htmurl=/nagios --with-  
nagios-user=nagios --with-nagios-group=nagios  
--with-gd-lib=/usr/local/lib  
--with-gd-include=/usr/local/include
```

Le ultime due opzioni non sono obbligatorie (il programma funziona anche senza librerie, ma le viste grafiche dinamiche non saranno possibili).

Compiliamo i binari

```
# make all
```

Installiamoli:

```
# make install
```

Installiamo lo script per l'avvio di Nagios (/etc/init.d/nagios)

```
# make install-init
```

È anche possibile installare una configurazione di base (molto utile all'inizio):

```
# make install-config
```

Una volta compiuto questi primi pochi passaggi se ci spostiamo nella directory di Nagios, appena creata, dovremmo ritrovare altre cinque sottodirectory.

```
# cd /usr/local/nagios
```

cioè:

/etc: contiene il file di configurazione di nagios

/var: directory dei log

/share: file HTML per l'interfaccia Web di Nagios

/bin: Nagios core (con il programma eseguibile)

/sbin: i file CGI necessari per l'interfaccia Web

## 2.3.1 Modifica della configurazione di Apache

Affinchè l'interfaccia Web funzioni, è necessario modificare il file di configurazione di Apache (/etc/httpd/conf/httpd.conf) e aggiungere le seguenti righe:

```
ScriptAlias /nagios/cgi-bin/
"/user/local/nagios/sbin/"
<Directory "/user/local/nagios/sbin">
    AllowOverride None
    Options ExecCGI
    Order allow,deny
    Allow from all
</Directory>

Alias /nagios/ "/user/local/nagios/share/"
<Directory "/user/local/nagios/share">
    Options None
    AllowOverride None
    Order allow,deny
    Allow from all
</Directory>
```

Si noti la direttiva AllowOverride None: in questo modo, e modificando opportunamente il file di configurazione di Nagios cgi.cfg (use\_authentication=0), è possibile accedere a Nagios via Web senza autenticazione http. Quindi è una configurazione pericolosa, e in caso di utilizzo reale assolutamente sconsigliata; per le prove del programma in rete locale invece, è utilissima e velocizza i tempi di accesso. Nel caso invece si voglia accedere a Nagios solo tramite autenticazione, si setta AllowOverride AuthConfig e si posiziona un file .htaccess nelle cartelle condivise del programma (per questo rimando alla documentazione ufficiale di Apache).

È possibile anche utilizzare la configurazione che segue per l'autenticazione, senza creare i file .htaccess nelle cartelle, e utilizzando una

maggior sicurezza (si noti che l'accesso è consentito solo ai computer di rete locale):

```
ScriptAlias /nagios/cgi-bin/ /usr/lib/nagios/cgi/
<Directory /usr/lib/nagios/cgi/>
Options ExecCGI
order deny,allow
deny from all
allow from 192.168.0.0/255.255.255.0 127.0.0.1
AuthName "Nagios Access"
AuthType Basic
AuthUserFile /etc/nagios/htpasswd.users
require valid-user
</Directory>

Alias /nagios/ /usr/share/nagios/
<Directory /usr/share/nagios/>
Options None
order deny,allow
deny from all
allow from 192.168.0.0/255.255.255.0 127.0.0.1
AuthName "Nagios Access"
AuthType Basic
AuthUserFile /etc/nagios/htpasswd.users
require valid-user
</Directory>
```

Sempre seguendo questa configurazione con autenticazione, dobbiamo creare nel file `htpasswd.users` l'utente speciale (solitamente *nagiosadmin*) che verrà utilizzato per la connessione all'interfaccia Web, dopodichè bisognerà far sì che l'utente apache sia in grado di leggere il file appena creato:

```
# htpasswd -c /etc/nagios/htpasswd.users nagiosadmin
# chown apache.apache /etc/nagios/htpasswd.users
```

## 2.4 Strumentazione utilizzata

Per le prove pratiche necessarie per la stesura di questa tesi sono stati utilizzati:

- Un host Fedora Core 2 con Nagios installato (192.168.12.240)
- Un host Fedora Core 3 con installato il demone NRPE (192.168.12.130)
- Un host Fedora Core 2 (192.168.12.10)
- Un host Windows Xp (192.168.12.2)
- Un host Windows 2003 Server con Web Server Apache e Server Ftp (192.168.12.3)
- Un router/access point/switch della U.S. Robotics con DHCP server (192.168.12.1)

## 2.5 Plugin

Una caratteristica peculiare di Nagios, che lo rende particolarmente versatile, è la possibilità di estendere i controlli previsti, integrando qualsiasi controllo esterno, mediante scrittura di plugin, ovvero programmi, sotto forma di semplici script o eseguibili, che Nagios è in grado di invocare per controllare lo stato di una risorsa o di un servizio.

Nagios fornisce numerosi plugin pronti all'uso, che consentono di controllare i servizi di rete più comuni tra cui: HTTP, SMTP, POP, FTP, SMB, DNS, DHCP, LDAP, Oracle.

La comunità degli utenti di Nagios ha sviluppato molti plugin che possono essere liberamente reperiti all'indirizzo <http://www.nagiosexchange.org>.

### 2.5.1 Installazione dei plugin

Ricordiamoci che Nagios diventa un programma molto potente grazie ai plugin, senza di essi sarebbe praticamente inutilizzabile.

Spostiamo nella directory dei sorgenti e decomprimiamo:

```
# tar zxvf nagios-plugins-1.3.1.tar.gz
```

Configuriamoli ed installiamoli:

```
# ./configure --prefix=/usr/local/nagios
# make
# make install
```

## 2.6 NRPE

Nagios consente anche l'esecuzione remota di plugin per monitorare i parametri critici di sistema come l'utilizzo della CPU e della memoria, lo spazio libero su disco, ecc. Questi plugin possono essere eseguiti dal server Nagios via SSH oppure possono utilizzare un demone NRPE da installare sul computer gestito.

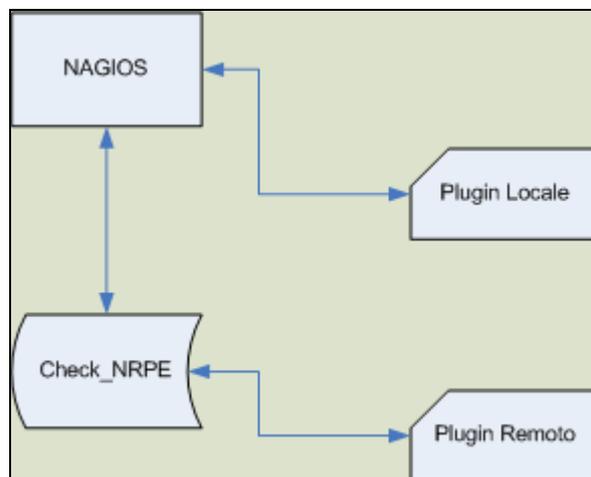


Figura 3. Funzionamento di NRPE

Il pacchetto NRPE è liberamente scaricabile (nella sezione Addons) dal sito [www.nagiosexchange.com](http://www.nagiosexchange.com), si decompone in una cartella qualsiasi dopodichè si lancia lo script di configurazione:

```
./configure
```

E lo si compila con il comando:

```
make all
```

La compilazione darà come risultato un file chiamato `check_nrpe`, che sarà il plugin che andrà posizionato nella directory `libexec` del computer sul quale installato Nagios; avremo ottenuto anche il file eseguibile per il demone NRPE (file `nrpe`), entrambi nella sottodirectory `src`.

Affinché si possa avviare il demone, bisogna copiare il file di configurazione `nrpe.cfg` nella directory `/etc` del sistema, e l'eseguibile `nrpe` nella directory `/bin` del sistema; non ha bisogno di nient'altro, solo di questi due file e il plugin installato sul server Nagios.

Esiste anche una versione del demone NRPE per Windows, ancora sperimentale, scaricabile all'indirizzo <http://www.miwi-dv.com/nrpent/>.

## 2.6.1 Configurazione di NRPE

La configurazione del demone NRPE si trova nel file `nrpe.cfg`. È importante e fondamentale impostare i plugin che devono essere eseguiti in remoto, che devono essere presenti sulla macchina in una directory a scelta.

Bisogna specificare gli host che possono comunicare con il server NRPE tramite la direttiva `allowed_hosts`. Poi si inserisce l'utente e il gruppo con il quale verrà eseguito il demone (`nrpe_user` e `nrpe_group`).

```
# ALLOWED HOST ADDRESSES
allowed_hosts=192.168.12.240
# NRPE USER
nrpe_user=nagios
# NRPE GROUP
nrpe_group=nagios
```

Per richiamare un plugin da remoto tramite `check_nrpe`, il demone deve avere nella configurazione i percorsi e i parametri dei comandi. L'esempio sotto riportato definisce il comando `check_disk`.

```
# COMMAND DEFINITIONS
# Command definitions that this daemon will run.
#Definitions are in the following format:
#
# command[<command_name>]=<command_line>

command[check_disk]=/bin/check_disk -w 20 -c 10 -p
```

```
/dev/hda1 -c 10 -p /dev/hdb1
```

Naturalmente è necessario dover configurare anche Nagios per poter inviare i comandi giusti al processo NRPE presente sull'host remoto [3].

Innanzitutto è necessario aggiungere una definizione di comando nel file di configurazione "checkcommands.cfg":

```
# Definizione per nrpe
define command{
command_name check_nrpe
command_line /usr/local/nagios/libexec/check_nrpe -H
$HOSTADDRESS$ -c
$ARG1$
}
```

Quindi Nagios creerà un comando che comprende l'indirizzo a cui chiedere il controllo ed il comando che si sarà definito nel file di configurazione di NRPE.

Infine è importantissimo configurare nel modo adeguato la definizione di servizio presente nel file services.cfg.

Un esempio pratico potrebbe essere:

```
define service{
use generic-service
host_name linux2
service_description DISK_NRPE
is_volatile 0
check_period 24x7
max_check_attempts 3
normal_check_interval 5
retry_check_interval 1
contact_groups stefano-group
notification_interval 120
notification_period 24x7
notification_options c,r
check_command check_nrpe!check_disk
}
```

## 2.6.2 Avvio di NRPE

Per avviare il demone NRPE utilizzare il comando:

```
nrpe -c <configuration file> -d
```

Dove <configuration file> è il percorso del file di configurazione (ad esempio, se si sono seguiti i consigli del paragrafo precedente, /etc/nrpe.cfg) e il parametro -d sta ad indicarne l'avvio in modalità demone stand-alone.

## 2.7 Configurazione di Nagios

Al contrario di quanto si possa pensare, la configurazione di Nagios, almeno nelle sue parti basilari, segue una logica chiara e semplice. Per far funzionare Nagios con qualche plugin di prova, bastano solo pochi minuti e una buona dose di pazienza.

Iniziamo dal file di configurazione centrale: nagios.cfg. In questo file saranno definiti le opzioni principali del programma (rimando alla documentazione ufficiale per spiegazioni dettagliate) e i richiami ai file esterni. I file esterni hanno nomi esplicativi per la loro funzione:

- **hosts.cfg** definisce gli host da monitorare
- **check\_commands.cfg** definisce i comandi da utilizzare nei servizi
- **services.cfg** definisce i servizi per ogni host
- **contacts.cfg** definisce i contatti necessari per le notifiche
- **hostgroups.cfg** definisce gruppi di host
- **contactgroups.cfg** definisce gruppi di contatti per le notifiche
- **timeperiods.cfg** definisce i periodi di tempo (ad esempio, il periodo 24x7 indica notifiche 24 ore al giorno, 7 giorni alla settimana)

```
cfg_file=/user/local/nagios/etc/services.cfg
cfg_file=/user/local/nagios/etc/timeperiods.cfg
cfg_file=/user/local/nagios/etc/hostgroups.cfg
cfg_file=/user/local/nagios/etc/host.cfg
cfg_file=/user/local/nagios/etc/checkcommands.cfg
cfg_file=/user/local/nagios/etc/contactgroups.cfg
```

```
cfg_file=/user/local/nagios/etc/contacts.cfg
```

In realtà la divisione in diversi file è solo concettuale: si può benissimo definire tutto in un unico file.

Nagios infatti, nel momento di avviarsi, crea un file chiamato `object.cache` nella directory `var` nel quale unisce tutti i file di configurazione in uno unico.

Per specificare dove Nagios troverà i plugin, inseriamo una ulteriore riga:

```
Resource_file=/user/local/nagios/etc/resource.cfg
```

Nel file `resource.cfg` andremo a definire i percorsi delle risorse (in questo caso il percorso dei plugin da eseguire).

```
#####  
# RESOURCE.CFG - Sample Resource File for Nagios  
#####  
# Sets $USER1$ to be the path to the plugins  
$USER1$=/usr/local/nagios/libexec
```

## 2.7.1 Configurazione dei contatti

La definizione dei contatti si basa su due fasi: per primo si descrive un contatto, inteso come persona fisica identificata da un nome e da, ad esempio, l'indirizzo mail sul quale inviare le notifiche. Si può anche definire la modalità di notifica.

```
# 'stefano' contact definition  
define contact{  
    contact_name           stefano  
    alias                  Nagios Admin  
    service_notification_period 24x7  
    host_notification_period 24x7  
    service_notification_options w,u,c,r  
    host_notification_options d,u,r  
    service_notification_commands notify-by-email  
    host_notification_commands host-notify-by-email
```

```
email
stefano-admin@localhost.localdomain
}
```

Successivamente va descritto un contact-groups, ovvero un gruppo cui i contatti definiti precedentemente faranno parte. In questo modo le notifiche, se specificato, verranno inviate al gruppo (inteso come insieme di contatti).

```
# 'stefano-group' contact group definition
define contactgroup{
    contactgroup_name    Stefano-group
    alias                 Gruppo Amministratore
    members               stefano
}
```

### 2.7.2 Configurazione dei comandi

La definizione di un comando esige un nome (che verrà richiamato dalla definizione successiva di un servizio, `command_name`) e dalla linea di comando del plugin utilizzato (ricordiamo che la variabile `$USER1$` è descritta in `resources.cfg`) con le opzioni necessarie. I parametri passati avranno una numerazione incrementale (`$ARG1,$ARG2$,...`) mentre la variabile che indica l'indirizzo Ip sul quale effettuare il controllo è fissa (chiamata `$HOSTADDRESS$`). Una buona dose di prove chiarirà ogni dubbio.

```
# 'check_tcp' command definition
define command{
    command_name    check_tcp
    command_line    $USER1$/check_tcp -H $HOSTADDRESS$
                  -p $ARG1$
}
```

Inseriamo, sempre nel file `check_commands.cfg`, le direttive relative alle notifiche via mail (nell'esempio citato una procedura per quelle relative ai servizi e una per problemi riguardanti host).

```
define command{
command_name    notify-by-email
command_line    /usr/bin/printf "%b" "*** Nagios  **\n\n
Notification Type: $NOTIFICATIONTYPE$\n\n
Service: $SERVICEDESC$\n
Host: $HOSTALIAS$\n
Address: $HOSTADDRESS$\n
State: $SERVICESTATE$\n\n
Date/Time: $LONGDATETIME$\n\n
Additional Info:\n\n$OUTPUT$" | /bin/mail -s "***
$NOTIFICATIONTYPE$ alert - $HOSTALIAS/$SERVICEDESC$
is $SERVICESTATE$ *" $CONTACTEMAIL$
}
```

Questo comando invia una mail di notifica ad un contatto specificando:

- la descrizione dell'host preso in considerazione (alias);
- l'indirizzo Ip;
- il nome del servizio;
- lo stato del servizio;
- informazioni aggiuntive come data,ora e output del plugin.

```
define command{
command_name    host-notify-by-email
command_line    /usr/bin/printf "%b" "*** Nagios  **\n\n
Notification Type: $NOTIFICATIONTYPE$\n
Host: $HOSTNAME$\n
State: $HOSTSTATE$\n
Address: $HOSTADDRESS$\n
Date/Time: $LONGDATETIME$\n
Info: $OUTPUT$\n\n" | /bin/mail -s "Host $HOSTSTATE$
alert for $HOSTNAME$!" $CONTACTEMAIL$
}
```

### 2.7.3 Configurazione dei “Time Periods”

Un “time period” è una descrizione di “quando” sia possibile inviare notifiche. Ad esempio, posso specificare che le notifiche debbano essere inviate solo durante la notte, o solo in determinati giorni [1].

Ogni periodo ha un nome (ad esempio 24x7) ,una breve descrizione (alias) e un elenco dei giorni e degli orari ammessi.

```
# This defines a timeperiod where all times are valid
# for checks, notifications, etc. The classic "24x7"
# support nightmare. :-)

define timeperiod{
    timeperiod_name 24x7
    alias            24 Hours A Day, 7 Days A Week
    sunday          00:00-24:00
    monday          00:00-24:00
    tuesday         00:00-24:00
    wednesday       00:00-24:00
    thursday        00:00-24:00
    friday          00:00-24:00
    saturday        00:00-24:00
}
```

### 2.7.4 Configurazione degli host

Definiamo un template comune a tutti gli host, il che significa che tutti quelli che utilizzeranno questa definizione avranno le stesse caratteristiche basilari:

```
#Generic Host Definition Template
define host{
name generic-host
event_handler_enabled 1
flap_detection_enabled 1
```

```
process_perf_data 1
retain_status_information 1
retain_nonstatus_information 1
register 0
}
```

Per ogni computer in rete da monitorare dovremo inserire una definizione :

```
#WinServer Host Definition
define host {
use generic-host
host_name winserver
alias Computer Windows Server
address 192.168.12.3
check_command check_tcp!80
max_check_attempts 10
notification_interval 480
notification_period 24x7
notification_options d,u,r
contact_groups stefano-group
}
```

In ogni definizione è necessario configurare i seguenti parametri: [2]

- il template da usare (`use`)
- un nome con il quale sarà identificato all'interno di tutti i file di configurazione di Nagios (`host name`)
- un alias che descrive brevemente l'host (`alias`)
- un comando che Nagios utilizzerà per determinare lo stato dell'host (`check_command`)
- il numero massimo di tentativi consecutivi da effettuare prima della notifica di irraggiungibilità di un host (`max_check_attempts`)
- l'intervallo di tempo da aspettare, in rimuti, prima di inviare una notifica (`notification_interval`)

- quando è possibile inviare notifiche, secondo il file di configurazione `timeperiods.cfg` (`notification_options`)
- Tipi di notifiche

I tipi di notifiche sono definiti come: [3]

1. **d** Host in stato “down” (non risponde)
2. **u** Host in stato di “unreacheable” (non può essere raggiunto)
3. **r** Host in stato di “recovery” (la situazione ritorna alla normalità dopo che l'host è stato o “down o “unreachable”)

## 2.7.5 Gruppi di hosts

Nagios mette a disposizione dell'utente anche la possibilità di raggruppare più host insieme e di vederne una visuale completa tramite tre viste, Hostgroup Overview, Hostgroup Summary e Hostgroup Grid. La definizione di un gruppo e dei propri membri necessita di un nome (`hostgroup_name`), di una breve descrizione (`alias`), e dell'elenco degli hosts membri separati da una virgola (`members`).

```
# 'Laboratorio via venezia 1 piano' host group def.
define hostgroup{
    hostgroup_name labviaven
    alias      Laboratorio via Venezia primo piano
    members   localhost,router,winservet
}
```

Laboratorio Via Venezia Primo Piano (labviaven)

Host	Status	Services	Actions
<a href="#">localhost</a>	UP	2 OK	
<a href="#">router</a>	UP	1 OK	
<a href="#">winservet</a>	DOWN	2 CRITICAL	

Figura 4. Una vista del gruppo appena creato

## 2.7.6 Configurazione dei servizi

Definiamo un template comune a tutti i servizi, in cui specificheremo, ad esempio, di abilitare la notifica in caso di problemi (`notifications_enabled`).

```
# Generic service definition template
define service{
    name                generic-service
    active_checks_enabled    1
    passive_checks_enabled  1
    parallelize_check       1
    obsess_over_service     1
    check_freshness         0
    notifications_enabled   1
    event_handler_enabled   1
    flap_detection_enabled  1
    process_perf_data       1
    retain_status_information 1
    retain_nonstatus_information 1

    register            0
}
```

Inseriamo una definizione per ogni servizio da utilizzare; dovrà contenere una descrizione, l'host verso il quale effettuare il servizio, il contact-group per le notifiche e, non per ultimo, il comando definito in `check_commands` da usare. I parametri vengono passati con la sintassi:

```
nome_plugin!argomento1!argomento2!...!argomentoN
```

```
define service{
    use                generic-service
    host_name          winserver
    service_description  TCP80
    is_volatile        0
    check_period       24x7
    max_check_attempts  3
    normal_check_interval  3
    retry_check_interval  1
    contact_groups     stefano-group
    notification_interval 20
}
```

```
notification_period      24x7
notification_options     c,r
check_command            check_tcp!80
}
```

Nell'esempio di configurazione esposto si vuole controllare, per la macchina winserver, se è possibile generare una connessione TCP attraverso la porta 80. Nel caso in cui non si abbia un riscontro immediato, Nagios eseguirà il controllo del servizio per tre volte (`max_check_attempt`) ad intervalli di un minuto (`retry_check_interval`); se l'esito continua ad essere negativo, verranno inviate notifiche.

Nel caso di successo del controllo, Nagios è configurato per ricontrollare il servizio ogni tre minuti (`normal_check_interval`).

## 2.8 Risoluzione dei problemi comuni

### 2.8.1 Verifica della validità della configurazione

Nel momento in cui si avvia Nagios per la prima volta, è buona norma verificarne la configurazione con il comando:

```
/usr/local/nagios/bin/nagios
-v /usr/local/nagios/etc/nagios.cfg
```

In questo modo è possibile controllare eventuali errori e risolverli.

### 2.8.2 Errore di scrittura dei file log

Fondamentale è dare alla directory `var` i permessi giusti e l'utente giusto: bisogna permettere all'utente fittizio che avvia Nagios la possibilità di scrivere i vari file log e il file `object.cache`, fondamentali per un corretto funzionamento.

### 2.8.3 Errore numero 127

Nel caso capitasse un errore del tipo “Return code of 127 is out of bounds - plugin may be missing” significa che Nagios non è riuscito ad eseguire il plugin. Le cause possono essere:

1. nella configurazione `check_commands.cfg` è stato specificato un nome sbagliato del comando;
2. nella configurazione `resorces.cfg` è stato specificato un percorso errato
3. il plugin non si trova nella cartella `libexec`.

## 2.9 Consigli sulla sicurezza

L'autenticazione basata su HTTP di Nagios è facilmente oltrepassabile. Il mio consiglio è quindi di abilitare sul server apache una connessione sicura https per l'accesso all'interfaccia web di Nagios, e se necessario inglobare il tutto in una rete privata virtuale per una crittografia superiore.

I motivi sono alquanto semplici: una volta aggirata l'autenticazione, con le impostazioni di default è possibile visualizzare in remoto la configurazione dei servizi di Nagios con i relativi parametri:

Service						
Host	Description	Max. Check Attempts	Normal Check Interval	Retry Check Interval	Check Command	Check Period
<a href="#">localhost</a>	Current Users	4	0h 5m 0s	0h 1m 0s	<a href="#">check_local_users!20!50</a>	<a href="#">24x7</a>
<a href="#">localhost</a>	Root Partition	4	0h 5m 0s	0h 1m 0s	<a href="#">check_local_disk!20%!10%!/</a>	<a href="#">24x7</a>
<a href="#">router</a>	CHECKHTTP	3	0h 5m 0s	0h 1m 0s	<a href="#">check_http</a>	<a href="#">24x7</a>
<a href="#">scrivania</a>	Check_IpSpoofing	3	0h 5m 0s	0h 1m 0s	<a href="#">check_ipspoofing!00:11:2F:B3:F6:A2</a>	<a href="#">24x7</a>
<a href="#">scrivania</a>	PING	3	0h 5m 0s	0h 1m 0s	<a href="#">check_ping!100.0,20%!500.0,60%</a>	<a href="#">24x7</a>

Figura 5. i parametri dei servizi letti da un utente malintenzionato

Un'altra soluzione consiste nel cambiare la configurazione del file `cgi.cfg` e limitando i permessi degli utenti remoti. In questo modo non sarà possibile per un utente, non sia fisicamente al computer sul quale è presente il server Nagios, vedere i parametri dei servizi, oppure riavviare o fermare il programma e così via.

Una ulteriore modalità per rendere più sicuro Nagios verso gli utenti esterni alla rete locale è quella di affiancare all'interfaccia web nativa una creata appositamente con funzionalità limitate. Questa soluzione verrà analizzata ed implementata nel capitolo 6.



# Capitolo 3: Panoramica sui plugin di Nagios

## Nagios

### 3.1 Check\_breeze

**Comportamento:** riporta la potenza del segnale radio di un access point wireless della Breezecom tramite il protocollo *SNMP*.

**Uso:**

```
check_breeze -H <host> [-C <community>] -w <warn> -c <crit>
```

**Parametri:**

- H <host> indirizzo ip dell'access point
- C <community> SNMPv1 community
- w <warn> valore percentuale della potenza del segnale al di sotto del quale il plugin genera un errore di tipo *WARNING*
- c <crit> valore percentuale della potenza del segnale al di sotto del quale il plugin genera un errore di tipo *CRITICAL*

**Esempio d'uso:**

```
check_breeze -H 192.168.12.1 -w 25% -c 10%
```

Controlla il segnale al 25% e al 10% per l'access point 192.168.12.1

### 3.2 Check\_disk

**Comportamento:** controlla l'ammontare dello spazio disco usato su un qualsiasi file system montato, generando un errore se il valore è al di sotto di una

certa percentuale definita (come parametro). Tra i file system controllati c'è anche la memoria ram (/dev/shm).

### Uso:

```
check_disk -w <wlim>% -c <clim>% [-p <path>] [-l] [-M]
[-e] [-u <units>]
```

### Parametri:

-w <wlim>% percentuale di spazio libero al di sotto della quale viene generato un errore di tipo *WARNING*

-w <wlim> numero di unità di spazio libero al di sotto della quale viene generato un errore di tipo *WARNING*

-c <clim>% percentuale di spazio libero al di sotto della quale viene generato un errore di tipo *CRITICAL*

-c <clim> numero di unità di spazio libero al di sotto della quale viene generato un errore di tipo *CRITICAL*

-p <path> percorso della cartella o partizione da controllare

-l controlla solo i file system locali e non quelli remoti

-M visualizza anche i mount point

-e visualizza solo i device/mountpoint con errori

-u <units> specifica le unità da utilizzare; di default è MB, ma si può scegliere anche kB, GB, TB.

### Esempio d'uso:

```
check_disk -w 10% -c 5% -p /tmp -p /var -w 100000 -c
50000 -p /
```

controlla le cartelle /tmp e /var al 10 e 5 % e la directory radice / a 100 MB, 50 MB.

### 3.3 Check\_swap

**Comportamento:** controlla lo spazio libero sulla/e partizione/i swap presenti sul computer locale; genera un errore se il valore è minore di una percentuale (o di un intero) definita come parametro.

**Uso:**

```
Check_swap -w <warn> -c <crit> [-a]
```

**Parametri:**

-w <warn>% percentuale di spazio libero al di sotto della quale il plugin genera un errore di tipo *WARNING*

-w <warn> numero intero di bytes di spazio libero al di sotto del quale il plugin genera un errore di tipo *WARNING*

-c <crit>% percentuale di spazio libero al di sotto della quale il plugin genera un errore di tipo *WARNING*

-c <crit> numero intero di bytes di spazio libero al di sotto del quale il plugin genera un errore di tipo *WARNING*

-a fa una comparazione per tutte le partizioni swap, una per una

**Esempio d'uso:**

```
check_swap -w 10% -c 5%
```

### 3.4 Check\_users

**Comportamento:** controlla il numero di utenti correntemente collegati al sistema locale e genera un errore se il valore eccede le soglie specificate.

**Uso:**

```
Check_users -w <warn> -c <crit>
```

**Parametri:**

-w <warn> numero intero di utenti al di sopra del quale il plugin genera un errore di tipo *WARNING*

-c <crit> numero intero di utenti al di sopra del quale il plugin genera un errore di tipo *CRITICAL*

Esempio d'uso:

```
Check_users -w 5 -c 10
```

## 3.5 Check\_icmp

**Comportamento:** controlla l'host specificato inviando pacchetti icmp; genera errore se il ritardo (RTA) della risposta o la percentuale di pacchetti persi (packet loss) sono maggiori di valori dati.

Il protocollo ICMP (*Internet Control Message Protocol*) è usato dai router per segnalare eventi inattesi. Viene anche utilizzato per testare la rete: per esempio viene usato da comando PING (*Racket Internet Groper*) per verificare se è possibile comunicare con un host [6].

**Uso:**

```
Check_icmp [parametri] -H <hostaddress>
```

**Parametri:**

-H indirizzo IP dell'host da controllare

-w soglia critica al di là della quale il plugin genera un errore di tipo *WARNING*; può essere espressa in millisecondi (di default 200.0ms) o in percentuale di pacchetti persi (dei default 40%)

-c soglia critica al di là della quale il plugin genera un errore di tipo *CRITICAL*; può essere espressa in millisecondi (di default 500.0ms) o in percentuale di pacchetti persi (dei default 80%)

- n numero di pacchetti da inviare (di default 5)
- i intervallo di tempo massimo tra i pacchetti (di default 80.0ms)

Nota: le unità di misura del ritardo possono essere di microsecondi (us), millisecondi (ms) e secondi (s).

### Esempio d'uso

```
Check_icmp -w 10% -c 30% -H 192.168.12.1
```

## 3.6 Check\_dhcp

**Comportamento:** questo plugin controlla la disponibilità di un server DHCP su una rete. Genera errori se il server non è raggiungibile, o se il messaggio *DHCPOFFER* arriva dopo un ritardo superiore ad un certo timeout specificato.

Il protocollo DHCP (*Dynamic Host Configuration Protocol*) è un sistema di tipo client/server per la configurazione automatica e dinamica degli host. Il server viene configurato con degli intervalli di indirizzi IP (*scope*) che può assegnare. Oltre all'indirizzo IP può fornire al client anche altre informazioni di configurazione come la maschera di sottorete, il gateway o il server DNS.

### Uso:

```
check_dhcp [-r <requestedip>] [-i <interface>]  
[-t <timeout>] -s <serverip>
```

### Parametri:

- s <serverip> indirizzo IP del server DHCP da controllare
- r <requestedip> indirizzo IP che dovrebbe essere offerto dal server
- i <interface> interfaccia di rete da usare per il controllo (ad esempio *eth0*)
- t <timeout> intervallo di tempo massimo (in secondi) da aspettare per l'arrivo di un messaggio *DHCPOFFER*

**Esempio d'uso:**

```
check_dhcp -i eth0 -t 20 -s 192.168.12.1
```

## 3.7 Check\_dns

**Comportamento:** questo plugin usa il programma nslookup per ottenere l'indirizzo IP del dominio/host specificato. Può essere usato un server DNS opzionale, ma se non è specificato, viene usato il server (o i server) DNS di default presenti nel file */etc/resolv.conf*.

Nslookup è un programma per le richieste DNS presente in qualsiasi distribuzione Linux.

**Uso:**

```
check_dns -H <host> [-s <server>]  
[-a <expected-address>] [-A] [-t <timeout>]
```

**Parametri:**

-H <host> l'indirizzo stringa che si vuole controllare (ad esempio *www.unibo.it*)

-s <server> Server DNS opzionale con cui si vuole controllare (al posto di quello di default)

-a <expected-address> Indirizzo IP opzionale che ci si aspetta dal server DNS come risposta (dopo aver risolto la corrispondenza nome-indirizzo)

-t <timeout> Secondi prima che la connessione venga chiusa (timeout), di default è 10

Viene generato errore se non è possibile risolvere il nome o se il timeout scade.

**Esempio d'uso:**

```
check_dns -H www.google.it -s 192.168.12.2 -t 20
```

Controlla il server DNS 192.168.12.2 con l'indirizzo `www.google.it` e impone un timeout di 20 secondi

## 3.8 Check\_ssh

**Comportamento:** Prova a connettersi ad un server SSH attraverso un indirizzo e una porta specificati.

**Uso:**

```
check_ssh [-46] [-t <timeout>] [-r <remote version>]
[-p <port>] [-H] <host>
```

**Parametri:**

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default: 22)
- 4 Usa una connessione IPv4
- 6 Usa una connessione IPv6
- t <timeout> Secondi prima che la connessione venga chiusa (timeout), di default è 10
- r <remote version> Genera un errore di tipo *WARNING* se la versione del server SSH non coincide con la stringa (ad esempio: `OpenSSH_3.9p1`)

**Esempio d'uso:**

```
check_ssh -H 192.168.12.5
```

## 3.9 Check\_ups

**Comportamento:** questo plugin cerca di determinare lo stato di un UPS (*Uninterruptible Power Supply*) su un host remoto o locale. Se l'UPS è acceso o

alimentato dalla rete elettrica, viene dato a Nagios il valore OK di ritorno. Se la batteria è in uso (UPS scollegato dalla rete elettrica) viene emesso un valore *WARNING*. Infine, se l'UPS è spento o la batteria è quasi scarica viene emesso un errore di tipo *CRITICAL*.

Deve essere in esecuzione il programma *Network UPS Tools* scaricabile da [www.networkupstools.org](http://www.networkupstools.org) affinché il plugin funzioni.

### Uso:

```
Check_ups -H <host> -u <ups> [-p <port>]
[-v <variable>] [-w <warn_value>] [-c <crit_value>]
[-to <to_sec>] [-T]
```

### Parametri:

- H <host> Indirizzo IP o nome dell'host
- p <port> Numero di porta (intero, di default: 3493)
- u <ups> Nome (stringa) del UPS
- T L'output delle temperature viene espresso in gradi Celsius
- w <warn\_value> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*
- c <crit\_value> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout), di default è 10
- v <variable\_to\_check> Variabile da controllare

Con l'opzione [-v] si può specificare cosa controllare (come temperatura, voltaggio, carica della batteria eccetera) e utilizzare i parametri -w e -c per i valori di allarme relativi a quella variabile

Se l'host remoto ha più di un UPS si può monitorare quello desiderato specificandone il nome nell'opzione [-u].

**Esempio d'uso:**

```
Check_ups -H 192.168.12.9 -v temperature -w 50 -c 60
```

## 3.10 Check\_disk\_smb

**Comportamento:** controlla la percentuale di spazio occupato di dischi con sistemi operativi Windows in remoto tramite un client Samba.

**Uso:**

```
check_disk_smb -H <host> -s <share> [-u <user>]  
[-p <password>] -w <warn> -c <crit> [-W <workgroup>]  
[-P <port>]
```

**Parametri:**

- H <host> Nome NetBIOS del server (del computer sul quale è attiva una condivisione)
- s <share> Nome della condivisione da controllare
- W <workgroup> Gruppo di lavoro o nome del dominio usato (di default “*WORKGROUP*”)
- u <user> Nome utente per il login al server (di default “*guest*”)
- p <password> Password per il login al server (di default “*guest*”)
- w <warn> Numero intero (seguito da k, M o G) o percentuale di spazio occupato al quale il programma genera un errore di tipo *WARNING* (di default 85%). Il numero intero si riferisce allo spazio rimanente.
- c <crit> Numero intero (seguito da k, M o G) o percentuale di spazio occupato al quale il programma genera un errore di tipo *CRITICAL* (di default 95%). Il numero intero si riferisce allo spazio rimanente.
- P <port> Porta da usare per la connessione. Certi sistemi Windows utilizzano la 139, altri la 445 (di default è la porta settata dal client Samba *smbclient*)

**Note:** nelle soglie dei parametri [-w] e [-c] “k”, “M” e “G” significano rispettivamente (kilobytes, Megabytes, Gigabytes).

La percentuale di spazio per l'errore di tipo *WARNING* deve essere più piccola di quella dell'errore di tipo *CRITICAL*.

Lo spazio libero del parametro di *WARNING* deve essere più grande di quello del parametro dell'errore *CRITICAL*.

**Esempio d'uso:**

```
Check_disk_smb -H winserver -s documenti -w 10M -c 20M
```

## 3.11 Negate

**Comportamento:** nega lo stato di un plugin di Nagios (ritorna in uscita *OK* per *CRITICAL*, e viceversa). Gli altri stati (*WARNING* e *UNKNOWN*) rimangono gli stessi. Per richiamare un plugin bisogna specificare tutto il percorso (ad esempio /usr/local/nagios/libexec/any\_plugin).

**Uso:**

```
negate [-t <timeout>] <definition of wrapped plugin>
```

**Parametri:**

-t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10), al termine dei quali genera un errore di tipo *CRITICAL*.

**Esempi d'uso:**

```
negate "/usr/local/nagios/libexec/check_ping -H host"
```

Richiama check\_ping e inverte il risultato.

```
negate "/usr/local/nagios/libexec/check_procs -a 'vi  
negate.c' "
```

Si devono usare gli apici doppi (“”) per il plugin da eseguire, si possono usare gli apici singoli (‘) per ulteriori parametri con spazio (vedi esempio).

## 3.12 Check\_smtp

**Comportamento:** questo plugin cerca di aprire una connessione *SMTP* con un host specificato.

Una connessione con successo ritorna un stato *OK*, una connessione rifiutata o che ha fatto scadere il timeout dà uno stato *CRITICAL*, gli altri errori uno stato *UNKNOWN*. Una connessione con successo ma una risposta non corretta dall'host controllato ritorna uno stato *WARNING*.

Il protocollo *SMTP* (*Simple Mail Transfer Protocol*) è utilizzato per inviare la posta attraverso un server *SMTP*. Il server copia i messaggi nelle appropriate caselle postali; se non è possibile viene spedita una notifica di errore al mittente [6].

### Uso:

```
check_smtp -H host [-p <port>] [-e <expect>]
[-C <command>] [-R <response>] [-f <from addr>]
[-w <warn>] [-c <crit>] [-t <timeout>] [-n] [-4|-6]
```

### Parametri:

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default: 25)
- 4 Usa una connessione IPv4
- 6 Usa una connessione IPv6
- e <expect> Stringa da aspettarsi come prima linea della risposta del server (di default: '220')
- n Sopprime i comandi *SMTP* command
- C <command> Specifica un comando *SMTP* (può essere usato ripetutamente)
- R <response> Risposta che ci si aspetta al comando eseguito (può essere usato ripetutamente)

- f <from addr> “FROM-address” da includere nel comando MAIL, richiesto da Exchange 2000
- w <warn> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*
- c <crit> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d’uso:**

```
check_smtp -H 192.168.12.11
```

### 3.13 Check\_file\_age

**Comportamento:** controlla “l’età” in secondi di un file specificato, ossia da quanto tempo è stato creato, e la dimensione del file.

**Uso:**

```
Check_file_age [-w <secs>] [-c <secs>] [-W <size>]  
[-C <size>] -f <file>
```

**Parametri:**

- w <secs> valore in secondi dell’età di un file al di là del quale viene generato un errore *WARNING*
- c <secs> valore in secondi dell’età di un file al di là del quale viene generato un errore *CRITICAL*
- W <size> dimensione del file (espressa in bytes) massima per l’errore di tipo *WARNING*
- C <size> dimensione del file (espressa in bytes) massima per l’errore di tipo *CRITICAL*
- f <file> il nome assoluto del file da controllare

**Esempio d'uso:**

```
check_file_age -w 10000 -f /home/stefano/tesi.doc
```

### 3.14 Check\_flexlm

**Comportamento:** controlla la disponibilità di un “license manager” chiamato *FLEXlm*(citazione). *FLEXlm* è il più popolare “license management tool” sul mercato. Si basa su una architettura client-server in cui il server mantiene in memoria tutte le licenze. Quando il client avvia un programma è obbligato a generare una connessione con il server per controllare la licenza. Il criterio con cui controllare le licenze varia a seconda della configurazione del license manager. Il license manager *FLEXlm* solitamente è avviato come singolo server o in modalità tre server. Il plugin produce in uscita *OK* se il singolo (o i tre) server sono accesi e funzionanti, genera *CRITICAL* se sono spenti, e *WARNING* nel caso che uno o due server su tre siano spenti.

**Uso:**

```
check_flexlm -F <filename> [-v] [-t] [-V]
```

- F <filename> Nome del file licenza (usualmente “license.dat”)
- v Stampa a video informazioni di debug (Nagios può troncane l’output)
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 15).
- V Mostra a video la versione e le informazioni sulla licenza

**Esempio d'uso:**

```
Check_flexlm -F "license.dat" -V
```

## 3.15 Check\_ftp

**Comportamento:** questo plugin testa una connessione FTP con l'host specificato.

Il protocollo FTP (*File Transfer Protocol*) è utilizzato per la trasmissione o la ricezione di file e richiede che l'utente si colleghi con nome e password (anche in modo anonimo, con nome *anonymous*) per avere accesso al server. Un server FTP utilizza due porte: la 20 (dati) e la 21 (controllo) [6].

**Uso:**

```
check_ftp -H <host> -p <port> [-w <warning time>]
[-c <critical time>] [-s <send string>]
[-e <expect string>] [-q <quit string>]
[-m <maximum bytes>] [-d <delay>]
[-t <timeout seconds>] [-v] [-4|-6] [-S <use SSL>]
```

**Parametri:**

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default: nessuna)
- 4 Usa una connessione IPv4
- 6 Usa una connessione IPv6
- s <send string> Stringa da inviare al server
- e <expect string> Stringa da aspettarsi nella risposta del server
- q <quit string> Stringa da inviare al server per iniziare una procedura di chiusura sicura della connessione
- m <maximum bytes> Chiude la connessione se viene ricevuto un numero di bytes superiore a questo valore di soglia
- d <delay> Secondi da aspettare tra l'invio di una stringa e la ricezione della risposta
- S Usa SSL per la connessione

-w <warning time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*

-c <critical time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*

-t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 15).

**Esempio d'uso:**

```
check_ftp -H 192.168.12.240 -p 21
```

### 3.16 Check\_mrtg

**Comportamento:** questo plugin controlla il valore medio o massimo di una variabile registrata in un file log MRTG.

MRTG (*Multi Router Traffic Grapher*) è uno strumento per il traffic monitoring di rete in grado di generare una serie di pagine HTML contenenti la rappresentazione grafica del traffico.

Questo tool, rilasciato sotto licenza GPL, consiste sostanzialmente in due componenti principali: uno script in linguaggio Perl ed un programma scritto in linguaggio C. Lo script Perl, grazie all'ausilio del protocollo SNMP (*Simple Network Management Protocol*) legge i contatori di traffico dei router, mentre il programma C si occupa di tenere traccia del traffico letto e di generare la rappresentazione grafica in formato PNG visualizzabile successivamente tramite un web browser.

MRTG fa uso di alcune librerie esterne: *gd* per la grafica, *libpng* per le immagini PNG e *zlib* per comprimere le immagini create. Se queste librerie non sono già presenti nel proprio sistema è necessario installarle.

Le caratteristiche principali di MRTG sono:

- Portabilità: E' disponibile per diverse versioni UNIX ed anche per Windows;

- Perl: Il programma è scritto in Perl ed il codice è liberamente disponibile e modificabile;
- Supporto SNMP: possibilità di monitorare qualsiasi variabile SNMP ed anche SNMPv2;
- Log di lunghezza costante: viene utilizzato un algoritmo che permette di mantenere costante la dimensione dei log;
- Identificazione affidabile interfaccia di rete: MRTG per identificare l'interfaccia di rete di un router può utilizzare oltre all'indirizzo IP anche la descrizione oppure l'ethernet address (MAC);
- Performance: le parti critiche del codice sono state scritte in linguaggio C;
- Personalizzazione: le pagine HTML prodotte da MRTG sono completamente personalizzabili;
- Grafica Free: le immagini PNG sono generate dalle librerie open source gd;
- Configurazione automatica: la configurazione è facilitata da alcuni tools forniti a corredo del programma;

Può essere scaricato dal sito:

<http://ee-staff.ethz.ch/~oetiker/webtools/mrtg/mrtg.html>

### Uso:

```
check_mrtg -F <log_file> -a <AVG | MAX> -v <variable>  
-w <warning> -c <critical> [-l <label>] [-u <units>]  
[-e <expire_minutes>] [-t <timeout>] [-v]
```

### Parametri:

- F <log\_file> Il file log MRTG che contiene i dati da monitorare
- e <expire\_minutes> Numero di minuti dopo i quali i dati MRTG sono considerati troppo vecchi
- a <AVG | MAX> Si specifica se monitorare i valori medi o massimi
- v <variable> Si specifica quale variabile bisogna monitorare
- w <warning> Valore di soglia per generare un errore di tipo *WARNING*

-c <critical> Valore di soglia per generare un errore di tipo *CRITICAL*  
-l <label> Breve descrizione dei dati (ad esempio: Conns, "Processor Load", In, Out)  
-u <units> Unità di misura per i dati (ad esempio: Packets/Sec, Errors/Sec, "Bytes Per Second", "% Utilization")

Se il valore supera la tensione di warning, viene generato un errore di tipo *WARNING*. Se il valore supera la tensione di critical, viene generato un errore di tipo *CRITICAL*. Se i dati contenuti nel file log sono più vecchi del valore <empire minutes>, viene generato un errore *WARNING*.

**Esempio d'uso:**

```
check_mrtg -F mrtg.log -a MAX -v 1 -w 10 -c 20
```

## 3.17 Check\_pop

**Comportamento:** questo plugin prova a generare una connessione *POP* verso l'host specificato. Se la connessione viene rifiutata genera un errore di tipo *CRITICAL*.

Il protocollo POP (*Post Office Protocol*) è utilizzato per comunicare coi i server di posta; possiede comandi per l'autenticazione del client, per copiare i messaggi e per cancellarli. La sua peculiarità è che i messaggi vengono copiati dalla casella di posta al computer dell'utente [6].

**Uso:**

```
Check_pop -H <host> -p <port> [-w <warning time>]  
[-c <critical time>] [-s <send string>]  
[-e <expect string>] [-q <quit string>]  
[-m <maximum bytes>] [-d <delay>]  
[-t <timeout seconds>] [-4|-6] [-S <use SSL>]
```

**Parametri:**

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default nessuna, usualmente la 110)
- 4 Usa una connessione IPv4
- 6 Usa una connessione IPv6
- s <send string> Stringa da inviare al server
- e <expect string> Stringa da aspettarsi nella risposta del server
- q <quit string> Stringa da inviare al server per iniziare una procedura di chiusura sicura della connessione
- m <maximum bytes> Chiude la connessione se viene ricevuto un numero di bytes superiore a questo valore di soglia
- d <delay> Secondi da aspettare tra l'invio di una stringa e la ricezione della risposta
- S Usa SSL per la connessione
- w <warning time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*
- c <critical time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d'uso:**

```
check_pop -H 192.168.12.5 -p 110
```

### 3.18 Check\_imap

**Comportamento:** questo plugin prova a generare una connessione *IMAP* verso l'host specificato. Se la connessione viene rifiutata genera un errore di tipo *CRITICAL*.

Il protocollo IMAP (*Interactive Mail Access Protocol*) è utilizzato per la lettura dei messaggi di posta come POP, ma permette di leggere la posta dal server senza copiarla sul computer dell'utente; in questo modo si può accedere alla propria posta da computer diversi [6].

### Uso:

```
check_imap -H <host> -p <port> [-w <warning time>]
[-c <critical time>] [-s <send string>]
[-e <expect string>] [-q <quit string>]
[-m <maximum bytes>] [-d <delay>]
[-t <timeout seconds>] [-4|-6] [-S <use SSL>]
```

### Parametri:

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default: nessuna)
- 4 Usa una connessione IPv4
- 6 Usa una connessione IPv6
- s <send string> Stringa da inviare al server
- e <expect string> Stringa da aspettarsi nella risposta del server
- q <quit string> Stringa da inviare al server per iniziare una procedura di chiusura sicura della connessione
- m <maximum bytes> Chiude la connessione se viene ricevuto un numero di bytes superiore a questo valore di soglia
- d <delay> Secondi da aspettare tra l'invio di una stringa e la ricezione della risposta
- S Usa SSL per la connessione
- w <warning time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*
- c <critical time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d'uso:**

```
check_imap -H 192.168.12.12 -p 150
```

## 3.19 Check\_nagios

**Comportamento:** questo plugin controlla lo stato del processo Nagios sul computer locale, tramite un check sullo status log; se è più vecchio del numero di minuti specificati nell'opzione `expire`, genera un errore. Controlla anche che il processo sia attivo nella lista dei processi di sistema.

**Uso:**

```
check_nagios -F <status log file> -e <expire_minutes>  
-C <process_string>
```

**Parametri:**

`-F <status log file>` Nome del file log da controllare (percorso assoluto)

`-e <expire_minutes>` Soglia in minuti dopo la quale il file log è considerato obsoleto (e quindi non aggiornato da Nagios)

`-C <process_string>` Sottostringa da cercare nella lista dei processi

**Esempio d'uso:**

```
./check_nagios -F /usr/local/nagios/var/status.log  
-e 5 -C /usr/local/nagios/bin/nagios
```

## 3.20 Check\_nrpe

**Comportamento:** questo plugin utilizza il modulo aggiuntivo *NRPE* per lanciare un servizio su un computer remoto.

Il plugin richiede un demone *NRPE* attivo sul computer remoto da controllare. Il demone deve essere configurato tramite il file `nrpe.conf` (vedi capitolo 1) appositamente per ogni comando remoto che si vuole lanciare. `Check_nrpe`, quindi, permette di lanciare plugin in remoto.

**Uso:**

```
check_nrpe -H <host> [-p <port>] [-t <timeout>]
[-c <command>] [-a <arglist...>]
```

**Parametri:**

- H <host> Indirizzo dell'host su cui è attivo il demone *NRPE*
- p <port> Numero di porta sulla quale è in ascolto il demone (di default 5666)
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).
- c <command> Il nome del comando che il demone remoto deve avviare.
- a<arglist> Parametri opzionali che devono essere passati al comando remoto. Se più di uno, devono essere separati da uno spazio.

**Esempio d'uso:**

```
check_nrpe -H 192.168.12.130 -c check_disk2
```

## 3.21 Check\_tcp

**Comportamento:** questo plugin genera e testa una connessione TCP con l'host specificato.

**Uso:**

```
check_tcp -H host -p port [-w <warning time>]
[-c <critical time>] [-s <send string>]
[-e <expect string>] [-q <quit string>]
```

```
[-m <maximum bytes>] [-d <delay>]  
[-t <timeout seconds>] [-4|-6] [-S <use SSL>]
```

**Parametri:**

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default: nessuna)
- 4 Usa una connessione IPv4
- 6 Usa una connessione IPv6
- s <send string> Stringa da inviare al server
- e <expect string> Stringa da aspettarsi nella risposta del server
- q <quit string> Stringa da inviare al server per iniziare una procedura di chiusura sicura della connessione
- m <maximum bytes> Chiude la connessione se viene ricevuto un numero di bytes superiore a questo valore di soglia
- d <delay> Secondi da aspettare tra l'invio di una stringa e la ricezione della risposta
- S Usa SSL per la connessione
- w <warning time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*
- c <critical time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d'uso:**

```
check_tcp -H 192.168.12.1 -p 80
```

## 3.22 Check\_udp

**Comportamento:** questo plugin genera e testa una connessione UDP con l'host specificato.

**Uso:**

```
check_udp -H host -p port [-w <warning time>]
[-c <critical time>] [-s <send string>]
[-e <expect string>] [-t <timeout seconds>]
```

**Parametri:**

-H <host> Nome dell'host o indirizzo IP

-p <port> Numero della porta da utilizzare (di default: nessuna)

-s <send string> Stringa da inviare al server

-e <expect string> Stringa da aspettarsi nella risposta del server

-w <warning time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*

-c <critical time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*

-t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d'uso:**

```
check_udp -H 192.168.12.1 -p 5000
```

## 3.23 Check\_time

**Comportamento:** questo plugin controlla l'orario di un host specificato.

**Uso:**

```
check_time -H <host_address> [-p port] [-u]
[-w variance] [-c variance] [-W connect_time]
[-C connect_time] [-t timeout]
```

**Parametri:**

- H <host> Nome dell'host o indirizzo IP
- p <port> Numero della porta da utilizzare (di default: 37)
- u Usa il protocollo UDP per la connessione al posto del TCP
- w <variance> Differenza di orario (in secondi) necessaria per generare un errore di tipo *WARNING*
- c variance Differenza di orario (in secondi) necessaria per generare un errore di tipo *CRITICAL*
- W <connect\_time> Tempo di risposta necessario per generare un errore di tipo *WARNING*
- C <connect\_time> Tempo di risposta necessario per generare un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d'uso:**

```
check_time -H 192.168.12.3 -w 30 -c 60
```

## 3.24 Check\_nt

**Comportamento:** questo plugin riceve informazioni da un servizio *NSClient* avviato in background su un computer Windows NT/2000/XP/2003 server.

*NSClient* è un programma per inviare sulla rete dati riguardo memoria, uso della cpu, tempo di accensione della macchina, spazio disponibile su disco ecc. Il binario da installare si trova al sito <http://nsclient.ready2run.nl>. Per l'installazione, compattare il file zip in una cartella a scelta ed eseguire da linea di comando (da prompt di Windows) `NSClient.exe /install`. Dopo di questo il programma è stato registrato come servizio, e può essere avviato con il comando `net start nsclient` o tramite la gestione servizi del Pannello di Controllo.

**Uso:**

```
check_nt -H <host> -v <variable> [-p <port>]
[-w <warning>] [-c <critical>] [-l <params>]
[-d SHOWALL] [-t <timeout>] [-s <password>]
```

**Parametri:**

-H <host> Nome dell'host o indirizzo IP

-p <port> Numero della porta da utilizzare (di default: 1248)

-s <password> Password necessaria per la richiesta

-w <warning> Soglia critica al di là della quale il plugin genera un errore di tipo *WARNING*

-c <critical> soglia critica al di là della quale il plugin genera un errore di tipo *CRITICAL*

-t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

-v <variable> Variabile da controllare a scelta tra le seguenti possibilità:

- CLIENTVERSION = Richiede al versione NSClient
- CPULOAD = Carico della CPU medio degli ultimi X minuti  
Richiede un parametro -l con la seguente sintassi  
-l <minuti >,<soglia warning>,<soglia critical>.
- UPTIME = Tempo trascorso dall'avvio del computer.
- USEDDISKSPACE = Dimensione e percentuale dello spazio usato su disco rigido. Richiede un parametro -l con la lettera dell'unità.  
Le soglie di errore si specificano con le opzioni -w e -c
- MEMUSE = Uso della memoria.  
Le soglie di errore si specificano con le opzioni -w e -c
- SERVICESTATE = Controlla lo stato di uno o più servizi  
Richiede un parametro -l con la seguente sintassi:  
-l <service1>,<service2>,<service3>,...

Si può specificare l'opzione -d SHOWALL nel caso si voglia vedere la lista dei servizi avviati

- PROCSTATE = Controlla se un processo è attivo  
Stessa sintassi di SERVICESTATE.

**Esempio d'uso:**

```
check_nt -H 192.168.12.2 -V CPULOAD -l 10,50,60
```

Controlla il carico medio della CPU negli ultimi 10 minuti: genera un *WARNING* se è maggiore del 50%, genera un *CRITICAL* se è maggiore del 60%.

### 3.25 Check\_http

**Comportamento:** questo plugin testa il servizio HTTP su un host specificato. Può controllare un web server normale (http) e sicuro (https), seguire ridirezioni, controllare i messaggi di risposta del server, controllare il tempo di connessione, testare certificati.

Tenta di generare una connessione HTTP con l'host. Una connessione con successo genera uno stato OK, connessioni rifiutate o con timeout scaduto generano uno stato *CRITICAL*, gli altri errori uno stato *UNKNOWN*. Una connessione corretta, ma messaggi di risposta sbagliati dall'host generano uno stato *WARNING*.

Il plugin può anche testare se sia possibile una connessione tramite SSL, o controllare se il certificato X509 sia ancora valido per un numero specificato di giorni.

**Uso:**

```
check_http -H <vhost> | -I <IP-address> [-u <url>]  
[-p <port>] [-w <warn time>] [-c <critical time>]  
[-t <timeout>] [-a auth] [-A <useragent>] [-N]  
[-f <ok | warn | critical | follow>] [-e <expect>]  
[-4|-6] [-M <age>] [-T <con_type>] [-C <days>]
```

**Parametri:**

- H <vhost> Parametro host name per i server che utilizzano virtual host. Se viene usata una porta diversa dalla 80, specificarlo subito dopo l'indirizzo (ad esempio *example.com:5000*).
- I <IP-address> Indirizzo Ip o nome dell'host (utilizzare l'indirizzo numerico per evitare la richiesta al DNS)
- p <port> Numero della porta (di default: 80)
- 4 Utilizza il protocollo ipv4 per la connessione
- 6 Utilizza il protocollo ipv6 per la connessione
- S Connessione tramite SSL
- C <days> Minimo numero di giorni affinché un certificato sia valido (quando l'opzione è in uso non viene controllato l'url, ma solo il certificato).
- e <expect> Stringa che ci si aspetta nella prima linea della risposta del server (default: HTTP/1.)
- u <url> URL per GET o POST ( di default: /)
- N Non aspetta il corpo (body) del documento: smette di leggere dopo le intestazioni.
- M <age> Genera un errore di tipo *WARNING* se il documento è più vecchio di <age> secondi. Il numero può anche essere nella forma "10m" per i minuti, "10h" per le ore e "10d" per i giorni.
- T <con\_type> Specifica l'intestazione (header) Content-Type
- a <username:password> Nome utente e password per i siti con basic authentication
- A <useragent> Stringa da inviare nell'header http come "User Agent"
- f <ok|warning|critical|follow> Come comportarsi ad una ridirezione
- w <warning time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *WARNING*
- c <critical time> Tempo di risposta massimo (double, in secondi) al di là del quale emette un errore di tipo *CRITICAL*
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

**Esempio d'uso:**

```
check_http -w 5 -c 10 --ssl www.verisign.com
```

Quando il server 'www.verisign.com' invia il proprio content in meno di 5 secondi, viene generato uno stato *OK*. Quando lo invia ma a tempo scaduto (in più di 5 secondi) viene emesso un errore di tipo *WARNING*. Ad ogni altro errore, viene generato un errore di tipo *CRITICAL*. Nell'esempio inoltre viene utilizzata una connessione sicura tramite SSL.

```
check_http www.verisign.com -C 14
```

Quando il certificato di 'www.verisign.com' è valido da più di 14 giorni, viene generato uno stato *OK*. Quando il certificato è ancora valido, ma da meno di 14 giorni, viene emesso un errore *WARNING*. Se il certificato è scaduto, un errore di tipo *CRITICAL*.

## 3.26 Check\_log

**Comportamento:** analizza l'evolversi del contenuto di un file di log. La prima volta che viene richiamato memorizza in un file (old) il contenuto del log, e le volte successive li confronta alla ricerca di qualche differenza basata su un template specificato. Se non ci sono differenze viene generato uno stato *OK*, in caso contrario un errore di tipo *WARNING*.

**Uso:**

```
check_log -F <file_log> -O <old_file_log> -q <query>
```

**Parametri:**

- F <file\_log> il file log da controllare (in percorso assoluto, ad esempio /var/prova.log)
- O <old\_file\_log> il file di appoggio per i confronti successivi
- q <query> il template tra virgolette da usare

**Esempio d'uso:**

```
check_log -F /var/attacks.log -O /var/attacks.old -q  
"Attacco"
```

Ipotizziamo ad esempio che un programma scriva su un file il log degli attacchi ricevuti su una determinata porta:

```
Ore 14:30 Attacco da 192.168.12.1  
Ore 14:33 Attacco da 192.168.12.3
```

Un richiamo al plugin come nell'esempio porterebbe a riconoscere la presenza di nuovi attacchi, basando infatti la ricerca di differenze sulla parola "Attacco".

## 3.27 Check\_ping

**Comportamento:** usa il comando ping per verificare le statistiche di connessione con un host remoto. Si può utilizzare per controllare la percentuale di pacchetti persi o la media del round trip time (in millisecondi).

**Uso:**

```
check_ping -H <host_address> -w <wrta>,<wpl>%  
-c <crta>,<cpl>% [-p <packets>] [-t <timeout>] [-4|-6]
```

**Parametri:**

- 4 Utilizza il protocollo ipv4 per la connessione
- 6 Utilizza il protocollo ipv6 per la connessione
- H <host-address> Indirizzo Ip o nome dell'host
- w <wrta> ,<wpl>% Coppia di soglie WARNING (vedi commento finale)
- c <crta> ,<cpl>% Coppia di soglie CRITICAL (vedi commento finale)
- p <packets> Numero di pacchetti ICMP ECHO da inviare (Default: 5)
- t <timeout> Secondi prima che la connessione venga chiusa (timeout, di default è 10).

Le soglie sono nel formato <rt>,<pl>% dove <rt> è the la media del round trip travel time (ms) e <pl> la percentuale di pacchetti persi (entrambi se superati generano errori *CRITICAL* o *WARNING*).

**Esempio d'uso:**

```
check_ping -H 192.168.12.1 -w 50,60% -c 60,80%
```

# Capitolo 4: Linee guida per lo sviluppo di plugin

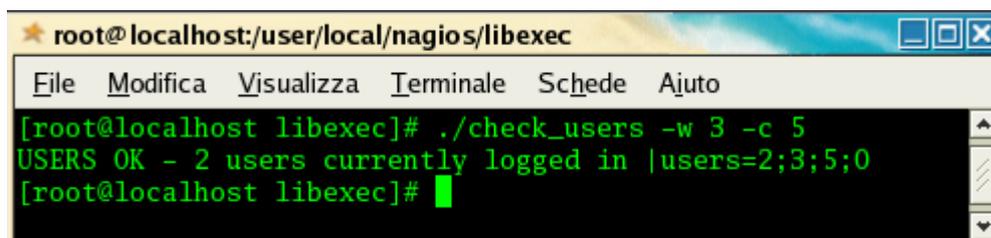
## 4.1 Nozioni basilari

I plugin sono strumenti fondamentali per il monitoraggio di rete tramite Nagios. Possono essere scritti in linguaggio C, Perl o in script di Shell, e devono poter funzionare stand-alone, cioè direttamente da linea di comando senza l'interposizione del programma principale.

Infatti Nagios non fa altro che gestire la comunicazione tra i risultati dei controlli dei plugin e l'interfaccia Web.

In questo capitolo cercherò di mostrare le linee guida per lo sviluppo di un plugin.

*Figura 1. Utilizzo di un plugin da linea di comando*



```
★ root@localhost:/user/local/nagios/libexec
File Modifica Visualizza Terminale Schede Aiuto
[root@localhost libexec]# ./check_users -w 3 -c 5
USERS OK - 2 users currently logged in |users=2;3;5;0
[root@localhost libexec]#
```

### 4.1.1 L'Output dei plugin

Poiché deve poter essere richiamato anche da linea di comando, un plugin deve fare uso dello Standard Output (STDOUT). In particolare, è consigliabile stampare a video qualcosa che mostri se il servizio funziona bene o, nel caso sia fallito, il perché del malfunzionamento. Bisogna cercare di scrivere un output relativamente corto (sugli 80 caratteri) poiché Nagios non è in grado di gestire un numero superiore. Non genera alcun tipo di errore, ma semplicemente vengono mostrati solo i primi caratteri del messaggio sull'interfaccia Web di Nagios. Tutto ciò che viene scritto sullo STDOUT, viene mostrato nella colonna "Status Information" dell'interfaccia [4].

## 4.1.2 Scrivere solo un linea di testo come Output

Nagios utilizza solo la prima linea di testo dello STDOUT per le notifiche (via mail o SMS) circa potenziali problemi. Scrivere altre righe all'infuori della prima vengono utilizzate solo per l'utilizzo stand-alone del plugin. Una raccomandazione che viene fatta dagli sviluppatori è *“remember, keep it short and to the point”*, cioè brevi, concisi e dritti al punto.

L'Output dovrebbe essere nel formato:

```
STATO DEL SERVIZIO: Testo informativo
```

Bisogna precisare che Nagios non legge lo stato del servizio dall'output ma dal codice di ritorno del plugin, quindi non è obbligatorio ai sensi del corretto funzionamento del programma scrivere l'output nel formato specificato, è solo consigliato.

## 4.1.3 Verbose output

Utilizzando l'opzione [-v] deve essere possibile visualizzare la modalità “verbose”, ossia mostra riga per riga, passo per passo tutto quello che fa il plugin. Al momento della stesura del codice bisogna tenere conto anche delle diverse modalità di questa opzione (che corrispondono a diversi livelli di dettagli):

**Tabella 1. Livelli della modalità *Verbose***

Livello	Tipo di uscita a video
0	Linea singola, output minimo. Sommario
1	Linea singola, informazioni (es. lista dei malfunzionamenti)
2	Multilinea, Output per il debug (es. comandi usati)
3	Molti dettagli per la diagnostica del plugin

### 4.1.4 Output guida a video

Oltre ai messaggi di funzionamento e di diagnostica, un buon plugin deve avere la possibilità, tramite il parametro [-h] o [--help], di mostrare a video una sorta di guida in linea all'uso. In particolare, è buona norma descriverne brevemente il funzionamento e in dettaglio tutti i vari parametri.

L'help a video (ma anche tutti gli altri messaggi) deve rispettare la dimensione di 80x25 della riga di comando.

### 4.1.5 Generazione dello stato del servizio

Ogni plugin deve avere un valore di ritorno (return state) che indica lo stato del servizio. Nel caso di programma scritto in linguaggio C, si dovrà predisporre una funzione main con una uscita intera:

```
int main(int argc, char **argv) {  
    ...  
    return STATE_OK;  
}
```

La corrispondenza tra i vari stati e i valori numerici è descritta nella Tabella 2 del paragrafo successivo. E' buona norma generare un errore *UNKNOWN* se il plugin è richiamato con parametri sbagliati o inesistenti.

### 4.1.6 Codici di ritorno (return codes)

La corrispondenza tra valori numerici e stati del servizio è basata su uno standard POSIX. Questo standard si basa su una serie di valori positivi (0,1,2,3); il programma NetSaint nelle versioni precedenti alla 0.0.7 non supportava i codici POSIX e utilizzava il valore "-1" per *UNKNOWN*. Nagios li supporta e li utilizza per default [4].

Certi plugin potrebbero emettere, quando viene causato un errore, un valore compreso tra 138 e 255 sullo STDOUT che è dovuto ad una superficialità nella programmazione del plugin: infatti in tal caso questo è dovuto alla mancanza della gestione dell'output inaspettato. E' consigliabile inserire una parte di codice che gestisca questi errori e invii a Nagios un codice *UNKNOWN*.

**Tabella 2. Codici di funzionalità di un servizio**

Valore Numerico	Stato del servizio	Descrizione dello stato
0	OK	Il plugin ha potuto testare il servizio e questo a funzionato correttamente.
1	Warning	Il plugin ha potuto testare il servizio ma è stata superata la soglia "warning" oppure sembra che non funzioni correttamente.
2	Critical	E' stata superata la soglia "critical" oppure il plugin non è stato avviato correttamente.
3	Unknown	Argomenti errati da linea di comando o il plugin non ha potuto testare lo stato del dato host/servizio.

## 4.2 Comandi di sistema e gestione dei file ausiliari

### 4.2.1 Comandi di sistema esterni

Non utilizzare `exec()`, `popen()`, ecc. per eseguire comandi esterni senza esplicitamente usare il percorso complete del programma esterno.

Non facendolo si espone il computer al rischio di vulnerabilità (Trojan horse, cavalli di Troia).

La documentazione ufficiale consiglia di osservare i sorgenti dei plugin esistenti per comprenderne il funzionamento.

Un esempio dell'uso di queste funzioni potrebbe essere:

```
#include <stdio.h>
int main (int argc, char **argv)
{
    int MAX_INPUT_BUFFER=50;
    int result = 0;
    char buf[MAX_INPUT_BUFFER];
    FILE *child_process;
    child_process=popen("/bin/ping -c 4
192.168.12.10", "r");
    while (fgets (buf, MAX_INPUT_BUFFER - 1,
child_process)!=NULL) {
        printf("%s",buf);
    }
    pclose(child_process);
    return result;
}
```

In cui viene creato un processo figlio che esegue il ping ad un host, e l'output viene ridirezionato sullo STDOUT.

## 4.2.2 Uso di spopen()

Viene consigliato l'utilizzo della funzione `spopen()`, variante di `popen()`, se si ha la necessità di eseguire comandi esterni. Questa funzione è stata scritta da Karl DeBisschop per il linguaggio C, ed è parte del pacchetto di Nagios. La funzione di chiusura ovviamente si chiama `spclose()`.

## 4.2.3 File temporanei

L'uso dei file temporanei è possibile, ma bisogna essere sicuri che il plugin, nel caso non riesca ad avere accesso al file (es. spazio su disco insufficiente, permessi di lettura/scrittura errati), non generi alcun errore (o comunque che l'errore sia gestito all'interno del codice).

A processo completo, il plugin deve cancellare i file temporanei utilizzati (quindi prima di inviare lo stato del servizio al programma centrale). Utilizzare sempre, per la memorizzazione, la directory `/tmp` e specificare ad ogni occasione il percorso assoluto (ad esempio `/tmp/check_OK.tmp`).

## 4.2.4 Gestione dei links

Se il plugin che si vuole sviluppare deve aprire qualche file, bisogna assicurarsi che questo file non sia un link, evitando così che il plugin segua un collegamento ad un altro punto del file system.

## 4.2.5 Gestione dell'input

Si deve sempre cercare di gestire al meglio tutte le eccezioni derivanti dall'input, come numero di parametri errato, formato dei parametri errato, eccetera.

## 4.3 Opzioni dei plugin

Come già specificato, un buon plugin deve avere la possibilità, tramite il parametro `[-h]` o  `[--help]`, di mostrare a video una sorta di guida in linea all'uso che rispetti le dimensioni standard 80x25 dello schermo.

### 4.3.1 Parametri

Per i plugin scritti in linguaggio C, gli sviluppatori di Nagios raccomandano lo standard *C getopt library* per le opzioni di tipo corto (ad esempio `-h`). Si possono anche utilizzare opzioni lunghe (ad esempio `--help`) seguendo lo standard *getopt\_long*. Per i plugins scritti in Perl, raccomandato il modulo `getopt::long`.

Gli argomenti sensibili alla posizione sono necessariamente da evitare. Questo significa che nel codice del plugin sarà inclusa una funzione per il controllo dei parametri più dinamica possibile [4].

Ci sono opzioni che non devono essere usate per altri scopi:

- `-V versione (--version)`

- -h guida (--help)
- -t timeout (--timeout)
- -w soglia d'allarme (--warning)
- -c soglia critica (--critical)
- -H Indirizzo Ip dell'host (--hostname)
- -v verbose output (--verbose)

In aggiunta alle opzioni riservate, si sono alcune opzioni standard:

- -C SNMP community (--community)
- -a password di autenticazione (--authentication)
- -l nome per il login (--logname)
- -p porta o password (--port o --passwd/--password)
- -u url o nome utente (--url o --username)

### 4.3.2 Le opzioni standard

Le opzioni -V o --version devono essere presenti in tutti i plugin. Per i plugin scritti in C solitamente si utilizza un richiamo alla funzione `print_revision()` contenuta nel file `utils.c` che ha due argomenti, il nome del comando e la versione del plugin.

Le opzioni -h o --help devono essere presenti in tutti i plugin. Per i plugin scritti in linguaggio C solitamente si utilizza un richiamo alla funzione `print_help()` contenuta nel file `utils.c` che richiama `print_revision()` e `print_usage()`, in modo da visualizzare una guida all'uso dettagliata.

Le opzioni -v o --verbose è consigliabile che siano presenti in tutti i plugin. L'utente potrà deciderne il livello come descritto nel paragrafo 4.1.3.

## 4.4 Invio di nuovi plugin

Creando nuovi componenti di Nagios si può anche decidere di inviarli agli sviluppatori, che decideranno se includerli nelle release successive del programma.

I requisiti minimi sono:

1. Le opzioni standard siano supportate (`--help`, `--version`, `--timeout`, `--warning`, `--critical`)
2. Non sia ridondante, ovvero non ci sia un altro plugin che faccia la stessa cosa
3. Uno degli sviluppatori abbia il tempo di analizzarne il codice e dichiararlo pronto per la distribuzione dei plugin standard di Nagios

I nuovi plugin devono essere inviati al “*SourceForge's tracker system for Nagiosplug new plugins*”, chiaramente come codice sorgente, all'indirizzo: ([http://sourceforge.net/tracker?group\\_id=29880&atid=541465](http://sourceforge.net/tracker?group_id=29880&atid=541465)).

Oppure possono essere inseriti nella comunità <http://www.nagiosexchange.com> (dopo l'opportuna registrazione al sito).

### 4.5 Un po' di esempi pratici

Il metodo migliore iniziale per mettere in pratica le linee guida descritte precedentemente è quello di scrivere qualche plugin chiamati “dummy”. Un programma “dummy” è privo di un significato e di uno scopo, con la semplice funzione di inviare a Nagios uno stato di ritorno voluto. Questo paragrafo elencherà alcuni esempi su realizzazioni di semplici plugin, sviluppati con il linguaggio C.

Il primo avrà lo scopo di generare uno stato OK. Anzitutto è necessario dare un valore alle costanti di stato, che dovranno rispettare lo standard del capitolo 4.1.6 (tabella 2).

```
enum {  
    STATE_OK,  
    STATE_WARNING,  
    STATE_CRITICAL,  
    STATE_UNKNOWN,  
};
```

Dopodichè definiamo qualche variabile utilizzate nella procedura della guida (è prassi definirle, ma non sono chiaramente indispensabili).

```
const char *programe = "check_OK";
const char *revision = "$Revision: 1.0 $";
const char *copyright = "1999-2004";
const char *email = "bianchins@hotmail.com";
```

Quindi una procedura per la visualizzazione delle modalità d'uso:

```
void print_usage (void)
{
    printf ("Usage:  ", programe);
}
```

A questo punto inseriamo il codice per la procedura di visualizzazione della guida (al richiamo del parametro -h o --help).

```
void print_help (void)
{
    printf ("Check_OK by Stefano Bianchini\n");
    printf ("\This plugin will simply return the OK
state \n\n");
    print_usage ();
}
```

Infine possiamo aggiungere la sezione relativa al main:

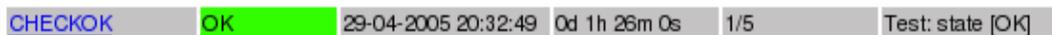
```
int main (int argc, char **argv)
{
    switch(argv) {
        case '-h':
            print_usage();
            break;
        case '--help':
            print_usage();
    }
```

```
        break;
    }
    int result = STATE_UNKNOWN;
    result = STATE_OK;
    printf ("Test: state [OK]");
    printf("\n");
    return result;
}
```

Si nota nel codice un semplicissimo controllo dei parametri di ingresso e la chiamata dove il plugin notifica a Nagios il successo del test:

```
return result
```

Configurando il plugin e lanciando il programma principale, questo è il risultato del nostro servizio visualizzato tramite l'interfaccia Web.



The screenshot shows a Nagios web interface with a table of services. The first row has a blue link 'CHECKOK', a green box with 'OK', a timestamp '29-04-2005 20:32:49', a duration '0d 1h 26m 0s', a progress indicator '1/5', and a description 'Test: state [OK]'.

<a href="#">CHECKOK</a>	OK	29-04-2005 20:32:49	0d 1h 26m 0s	1/5	Test: state [OK]
-------------------------	----	---------------------	--------------	-----	------------------

Figura 2: il plugin dummy in azione

Il programma dummy check\_OK ha dimostrato di funzionare egregiamente.

In questo caso il programma non necessitava di parametri (all'infuori di quelli per la guida) per cui il controllo dei parametri si riduceva al controllo dell'unico parametro accettato (-h o --help).

Nel caso di più parametri è necessaria una procedura di controllo che non tenga conto della posizione (come descritto nei paragrafi precedenti). Un buon esempio di funzione potrebbe essere:

```
int c;

int option = 0;
static struct option longopts[] = {
    {"hostname", required_argument, 0, 'H'},
    {"critical", required_argument, 0, 'c'},
    {"warning", required_argument, 0, 'w'},
    {"timeout", required_argument, 0, 't'},
```

```

        {"port", required_argument, 0, 'p'},
        {"expect", required_argument, 0, 'e'},
        {"send", required_argument, 0, 's'},
        {"verbose", no_argument, 0, 'v'},
        {"version", no_argument, 0, 'V'},
        {"help", no_argument, 0, 'h'},
        {0, 0, 0, 0}
    };
while (1) {
    c = getopt_long (argc, argv, "+hVvH:t: longopts,
&option);
        if (c == -1 || c == EOF || c == 1)
            break;
        switch (c) {
            case '?':
                /* print short usage statement */
                usage2 (_("Unknown argument"), optarg);
            case 'h':
                /* help */
                print_help ();
                exit (STATE_OK);
            case 'V':
                /* version */
                print_revision (progname, revision);
                exit (STATE_OK);
            case 'v':
                /* verbose mode */
                verbose = TRUE;
                break;
            case 't':
                /* timeout */
                socket_timeout = atoi (optarg);

```

```

        break;
    }
}

```

Come si nota vengono “compresi” i parametri tramite la funzione `getopt_long` (per supportarne anche il formato lungo) che man mano li salva nella variabile “c”. Un successivo case distingue i vari casi e lancia le procedure opportune. La struttura descritta nelle prime righe di codice definisce le corrispondenze tra parametri long e short, nonché specifica se il parametro richiede un argomento aggiuntivo (ad esempio il parametro `--timeout` dovrà avere a fianco un numero intero) oppure no.

Le successive prove di plugin sono state indirizzate verso gli stati diversi da OK: ho creato `check_WARNING`, `check_CRITICAL` e `check_UNKNOWN`. Mi è bastato riprendere il codice di `check_OK` e cambiare il valore di ritorno con la costante richiesta.

Il risultato finale di queste prove si può osservare nelle immagini seguenti:

<a href="#">CHECKCRITICAL</a>	CRITICAL	29-04-2005 20:04:59	0d 1h 24m 52s	3/3	Test: state [CRITICAL]
<a href="#">CHECKWARNING</a>	WARNING	29-04-2005 20:05:22	0d 1h 24m 29s	3/3	Test: state [WARNING]
<a href="#">check_UNKNOWN</a>	UNKNOWN	21-05-2005 11:01:15	0d 0h 0m 51s	1/3	Test: state [UNKNOWN]

Figura 3. I plugin in azione

Chiaramente nel caso si voglia fare qualcosa di più elaborato in linguaggio C, il consiglio degli sviluppatori è di utilizzare le funzioni che loro mettono a disposizione nella distribuzione dei plugin nei file header.

```

#include "common.h"
#include "netutils.h"
#include "utils.h"

```

## 4.6 Scheduling di un plugin

Come abbiamo già visto nel secondo capitolo, tramite l’opzione `normal_check_interval` è possibile definire (in caso di successo) ogni quanto tempo il plugin deve essere richiamato. Se si setta ad esempio quel

parametro al valore 1, ci si aspetta che Nagios esegua quel plugin ad intervalli di un minuto.

Entries sorted by **next check time** (ascending)

Host ↑↓	Service ↑↓	Last Check ↑↓	Next Check ↑↓	Active Checks	Actions
<a href="#">localhost</a>	<a href="#">Root Partition</a>	21-05-2005 11:24:36	21-05-2005 11:29:36	ENABLED	 
<a href="#">router</a>	<a href="#">CHECKCRITICAL</a>	21-05-2005 11:28:49	21-05-2005 11:29:49	ENABLED	 
<a href="#">router</a>	<a href="#">CHECKOK</a>	21-05-2005 11:28:54	21-05-2005 11:29:54	ENABLED	 
<a href="#">router</a>	<a href="#">CHECKWARNING</a>	21-05-2005 11:29:08	21-05-2005 11:30:08	ENABLED	 
<a href="#">scrivania</a>	<a href="#">Check_Arp</a>	21-05-2005 11:25:13	21-05-2005 11:30:13	ENABLED	 
<a href="#">fedora</a>	<a href="#">NRPE_CHECK_DISK</a>	21-05-2005 11:29:18	21-05-2005 11:30:18	ENABLED	 
<a href="#">fedora</a>	<a href="#">NRPE_CHECK_USERS</a>	21-05-2005 11:25:31	21-05-2005 11:30:31	ENABLED	 
<a href="#">winserver</a>	<a href="#">CHECKFTP</a>	21-05-2005 11:26:26	21-05-2005 11:31:26	ENABLED	 
<a href="#">localhost</a>	<a href="#">Current Users</a>	21-05-2005 11:26:44	21-05-2005 11:31:44	ENABLED	 
<a href="#">router</a>	<a href="#">CHECKHTTP</a>	21-05-2005 11:27:03	21-05-2005 11:32:03	ENABLED	 
<a href="#">router</a>	<a href="#">check_UNKNOWN</a>	21-05-2005 11:27:21	21-05-2005 11:32:21	ENABLED	 
<a href="#">winserver</a>	<a href="#">CHECKHTTP</a>	21-05-2005 11:27:39	21-05-2005 11:32:39	ENABLED	 

Figura 4. La schermata Scheduling Queue

Tutto questo vedendo la vista Service Detail non salta subito all'occhio: in realtà basta osservare la sezione Scheduling queue per rendersi conto che Nagios ha compreso le nostre intenzioni, cioè esegue quel determinato servizio una volta al minuto.

Volendo avere la certezza matematica che ciò che Nagios mostra corrisponde a verità, basta ricorrere ad un piccolo stratagemma.

Inventiamo un plugin fittizio, che generi sempre il valore di stato *OK*, ma facciamo in modo che ogni qualvolta venga richiamato scriva in append su un file temporaneo la data e l'ora dell'esecuzione. Chiameremo il file temporaneo `/tmp/check_sched.tmp`, e utilizzeremo lo script di shell.

Il codice è molto semplice:

```
#!/bin/sh
#Definizione Stati di uscita
STATE_OK=0
date >> /tmp/check_sched.tmp
```

```
exit $STATE_OK
```

La scrittura della data e dell'ora viene fatta tramite la chiamata `date`, ridirezionando l'output in append al nostro file temporaneo.

Dopo qualche minuto, il file `check_sched.tmp` contiene:

```
sab mag 21 11:21:49 CEST 2005
sab mag 21 11:22:49 CEST 2005
sab mag 21 11:23:49 CEST 2005
sab mag 21 11:24:49 CEST 2005
sab mag 21 11:25:49 CEST 2005
sab mag 21 11:26:49 CEST 2005
sab mag 21 11:27:49 CEST 2005
sab mag 21 11:28:49 CEST 2005
sab mag 21 11:29:49 CEST 2005
sab mag 21 11:30:49 CEST 2005
sab mag 21 11:31:49 CEST 2005
sab mag 21 11:32:49 CEST 2005
sab mag 21 11:33:49 CEST 2005
```

Che dimostra la veridicità di ciò che mostra Nagios nell'interfaccia Web: i servizi con parametro `normal_check_interval=1` vengono eseguiti ad intervalli di un minuto.

# Capitolo 5: Creazione di un plugin

## 5.1 Requisiti

Si vuole creare un plugin che controlli il cosiddetto “Ip Spoofing”, tramite un test sulla corrispondenza tra indirizzo Ip e indirizzo Mac. Si deve poter richiamare l'eseguibile con i parametri `-H<indirizzo ip dell'host>` e `-M <indirizzo Mac>`, dove quest'ultimo è l'indirizzo fisico che ci si aspetta da quel determinato indirizzo Ip. Se non corrisponde al vero, significa che qualche intruso si è inserito nella rete utilizzando un indirizzo Ip assegnato ad un'altra macchina.

## 5.2 Analisi dei Requisiti

Per prima cosa elenchiamo le descrizioni dei termini che compaiono nei requisiti.

### 5.2.1 Ip Spoofing

In una rete di computer, con il termine di IP spoofing si indica una tecnica tramite la quale si crea un pacchetto IP nel quale viene falsificato l'indirizzo IP del mittente.

Nell'header di un pacchetto IP si trova uno specifico campo, il Source Address, il cui valore indica l'indirizzo IP del mittente. Semplicemente modificando questo campo si può far credere che un pacchetto IP sia stato trasmesso da una macchina differente.

Questa tecnica può essere utilizzata per superare alcune tecniche difensive contro le intrusioni, in primis quelle basate sull'autenticazione dell'indirizzo IP. Infatti, è normale che in intranet aziendali l'autenticazione ad alcuni servizi avvenga sulla base dell'indirizzo IP, senza l'utilizzo di altri sistemi (come utente e password). Questo tipo di attacco ha tanto più successo tanto più i rapporti di "fiducia" tra due o più macchine sono forti.

Una delle difese che si possono attuare contro questo tipo di attacco è l'utilizzo di `packet_filtering`, settando opportune regole sulla base delle viene deciso quali pacchetti dall'esterno possono essere trasmessi all'interno della rete aziendale e viceversa. Nello specifico caso, per evitare un attacco basato sullo spoofing basta impostare una serie di regole che vieti il passaggio dall'esterno verso l'interno della rete aziendale di pacchetti IP che abbiano come indirizzo IP sorgente quello di una macchina interna. Ovviamente si possono settare anche delle regole in modo tale da evitare attacchi di spoofing dall'interno verso l'esterno.

L'IP spoofing risulta essere una tecnica inutile per ottenere anonimato (come invece molti credono), in quanto chi invia il pacchetto non sarà, generalmente, in grado di proseguire in modo coerente la comunicazione, dato che le risposte saranno inviate all'indirizzo IP modificato.

Si tratta di una tecnica utilizzata principalmente durante attacchi di tipo DoS e principalmente nella loro variante distribuita (o DDoS), per verificare che gli algoritmi e le policy di routing siano impostate correttamente.

### 5.2.2 Indirizzo Mac

L'indirizzo MAC (*MAC address*) viene detto anche *indirizzo fisico* o *indirizzo ethernet* o *indirizzo LAN*, ed è un codice di 48 bit (6 byte) assegnato in modo univoco ad ogni scheda di rete ethernet prodotta al mondo.

MAC è un acronimo che significa *Media Access Control* e viene utilizzato per l'accesso al mezzo fisico dal livello datalink secondo lo standard ISO/OSI.

I 48 bit del codice sono suddivisi in 12 cifre esadecimali: le prime 6 cifre individuano il produttore dell'interfaccia di rete mentre le successive corrispondono al numero di serie della scheda stessa. L'indirizzo MAC si scrive normalmente in 6 ottetti separati da un trattino (es. 00-50-FC-A0-67-2C) ed i primi 3 ottetti sono detti OUI (Organizationally Unique Identifier). Per questo tipo di indirizzi di solito si preferisce la notazione esadecimale anche per differenziarla dagli indirizzi Ip che usano la notazione decimale.

I blocchi di indirizzi MAC vengono assegnati ai produttori dall'IEEE ma non è chiaro se vengono assegnati a blocchi di  $2^{24}$  o di  $2^{25}$  indirizzi.

Ogni scheda ha un indirizzo unico perché i primi 24 bit sono identificativi della casa produttrice e i successivi della scheda. In questo modo ogni casa produttrice ha a disposizione  $2^{24}$  indirizzi, quindi più di 16 milioni di schede.

Si comprende, quindi, come l'indirizzo MAC non cambi se ci si sposta da una rete a un'altra, mentre lo stesso non accade per un indirizzo IP.

La conversione tra indirizzo MAC e indirizzo IP avviene mediante alcuni protocolli, il più conosciuto è ARP.

### 5.2.3 Protocollo ARP

Il protocollo ARP (*Address Resolution Protocol*), come specificato da RFC 826, è un protocollo che fornisce la "mappatura" tra l'indirizzo IP a 32bit (4byte) di un calcolatore e il suo MAC address, l'indirizzo fisico a 48bit (6 byte).

#### 5.2.3.1 Scopo del protocollo ARP

ARP è un protocollo di servizio, utilizzato in una rete di calcolatori che utilizzi il protocollo di rete IP sopra una rete di livello datalink che supporti il servizio di broadcast. Se questo servizio non è disponibile, come ad esempio in ATM, devono essere utilizzati altri meccanismi.

Per inviare un pacchetto IP ad un calcolatore della stessa sottorete, è necessario incapsularlo in un pacchetto di livello datalink, che dovrà avere come indirizzo destinazione il mac address del calcolatore a cui lo si vuole inviare. ARP viene utilizzato per ottenere questo indirizzo.

Se il pacchetto deve essere inviato ad un calcolatore di un'altra sottorete, ARP viene utilizzato per scoprire il mac address del gateway.

In ogni calcolatore, il protocollo ARP tiene traccia delle risposte ottenute in una apposita cache, per evitare di utilizzare ARP prima di inviare ciascun pacchetto. Le voci della cache ARP vengono cancellate dopo un certo tempo di inutilizzo.

#### 5.2.3.2 Funzionamento

L'host che vuole conoscere il mac address di un altro host, di cui conosce l'indirizzo IP, invia in broadcast una richiesta ARP (ARP-request), generata dal

protocollo IP, contenente l'indirizzo IP dell'host di destinazione ed il proprio indirizzo MAC.

Tutti i calcolatori della sottorete ricevono la richiesta. In ciascuno di essi il protocollo ARP verifica se viene richiesto il proprio indirizzo IP. L'host di destinazione che riconoscerà il proprio IP nel pacchetto di ARP-request, provvederà ad inviare una risposta (ARP-reply) in unicast all'indirizzo MAC sorgente, contenente il proprio MAC.

In questo modo, ogni host può scoprire l'indirizzo fisico degli altri host sulla stessa sottorete.

Bisogna osservare che il protocollo ARP viene usato tutte le volte che un host collegato ad una LAN deve inviare un messaggio ad un host sulla stessa LAN di cui conosce unicamente l'indirizzo di livello rete (IP), quindi lavora solo in subnet locali (non può attraversare router).

Il procedimento inverso viene chiamato RARP.

### 5.2.3.3 Sicurezza

Il protocollo ARP non prevede meccanismi per autenticare le risposte ricevute, quindi l'host che invia una richiesta "si fida" che la risposta arrivi dal "legittimo" proprietario dell'indirizzo IP richiesto, e identifica quell'indirizzo IP con il mac address che ha ricevuto.

È quindi molto facile configurare abusivamente un indirizzo IP su un host, purché questo sia collegato alla sottorete giusta, e l'indirizzo sia inutilizzato, oppure il legittimo proprietario sia spento.

Di conseguenza, il fatto che un host risponda ad un certo indirizzo IP non ci autorizza a "fidarci" di quell'host. Significa soltanto che è fisicamente collegato alla rete locale giusta (oppure che anche il routing è stato compromesso).

Nonostante questo, molti utilizzano l'indirizzo IP per autenticare gli utenti e consentire l'accesso a servizi non pubblici

## 5.3 Implementazione

Per prima cosa la scelta del linguaggio: le opzioni possibili sono:

- Linguaggio C

- Script di Shell
- Linguaggio Perl

La mia conoscenza del linguaggio Perl è un poco carente, il C possiede una grande flessibilità e potenza ma il codice in questo modo risulterebbe abbastanza lungo e complicato.

Leggendo per caso su una rivista per Linux, ho incontrato una frase che mi ha convinto: *“Non aspettate che qualcuno scriva un programma quando uno script può produrre una soluzione elegante...”*. Quindi, la mia scelta è ricaduta sullo script per bash.

La prima cosa da fare per uno script è aprire un editor di testi e scrivere:

```
#!/bin/sh
```

Chiunque abbia un po' di esperienza nell'uso di pagine CGI con un server Web è abituato a specificare all'inizio il path dell'eseguibile. In questo caso è la stessa cosa: infatti tutti gli script iniziano dicendo al sistema quale programma eseguibile far partire per leggere lo script ed eseguirne i comandi.

I parametri d'ingresso al nostro plugin saranno:

- L'indirizzo Ip dell'host che si vuole testare
- L'indirizzo Mac che si presume abbia quel determinato indirizzo Ip

Si ritiene fondamentale il richiamo di due comandi ben precisi comuni a tutti i sistemi UNIX:

- Ping: è uno strumento diagnostico che invia una richiesta del tipo “Ci sei?” verso un host specificato; se l'host è attivo e raggiungibile risponde in modo affermativo; ping controlla il tempo trascorso tra la domanda e la risposta e permette di controllare se ci sono forti ritardi. Utilizza il protocollo ICMP [5].
- Arp: è una utility che permette di gestire la cache ARP, ovvero la corrispondenza tra indirizzi IP e indirizzi fisici creata automaticamente dal sistema al bisogno; richiamando arp con l'opzione -n, si evita che risolva i nomi DNS (aumentando di fatto la velocità di risposta) [5].

## 5.3.2 Definizione degli stati di uscita

Definiamo gli stati di uscita ed eventuali altre variabili:

```
#Definizione Stati di uscita
STATE_OK=0
STATE_WARNING=1
STATE_CRITICAL=2
STATE_UNKNOWN=3
VERSION="v1.0"
```

## 5.3.3 Funzione print\_revision() e print\_help()

Inseriamo una funzione per la versione del plugin (standard di nagios):

```
#Funzione print_revision()
print_revision() {
echo "Check_ipspoofing (nagios-plugins 1.4) $VERSION"
    echo "-----"
    echo "The nagios plugins come with ABSOLUTELY NO
WARRANTY."
    echo "You may redistribute copies of the plugins
under the "
    echo "terms of the GNU General Public License."
    echo "For more information about these matters,
see the file"
    echo "named COPYING."
}
```

Come descritto dalle linee guida, il plugin necessiterà di una funzione per la visualizzazione della guida:

```
#Funzione print_usage()
print_usage() {
    echo "Check_ipspoofing by Bianchini Stefano
(bianchins@hotmail.com)"
```

```
    echo ""
    echo "Usage: check_arp -M <MacAddress> -H
<IpAddress>"
    echo "Usage: check_arp [-h] [--help]"
    echo "Usage: check_arp [-V]"
}
```

### 5.3.4 Parametri d'ingresso

Inseriamo ora una procedura di controllo dei parametri d'ingresso:

```
# CONTROLLO PARAMETRI
while test -n "$1"; do
    case "$1" in
        --help)
            print_help
            exit $STATE_OK
            ;;
        -h)
            print_help
            exit $STATE_OK
            ;;
        -V)
            print_revision
            exit $STATE_OK
            ;;
        -H)
            INDIRIZZOIP=$2
            shift
            ;;
        -M)
            INDIRIZZOMAC=$2
            shift
            ;;
    esac
done
```

```
        *)
            echo "Unknown argument: $1"
            print_usage
            exit $STATE_UNKNOWN
        ;;
    esac
    shift
done
```

Come richiesto, i parametri non sono posizionali (non importa immetterli in un ordine preciso), e grazie alla chiamata `shift` lo script è in grado di analizzarli uno per uno.

L'indirizzo Ip deve essere un parametro richiesto, e la mancata specifica deve portare ad un errore ed all'uscita dal programma (il plugin non può lavorare correttamente). Il medesimo controllo si deve effettuare sul parametro che specifica l'indirizzo fisico.

```
if [ -z $INDIRIZZOIP ]; then
    echo "STATUS CRITICAL: IP Address not specified"
    exit $STATE_CRITICAL
fi
if [ -z $INDIRIZZOMAC ]; then
    echo "STATUS CRITICAL: MAC Address not specified"
    exit $STATE_CRITICAL
fi
```

### 5.3.5 Il cuore del programma

Adesso è il momento del cuore del programma. Per prima cosa lo script dovrà eseguire un comando `ping` verso l'indirizzo Ip indicato, in modo che si aggiunga la voce relativa nella cache ARP. Per fare in modo che il risultato del ping non venga mostrato a video, viene ridirezionato l'output al device speciale "null".

```
#Richiamo il ping per la cache ARP e lo ridireziono
#sul device null
```

```
/bin/ping $INDIRIZZOIP -c 1 > /dev/null
```

Per leggere adesso la corrispondenza indirizzo Ip - indirizzo Mac, il programma deve andare a leggere nella cache ARP e elaborare il risultato:

```
#comandi per ottenere l'indirizzo mac di un
#determinato indirizzo ip
INDIRIZZOSARP=`/sbin/arp -n $INDIRIZZOIP |
/bin/grep '.....:.....:.....:.....:.....:.....' | /bin/cut -c 34-50`
```

Utilizzo varie pipe: la prima per selezionare, tramite comando grep, la linea dell'output relativa all'indirizzo fisico (che sarà della forma ..:..:..:..:..:..), la seconda per estrapolare dalla riga solo i caratteri interessati (ovvero l'indirizzo Mac).

Nel caso la variabile nella quale ho appoggiato il risultato risulti vuota, significa che il sistema non è stato in grado di leggere la corrispondenza nella cache ARP e quindi con buona probabilità il ping non è andato a buon fine. In un caso come questo il plugin non riesce a fare il suo lavoro, e deve emettere un errore di tipo *CRITICAL*.

```
#se la stringa è vuota significa che non sono
#riuscito a risalire all'indirizzo MAC quindi devo
#generare un errore CRITICAL
if [ -z $INDIRIZZOSARP ]; then
    echo "STATUS CRITICAL: Mac Addr NOT in ARP cache"
    exit $STATE_CRITICAL
fi
```

A questo punto si tratta di confrontare i due indirizzi fisici: quello presunto (come parametro) e quello effettivo (risultato di una lettura dalla cache ARP). Nel caso siano uguali viene generato uno stato OK; in caso contrario significa che qualcuno potrebbe essersi sostituito a quel indirizzo Ip e viene emesso un errore di tipo *WARNING*.

```
#Controllo indirizzo Mac
```

```
if [ $INDIRIZZOSARP = "$INDIRIZZOMAC" ]; then
    #l'indirizzo è giusto
    echo "STATUS OK: Mac Addr $INDIRIZZOMAC"
    exit $STATE_OK
else
    #l'indirizzo non corrisponde
    echo "STATE WARNING: Mac Addr $INDIRIZZOMAC does
    NOT match"
    exit $STATE_WARNING
fi
```

### 5.3.6 Un ultimo aggiustamento

Uno script per bash è, come tutto in Linux, un file; ha bisogno quindi dei relativi permessi di esecuzione per poter essere richiamato da linea di comando o da Nagios.

## 5.4 Inserimento in Nagios

Modifichiamo il file `check_commands.cfg` ed inseriamo le righe relative al nuovo comando che Nagios dovrà “imparare” ad usare:

```
# 'check_ipspoofing' command definition
define command{
    command_name    check_ipspoofing
    command_line    $USER1$/check_ipspoofing -H
                   $HOSTADDRESS$ -M $ARG1$
}
```

Quindi definiamo un nuovo servizio per un host a scelta che utilizzerà il comando descritto in precedenza; queste modifiche andranno eseguite sul file `service.cfg`:

```
define service{
    use                generic-service
    host_name          linux2
```

```
service_description      IPSPOOFING
is_volatile              0
check_period             24x7
max_check_attempts       3
normal_check_interval    5
retry_check_interval     1
contact_groups           mansarda
notification_interval    20
notification_period      24x7
notification_options     c,r
check_command            check_ipspoofing!00:11:2F:B3:F6:A2
}
```

Riavviamo Nagios, per controllare che le modifiche alla configurazione abbiano l'effetto voluto:

```
/etc/init.d/nagios restart
```

### 5.5 Verifica di funzionamento

La prima possibile verifica può essere effettuata direttamente da linea di comando. Posizioniamoci nella directory dei plugin di Nagios ed eseguiamo lo script:

```
cd /usr/local/nagios/libexec
./check_ipspoofing -H 192.168.12.2
-M 00:11:2F:B3:F6:A2
```

Ed otteniamo un buon risultato:

```
STATUS OK: Mac Addr 00:11:2F:B3:F6:A2
```

Se volutamente inseriamo un indirizzo fisico errato, ci aspettiamo un errore *WARNING*:

```
STATUS WARNING: Mac Addr 00:22:2F:B6:F8:A3 does not
```

```
match
```

Nel caso dimentichiamo l'indirizzo IP, il programma deve generare un errore di tipo *CRITICAL*.

```
STATUS CRITICAL: IP Address not specified
```

Specificando un indirizzo IP non esistente sulla rete, il programma avverte che l'indirizzo fisico non può essere ottenuto dalla lettura della cache ARP.

```
STATUS CRITICAL: MAC Addr NOT in ARP cache
```

Dimenticandosi il parametro relativo all'indirizzo fisico, il plugin avvisa l'utente che non tutti i parametri sono stati specificati.

```
STATUS CRITICAL: Mac Address not specified
```

Dopo aver verificato il corretto funzionamento del plugin da linea di comando, è stato inserito nella configurazione di Nagios: l'esito positivo è mostrato nella figura sottostante.

<a href="#">Check IpSpoofing</a>	OK	26-05-2005 18:51:06	0d 1h 7m 45s	1/3	STATUS OK: Mac Addr 00:11:2F:B3:F6:A2
----------------------------------	----	---------------------	--------------	-----	--

Figura 1. Il nuovo plugin in azione

# Capitolo 6: Una interfaccia web per Nagios

## 6.1 Motivazioni

Questo capitolo verrà dedicato ad una implementazione aggiuntiva di una interfaccia web per Nagios. Tutto nasce dalla necessità di avere una visione esterna controllata e limitata, nonché personalizzabile nella maniera che si ritiene più opportuna. Per visione esterna si intende un accesso da una rete diversa da quella locale; per quest'ultima invece l'interfaccia nativa di Nagios è quella più adatta e completa.

Ho deciso quindi di non mostrare tutto ciò che riguarda la configurazione del programma di monitoraggio per esigenze di sicurezza, come già accennato in un capitolo precedente. L'interfaccia avrà due parti ben distinte, una parte grafica e un “parser” che andrà ad interpretare i file di stato di Nagios.

## 6.2 Implementazione

Analizziamo ora le parti fondamentali da inserire, prendendo spunto dall'interfaccia nativa:

- Tactical Overview
- Service Details
- Host Details
- History Log

Implementando in linguaggio Php, questi quattro moduli saranno quattro diverse pagine dinamiche. Per i primi tre il parser necessita di aprire in lettura il file *status.dat*, per l'ultimo modulo di accedere al file *nagios.log*.

Per prima cosa ho creato i seguenti file:

- Config.php, che contiene i parametri di configurazione
- Include.php, che contiene le funzioni del parser

- Install.php, lo script che installa e configura l'interfaccia

Per l'implementazione dei moduli invece:

- Index.php, la pagina iniziale
- Tact.php, Tactical Overview
- Status.php, che implementa sia Service Detail che Host Detail
- Log.php, History Log

Non volendomi dilungare troppo con l'implementazione, invito a scaricare liberamente il codice completo (licenziato con la licenza GPL) al sito <http://www.nagiosexchange.com>, nella sezione *utilities*.

Il programma dà la possibilità di scelta tra due linguaggi al momento dell'installazione (italiano ed inglese).

Le immagini seguenti sono relative all'interfaccia web in funzione.

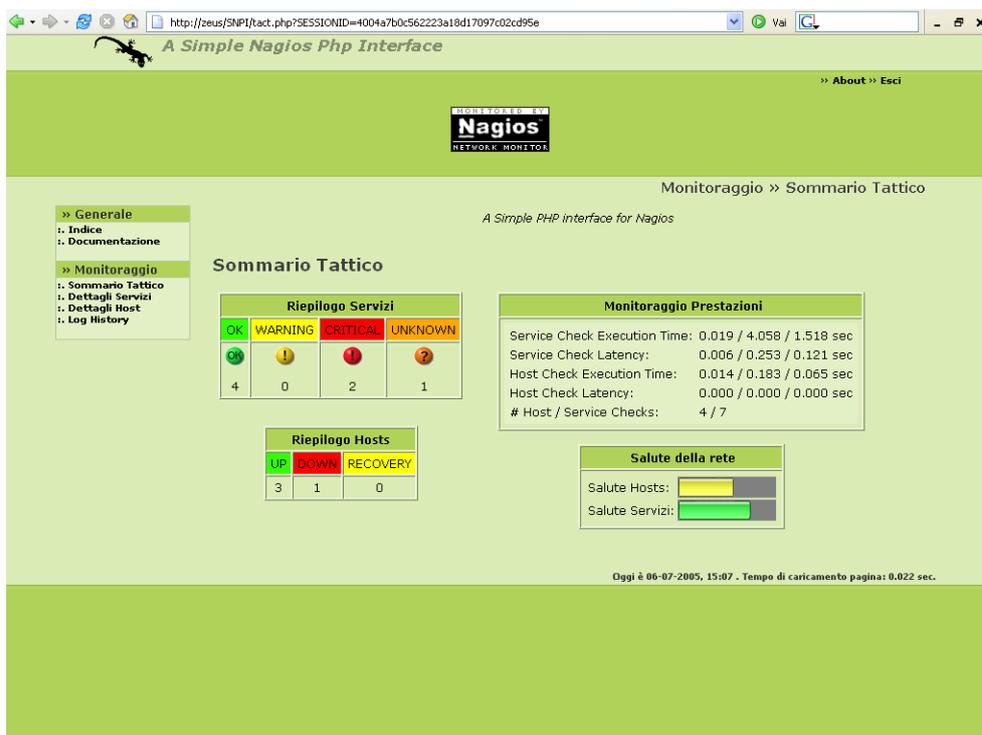


Figura 1. Tactical Overview

The screenshot shows the Nagios web interface for 'Dettagli Servizi'. The page title is 'Monitoraggio >> Dettagli Servizi'. A sidebar on the left contains navigation links: 'Generale', 'Indice', 'Documentazione', 'Monitoraggio', 'Sommarario Tattico', 'Dettagli Servizi', 'Dettagli Host', and 'Log History'. The main content area features a table with the following data:

Nome Servizio	Nome Host	Stato Servizio	Tentativi	Output
Current Users	localhost	OK	1 / 4	USERS OK - 3 users currently logged in
Root Partition	localhost	OK	1 / 4	DISK OK - free space: / 1179 MB (30%):
CHECKHTTP	router	OK	1 / 3	HTTP OK HTTP/1.0 200 OK - 444 bytes in 0.017 seconds
Check_IpSpoofing	scrivania	OK	1 / 3	STATUS OK: Mac Addr 00:11:2F:B3:F6:A2
PING	scrivania	UNKNOWN	3 / 3	/bin/ping -n -U -w 10 -c 5 192.168.12.2
CHECKFTP	winserv	CRITICAL	1 / 3	No route to host
CHECKHTTP	winserv	CRITICAL	1 / 3	No route to host

At the bottom of the page, it says: 'Oggi è 06-07-2005, 15:07 - Tempo di caricamento pagina: 0.002 sec.'

Figura 2. Service Detail

The screenshot shows the Nagios web interface for 'Dettagli Host'. The page title is 'Monitoraggio >> Dettagli Host'. The sidebar is identical to the previous screenshot. The main content area features a table with the following data:

Nome Host	Comando di testing	Stato Host	Tentativi	Output
localhost	check_tcp!80	UP	1 / 10	TCP OK - 0,001 second response time on port 80
router	check_http	UP	0 / 10	HTTP OK HTTP/1.0 200 OK - 444 bytes in 0.017 seconds
scrivania	check_dummy!0	UP	1 / 10	OK
winserv	check_tcp!80	DOWN	1 / 10	Network is unreachable

Below the table, there is a summary: 'Riepilogo: 3 Host Up, 1 Host Down, Su un totale di 4 Hosts'. At the bottom, it says: 'Oggi è 06-07-2005, 15:07 - Tempo di caricamento pagina: 0.002 sec.'

Figura 3. Host Detail

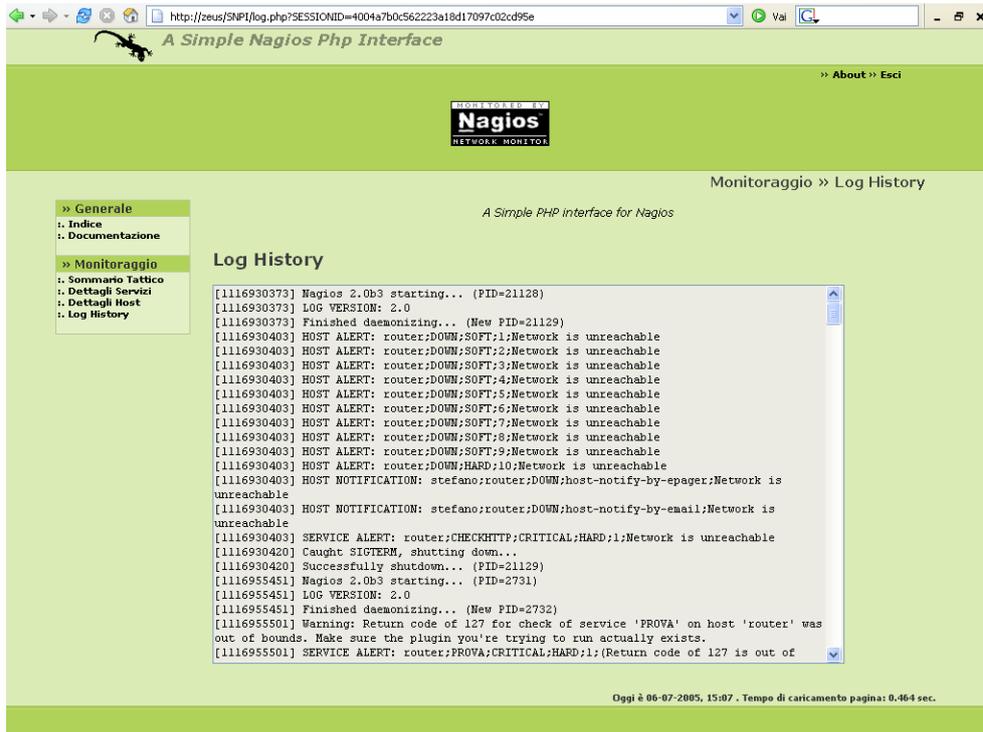


Figura 4. History Log

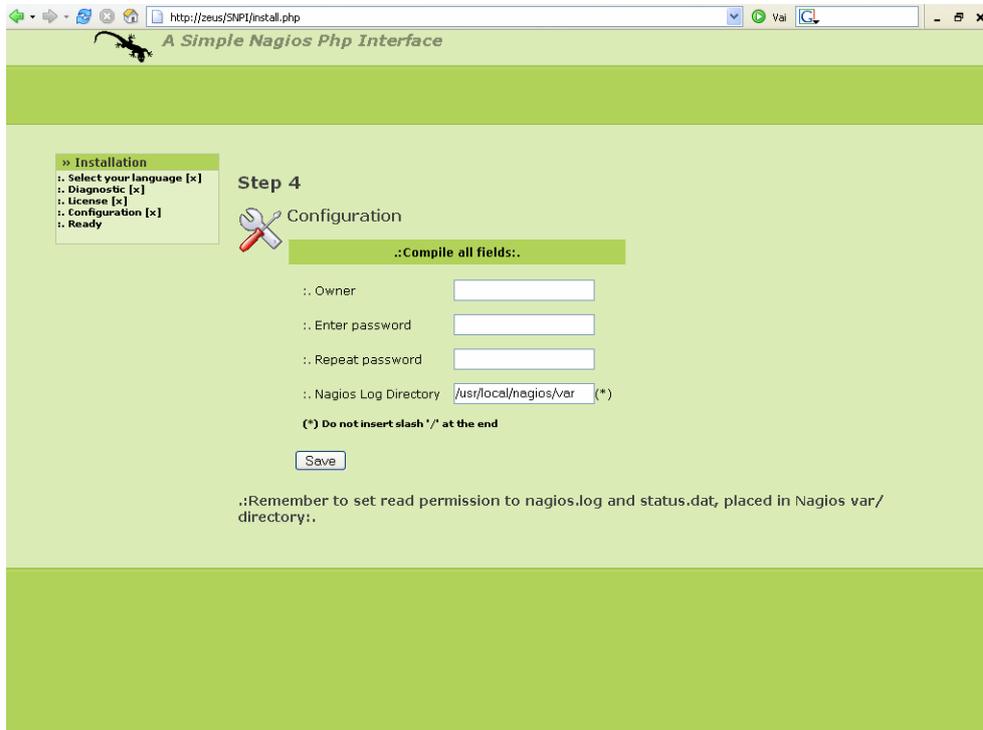


Figura 5. La procedura di installazione

## Capitolo 7: Conclusioni

Questa breve tesi ha la pretesa di essere un piccolo manuale iniziale per addentrarsi nell'utilizzo di un programma di monitoraggio di rete molto noto, Nagios. Le varie prove, sia a casa sia in laboratorio, hanno dimostrato che l'installazione e la configurazione di questo programma è relativamente semplice consultando queste pagine.

Parte di questo scritto è dedicata all'analisi dei plugin contenuti nel programma nativo e alla descrizione delle linee guida per lo sviluppo di nuovi “mattoni” che rendono Nagios uno strumento altamente personalizzabile.

Tutto questo mi ha permesso, tra l'altro, di immergermi nel mondo Linux con un approccio diverso dai precedenti tentativi di imparare ad usare un sistema operativo che non sia proveniente da Redmond.

Non per ultimo, infine, ho avuto la possibilità di provare sul campo tutte le nozioni acquisite installando e configurando Nagios per la rete del D.E.I.S. presso la sede di Ingegneria e di vedere il mio plugin contro l'ip spoofing funzionante in una rete di dimensioni molto diverse dalla LAN a 5 computer di casa.



## Bibliografia

[1] NAGIOS, *Documentazione Ufficiale*, Ethan Galstad

<http://nagios.sourceforge.net/docs>

[2] *Strumenti Open Source per il monitoraggio di reti locali*, Racci Riccardo

[3] *IMPLEMENTAZIONE DI UN SISTEMA DI MONITORAGGIO DI RETE L.A.N. CON TOOLS "OPEN SOURCE"*, Matteo Vitolo (Università degli Studi di Pavia)

[4] *Nagios plug-in development guidelines*, Nagios Plugins Development Team

<http://nagiosplug.sourceforge.net/>

<http://sourceforge.net/projects/nagiosplug/>

[5] *SISTEMI: elaborazione e trasmissione delle informazioni - II volume - Sistemi Operativi*, Fabrizia Scorzoni, Locher Editore

[6] *SISTEMI: elaborazione e trasmissione delle informazioni - III volume - Reti e protocolli*, Fabrizia Scorzoni, Locher Editore

[7] Sito Ufficiale NagiosExchange

<http://www.nagiosexchange.com>



# Ringraziamenti

Eccomi giunto ai ringraziamenti. Innanzitutto ringrazio il professor Cerroni che mi ha seguito e aiutato nella stesura di questa tesi, e l'ingegner Nazzareno Pompei per avermi affiancato nell'installazione di Nagios in Facoltà.

Il merito di questa tesi va a mio padre e mia madre, perché se non fosse stato per loro e per quel 486 dx2 a 66 Mhz che mi hanno comprato quando avevo dieci anni, a quest'ora non sarei dove sono; credo proprio che quello sia stato un ottimo investimento.

Un grazie a Romina, per essermi stata accanto e incoraggiato durante gli anni dell'Università (per riconoscenza le ho dedicato la tesi...).

Ed ora finalmente un bel ringraziamento a tutti i miei amici e compagni di studio: un grazie a Nick, il mio socio; a Pito (che mi ha dato le dritte sull'impaginazione della tesi) e Badile che hanno praticamente sempre studiato con me, persino la notte... un grazie anche a Roby, lo "smanettone" elettronico, e a Gallo, che con la sua intelligenza ci ha assistito con pazienza nella preparazione degli esami del primo anno...

E poi tutti gli altri, Mirko, Pesaro, Tommy, Alex Carmine (allora, quand'è che ti laurei?), Passione, Luga, Passero, Stacco, Corrado, Romagna, Di Mango, Antonello, Alex Del Bianco, Ugo, Tibo e tutti quelli che ho dimenticato, senza il vostro aiuto e la vostra compagnia questi anni non sarebbero trascorsi così velocemente.

Grazie, **Stefano**