Global Information Assurance Certification Paper

GSEC PRACTICAL ASSIGNMENT
VERSION 1.4b
OPTION 1

Practical Assignment Version: 1.4b

Option: 1

Course of Certificate: GIAC Security Essentials Certification GSEC

Title of Work: Distributed Monitoring For Security With Nagios

Author: Jasmine Chua Pei Chuin

Date: 12/08/2003

Table of Contents

Distributed Monitoring for Security with Nagios

Abstract

The purpose of this paper is not only to introduce to everyone the concept of distributed monitoring with Nagios but capturing the beauty of it to improve the security of computer networks. Firstly, an introduction to Nagios will be discussed to provide readers a brief overview of what Nagios is. Next, it will discuss how distributed network monitoring is an essential part to information security. It will then proceed to introducing the requirements needed to build a distributed Nagios network monitoring environment and demonstrate how Nagios can be configured to construct a distributed monitoring environment that helps improve the state of security of distributed networks. In essence, companies should be aware of the need for hiring specialized security analysts to perform round-the-clock systems monitoring to secure their resources.

Introduction to Nagios

Monitoring is defined as "observing and analysing the status and behaviour of the network, which involves end systems, intermediate systems and the core network. By monitoring a network the management entity can get the static, dynamic and statistical information of the network."[1] This is what the Nagios software can do. Nagios, written by Ethan Galstad, is actually nothing more than just a GNU General Public Licensed open-source network monitoring tool. Nagios is quoted as among "the top five open source packages for system administrators" by Æleen Frisch[2]. In fact, it does so by possessing the real-time capabilities to monitor almost any kind of running services on multiple platforms across networks. Nagios does it so well that analysts are able to manage networks comfortably including large networks. They should be able to address and distinguish between the problems of a particular server sitting on that particular network with another in a matter of seconds. The powerful feature underlying Nagios is its quick snapshot of the overall status of a particular network or networks shown below.

[1] http://www.tml.hut.fi/Opinnot/Tik-110.551/1999/papers/12ManagementOfIPngCore/ipcore.html
[2] http://www.onlamp.com/lpt/a/2787

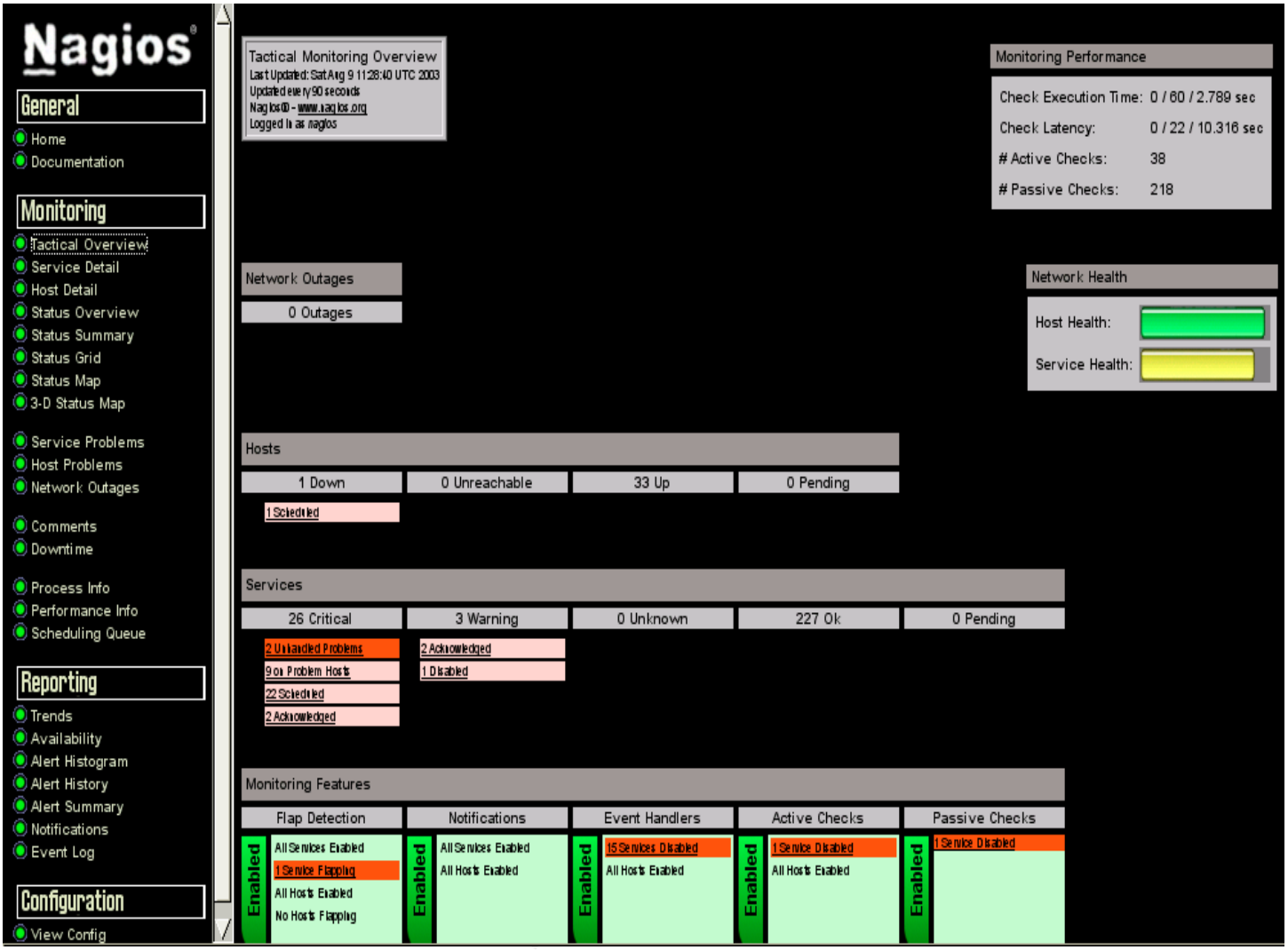


Figure 1. **Tactical Overview of Nagios.**

One attractive feature of Nagios is the colours because they are significant representatives of different status types (Green - "OK", Yellow - "WARNING", Red - "CRITICAL", Orange - "UNKNOWN"). Clicking on the "Status Overview or Summary" found at the side bar summarizes all the information one needs to know regarding the health of networks in neat graphical tables. The "Network Outages" further down shows something like a parent-child relationship when something fails. It tells one which are the children hosts and services that are affected by that particular parent host. This saves a lot of troubles when it comes to determining the root of a problem. Click on "Comments" and "Downtime", one can view all acknowledgments of all problems and the number of hosts in downtime beginning from the latest UTC timestamp. "Trends" and "Availability" allow one to generate hosts and services reports of what have been happening with customizable time spans. Next, the "Alert History" feature displays all events that have happened in the last one hour and so forth or identifies them based on their state types (warning, unknown, critical, recovery). One can also eliminate false positives with hiding options such as 'hide downtime alerts'. The "Alert Summary" allows one to generate summary reports of the most recent alerts. One can also monitor Nagios performance by clicking on either "Process" or "Performance info". And best of all, "Nagios monitors a wide variety of system properties, including system-performance metrics such as load average and free disk space; the presence of important services like HTTP and SMTP; and per-host network

availability and reachability."[2] Noisy alarms are triggered when thresholds configured for such metrics are exceeded and analysts can then attend to the problems and acknowledge these alarms by clicking and entering comments on another web page as shown in *Figure2* below. Such alarms can also be sent in other forms for instance, via email or smart messenging service depending on the configuration of Nagios.

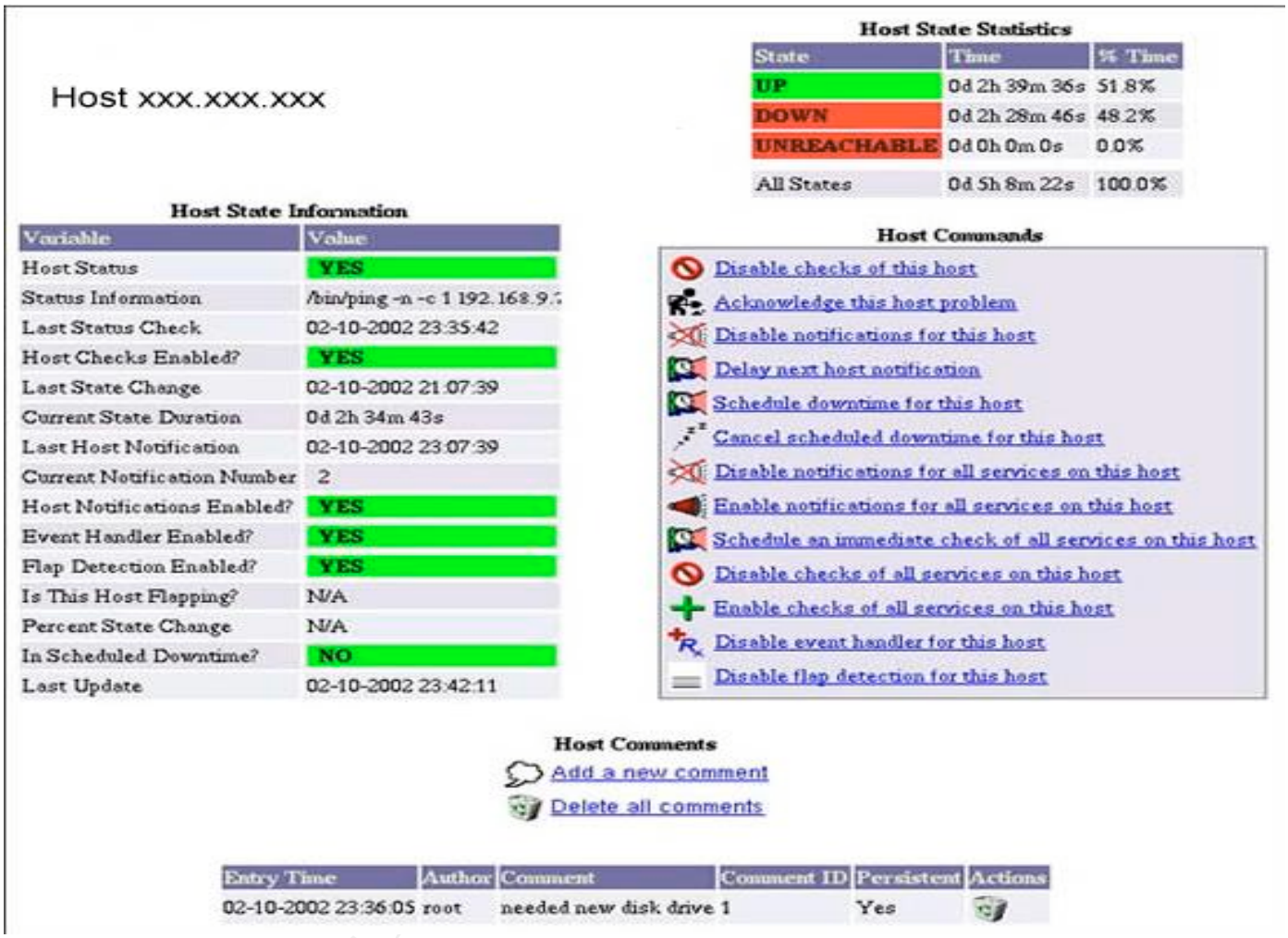


***Figure 2*. Detailed Host Status Information**.

The purpose of Nagios, in comparison to other tools such as the famous Fyodor's NMAP, Snort and many other applications for security auditing, intrusion monitoring and so forth, is not as clearly related to security. Having mentioned that one may wonder why the paper bothers to introduce Nagios then. Is it not deviating from the topic of security?

People may or may not have heard of Nagios but those that have, may not have actually thought that it could be used for security as well. In the next section, the paper will proceed to discuss why network monitoring tools like Nagios is essential for most networks today.

[2]http://www.onlamp.com/lpt/a/2787

The Usefulness of Nagios Distributed Monitoring for Security

"Companies can turn their security posture around by just integrating security into their day-to-day administration procedures. It can be that simple. After all, good security is 95 percent about good systems administration"[3]. And, an essential part of systems administration lies in close monitoring of the performance of servers and services. In fact, "non existent monitoring of enterprise security leaves organizations blind to information attacks targeted at their corporate networks." [4] The paper supports the view that the core of 'security' itself eventually still boils down to the need for round-the-clock monitoring. It is no different from the real world where policemen are required to do round-the-clock patrols. Nagios can be very useful especially when it comes to assisting operations in specialized security companies like Symantec[5] (take note of the picture in the foot note link 5 - URL) or large companies where they can afford an in-house 24x7 security department.

Imagine exploits such as a zero-day attack. "The knowledge that the zero day exploit will always be around. Some attacker will always know something that we do not and there is little we can do to protect against the unknown"[6]. Such unknown attacks simply defeat the whole purpose of having the best firewalls, network intrusion detection systems (NIDS), an incident response team that work diligently, when the attacks can still penetrate through into your internal network and damage or compromise servers without anyone's knowledge at all. "A key piece of your long-term security strategy -- especially after a successful attack has occurred -- is the development of a monitoring system that doesn't allow intrusions to go unnoticed."[7] This is when the importance of correlating the whole lot with a distributed monitoring environment using Nagios seeps in because at least early awareness is brought about whenever a host or service violation occurs. For instance, evidence has been shown that "ISPs claim heightened awareness and vigorous monitoring have helped reduce damage."[8]

Continuous monitoring of real-time events is important but for it to be successful in spotting hackers' activities, it requires some form of correlation with all logs coming from different security devices. "Managing security requires a global view of the activity (illicit or not) and the security level of the Information System. To that, security team (administrator or analyst) needs to manage the following information:

- ✍ Log generated by Firewall/VPN
- ✍ Alert generated by VA scanner (Vulnerability Assessment)
- ✍ Audit and security message generated by application and Operating System
- ✍ Warning and alert sent by system monitoring"[9]

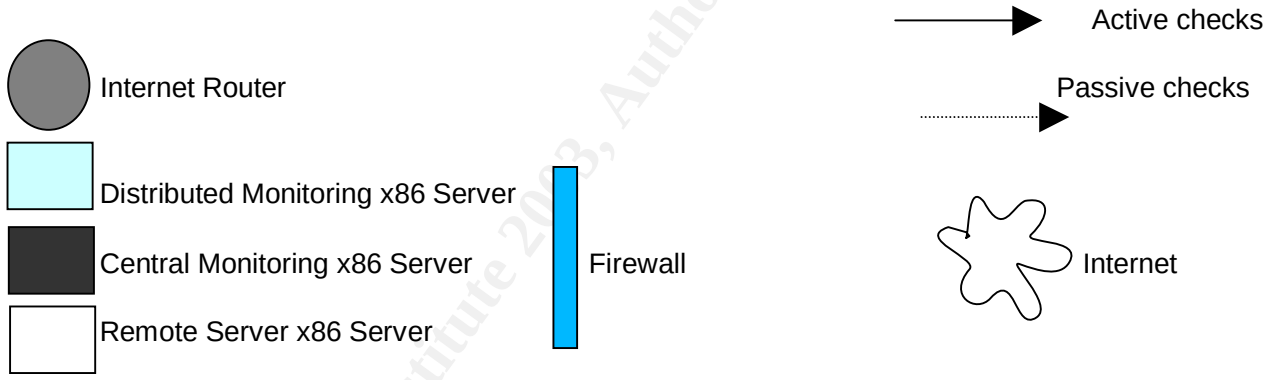
[3] http://www.infosecuritymag.com/articles/1999/winkler.shtml
[4] http://www.securityfocus.com/infocus/1231
[5] http://www.washingtonpost.com/wp-dyn/articles/A28625-2003Jan8.html
[6] http://www.itworld.com/nl/security_strat/10302002/
[7] http://insight.zdnet.co.uk/hardware/chips/0,39020436,2135496,00.htm
[8] http://www.nwfusion.com/news/2001/0205ddos.html
[9] http://www.exaprobe.com/labs/manage/

With Nagios, it is flexible enough to be combined with other security-related applications like Snort IDS or Prelude hybrid IDS system, and the end result is that the view of the information system described above is more achievable.

Why Nagios Distributed Monitoring?

From the information security perspective, other security devices like firewalls are deployed as needed because "splitting servers up not only by host but also by network limits and controls the level of trust between hosts greatly reducing the likelihood of a break-in"[10]. IDS applications have also adopted distributed network architectures. For instance, "Prelude is a new innovative Hybrid Intrusion Detection system designed to be very modular, distributed, rock solid and fast"[11]. And, therefore since Nagios is a network monitoring tool, it can be used by security analysts to monitor the status of distributed security devices. However, to be able to monitor these distributed security devices, there is a need to change the structure of Nagios network-monitoring itself. The following self-drawn diagrams attempt to show how the effectiveness and performance of Nagios network-monitoring could be improved in a distributed structure to monitor these remote security devices.

Legend for *self-drawn* diagrams:



Setting up a monitoring environment depends very much upon the size and layout of the underlying physical server infrastructure that is to be monitored. If the organisation's network is just kept within a single local area network (LAN), a simple setup of a centralised monitoring system is sufficient.

[10] http://w etadigm.co.uk/s utions/firewall.shtml
[11] http://www.prelude-ids.org/

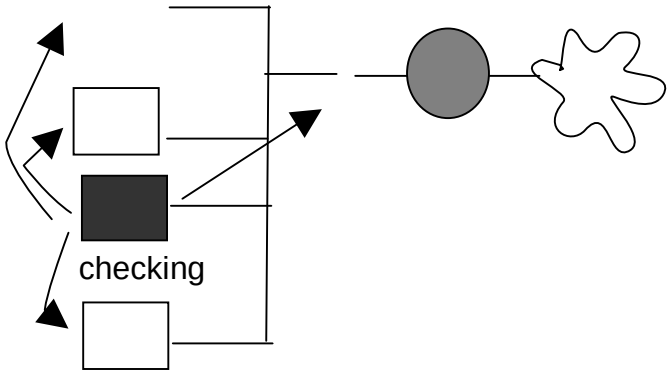


Figure 3. **Centralised Monitoring with Nagios.**

From the above diagram, one can see that there is a single Nagios server that performs host and service checks on the rest of the servers in the LAN, and thus security monitoring is done from a single console. This enables effective management of monitoring providing an overview of the current status of the LAN from a central monitoring server, without having the central controller to go out and "touch" every individual devices to check if they are well-protected.

However, a centralized Nagios monitoring setup is often futile in application to the networks today as Sun Microsystems commented on the following:

> With the growing use of individual workstations, personal computers, legacy systems and increased rates of adoption of the Internet and other networking platforms, development and implementation of a comprehensive distributed computing strategy is on the rise and a necessity for today's enterprise computing.[12]

[12] http://java.sun.com/products/javaspaces/backgrounder/

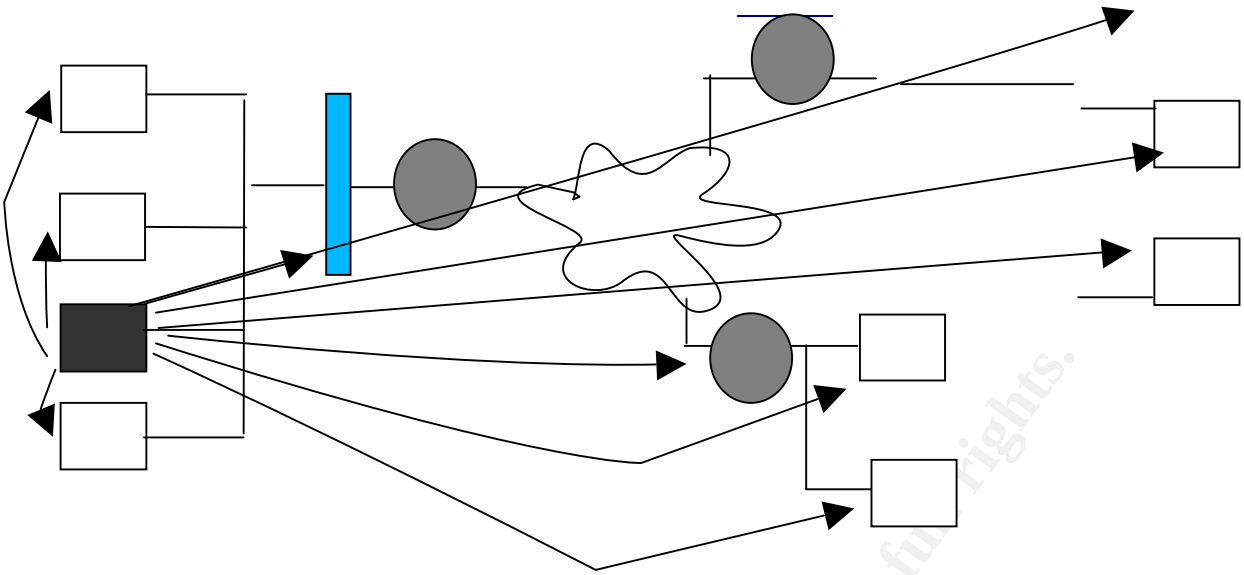


***Figure 4*. Nagios Centralised monitoring checking remote sites as well.**

With the adoption of distributed networks, complexity has to increase to ensure that each network segment is being monitored properly for security. The end result is that with centralised monitoring, the Nagios server will serve as a single point of failure and security monitoring itself is potentially unreliable. There is also a larger tendency for the Nagios server to become overloaded with many results of service checks to be processed. These results are read by the Nagios server through a pipe and "if Nagios does not read from this pipe fast enough, child processes can hang around waiting - this causes memory usage to rise. If the problem is serious enough, the server may start swapping these processes to disk, causing load issues."[13] A highly loaded monitoring server is obviously undesirable as efficiency is one of the key disciplines of an incident response team. Service checks may never get processed at all and the situation is just like implementing an NIDS that drops packets!

Local site - 192.168.10.0/24 **Remote site 1 - 192.168.20.0/24**

[13]http://www.Nagios.org/faqs/viewfaq.php?faq_id=115&expand=false&showdesc=true

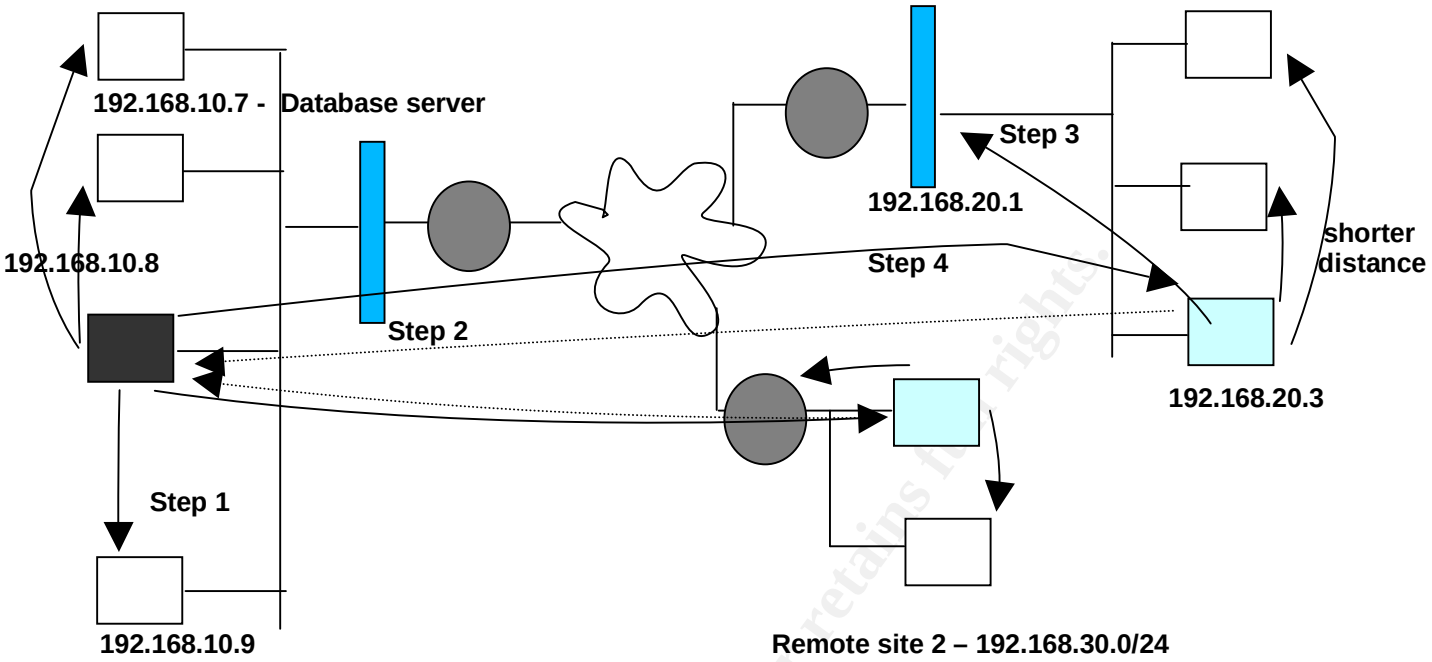


***Figure 5*. Distributed Monitoring with Nagios.**

In *Figure 5* above, by allocating the host and service checks to other distributed Nagios monitoring servers, results of the service checks at the central Nagios monitoring server are more updated facilitating real-time monitoring, because the distance that the packets have to travel to check remote site 1 for instance is lesser. Service checks are processed faster compared to *Figure 4* that relies on the central Nagios monitoring server to process these checks on remote LANs. "The overhead needed to process the results of passive service checks is much lower than that of 'normal' active checks, so make use of that piece of info if you're monitoring a slew of services"[11]. And, because distributed servers send passive service checks to the central server, the central Nagios monitoring server's load can be reduced with more distributed servers that add to the speed of processing service checks.

Moreover, the risk of a single point failure is diversified and minimised. If the central Nagios monitoring server fails, a distributed Nagios monitoring server can be configured to be informed and thus, taking over the tasks of the central Nagios monitoring server. Although the whole setup of Nagios distributed monitors comes at the expense of higher costs in terms of constant maintenance of more monitoring servers and increased administrative overheads, such costs can be offset when Nagios reside on the same server as the distributed IDS. Maintenance work can be done at the same time. Moreover, having a backup Nagios monitoring server is more important than cutting maintenance and administrative costs. Such costs should not then be a factor to consider when the distributed server plays an important role in securing networks. Properly secured corporate LANs should support usage of a wide variety of major security applications such as IDS together with firewall applications.

The Mechanisms in Nagios Distributed Monitoring and Security Issues in its Implementation

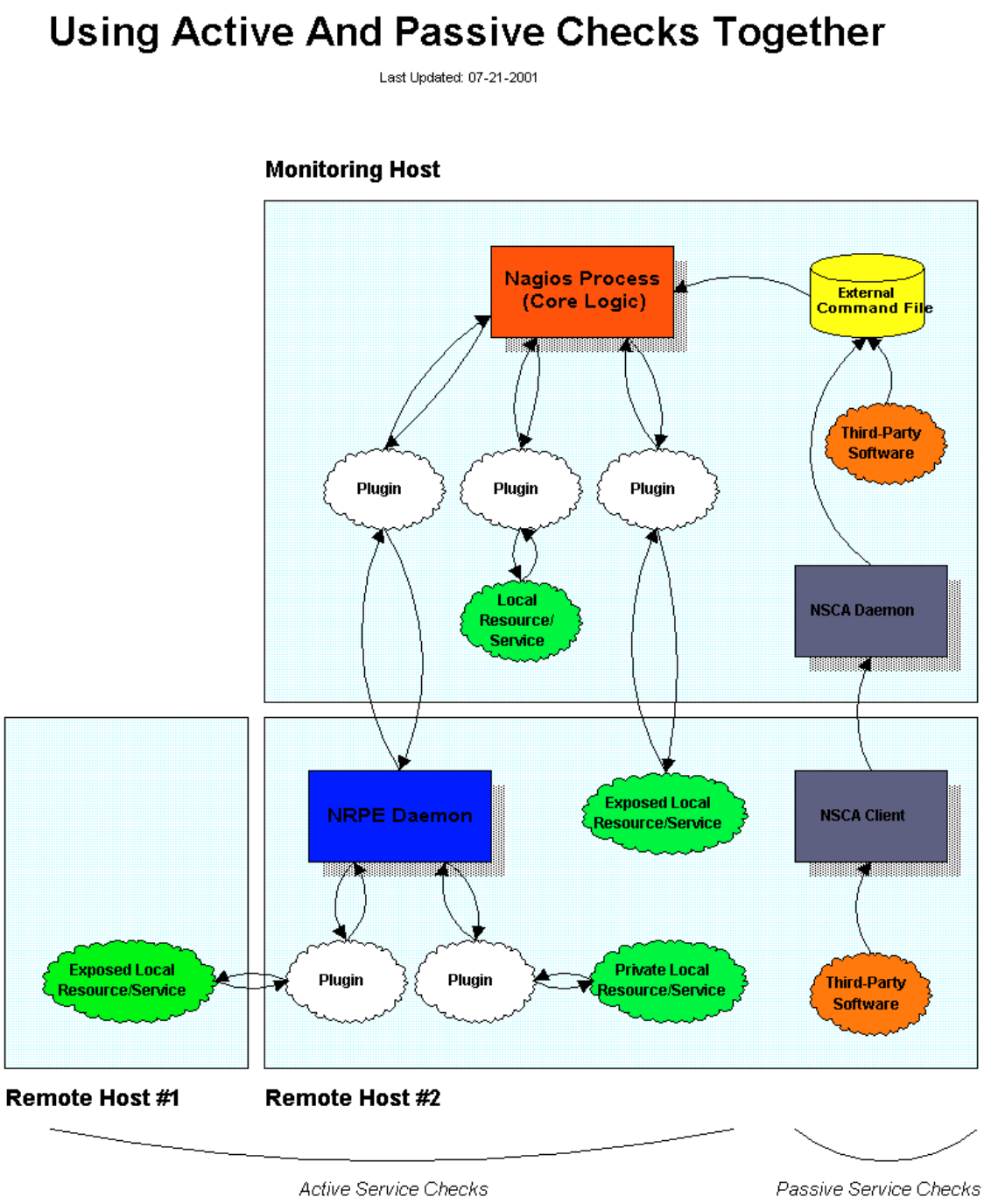


Figure 6. The mechanisms of distributed monitoring – NRPE and NSCA add-ons.

Distributed monitoring relies heavily on the interdependence between network segments, which means it is much more important to be aware of the status of a remote job. As shown in *Figure 6* above, "getting more information from a remote machine is not the same as checking to see if services are running"[14], because private local resources/services concerning the status of a particular

[14] http://www.networkcomputing.com/1109/1109sp5.html

host such as disk, memory usage, load or the number of running processes are highly sensitive data and a properly configured firewall will block remote connections. Such resources/services are different from the exposed local resources/services that are simply checked by a plugin directly from the Nagios Process (core logic) in *Figure 6.* Do not be confused by the adjective 'local' used to describe resources/services located on 'remote host #1' and 'remote host #2'. because that simply means these resources/services are considered as local from the remote point of view. Such exposed local resources/services are meant to allow remote usage. For instance, mail services like SMTP, POP3, database services like MYSQL, ORACLE and any other web connectivity services like DNS.

Therefore, in order for the Nagios monitoring server to check the private local resources/services of a remote server, secure shell (check_by_ssh) and other external addons such as Nagios Remote Plugin Executor (NRPE) for unix platforms, NRPE_NT and NSClient for windows platforms are recommended for protection of checking these sensitive system data. From *Figure 6*, the Nagios Process (core logic) executes service checks directly to the NRPE daemon using a plugin called 'check_nrpe', which "sends plugin execution requests to the nrpe agent on the remote host. The nrpe agent will then run appropriate plugins on the remote host and return the plugin output and return code to the check_nrpe plugin on the Nagios host. The check_nrpe plugin then passes the remote plugin's output and return code back to Nagios as if it were its own"[15]. The process of a Nagios monitoring server, whether it is a central or distributed, performing a check directly is called an 'active' check.

In terms of NRPE's security features, the newer version 2.0beta4 can be configured to run under Secure Sockets Layer/Transport Layer Security(SSL/TLS) and it uses AES-256 Bit Encryption using SHA and Anon-DH. One just have to configure it with '--enable-ssl' option. "*Most information these days can be twisted* into something that can live on a web page. That might be easier than managing the security implications of giving outsiders access to your internal proprietary network."[16] T*his encrypts all traffic and makes it much harder for packets to be sniffed across networks. Thus, it minimises the risk of highly sensitive data falling into the wrong hands. W*hen running as a daemon, NRPE (applies to all versions prior) *maintains* an access control list in its configuration file and only allows a connection to be made if that IP address is configured to be allowed, otherwise it will produce the following error in the logs:

July 2 13:35:17 remoteA nrpe[20712]: [ID 421412 daemon.error] Could not read request from client, bailing out...

Alternatively, NRPE can be configured to run under INETD or XINETD, which uses tcpwrappers for additional security. Tcpwrappers can be configured like a firewall to manage access control as well and deny all hosts except certain hosts to access a particular service. Hosts that are authorised could be specified in the 'hosts.allow' file and just like a firewall with a default drop policy, the 'hosts.deny' file can be set to deny all. The benefit of using wrappers is that the "connecting client is unaware that TCP wrappers are in use. Legitimate users will not notice anything different, and attackers never receive any additional information about why their attempted connections failed"[17].

[15]http://Nagios.sourceforge.net/docs/1_0/addons.html
[16]http://lists.shmoo.com/pipermail/vpn/1999-September/000223.html
[17]http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/ref-guide/ch-tcpwrappers.html

However, a weakness of NRPE is "when running in daemon mode, the nrpe agent authenticates plugin execution requests by doing a rudimentary comparison of the IP address of the calling host against a list of allowed IP addresses in the configuration file"[15]. IP addresses can always be spoofed. Even when running NRPE as a service under inetd or xinetd, although one can use tcpwrappers together with inetd to ensure that the correct remote daemon is started up, "tcpwrappers is vulnerable to IP spoofing because it uses IP addresses for host authentication"[18].

Spoofing is defined as follows:

"Spoofing is the creation of TCP/IP packets using somebody else's IP address. Routers use the 'destination IP' address in order to forward packets through the Internet, but ignore the 'source IP' address. That address is only used by the destination machine when it responds back to the source."[19]

There are a few types of spoofing attacks such as man-in-the-middle, source routing, blind spoofing, and syn flooding that result in a denial of service. For more details on spoofing, one can check out http://www.phrack.org/phrack/48/P48-14.

However, this issue is minor if SSL is enabled, as the spoofed attacker will have a tough time decrypting all the traffic even when captured.

Alternatively, it is also possible to use Simple Network Management Protocol ('check_snmp' plugin) to execute private local resource/service checks, but there is currently limited support for SNMP for Nagios. "Nagios is not designed to be a replacement from a full-blown SNMP management application like HP OpenView or OpenNMS"[20]. Moreover, SNMP itself has been well-known for its vulnerabilities especially in earlier version 1. "These vulnerabilities may cause denial-of-service conditions, service interruptions, and in some cases may allow an attacker to gain access to the affected device".[21]. Overall, anyone who sniffs a network with SNMP packets can easily read the values of systems devices and attempts a compromise because the community string names are transmitted in clear text. And, though these facts exist, "95% of devices still use it. SMNPv3 is more secure but upgrading to that is a time consuming and difficult process"[22].

As referenced in *Figure 6*, one can see "third-party software" bubbles. The setup of a Nagios distributed monitoring environment also utilises another type of external addon called Nagios Service Check Acceptor (NSCA). NSCA plays an important role when it comes to passing service check results onto another monitoring server.

[15]http://Nagios.sourceforge.net/docs/1_0/addons.html
[18]http://www.undergroundnews.com/files/texts/underground/hacking/keenveracity5.htm
[19]http://www.iss.net/security_center/advice/Underground/Hacking/Methods/Technical/Spoofing/default.html
[20]http://Nagios.sourceforge.net/docs/1_0/int-snmptrap.html
[21]http://www.cert.org/advisories/CA-2002-03.html
[22]http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci802127,00.html

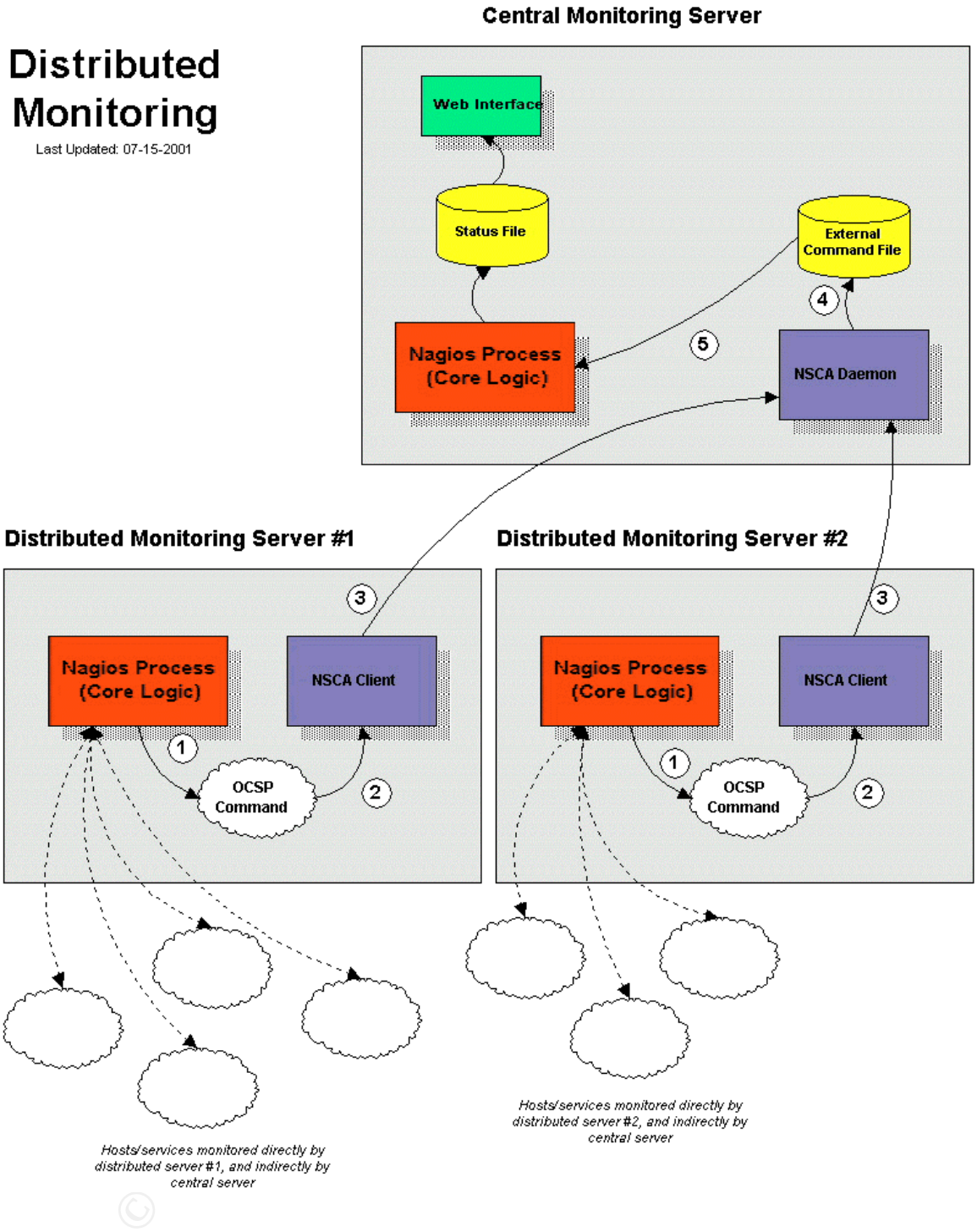


Figure 7. The mechanisms of NSCA facilitating the process of distributed monitoring[23].

NSCA works in a server-client architecture. As portrayed in *Figure* 7, when a Nagios monitoring server is configured to execute the obsessive compulsive service processor (OCSP) command, OCSP runs a 'submit_check_result' shell script that pipes the results to the NSCA client program called 'send_nsca'. The NSCA client program will then send the service check results to the NSCA

[23]http://Nagios.sourceforge.net/docs/1_0/images/distributed.png

server program installed on the central Nagios monitoring server. The NSCA server writes these service results to an external command file that acts as a temporary buffer waiting to be read by Nagios in run-time. Nagios logs all information into a status file called 'status.log' and then displays them on a user-friendly web interface, for instance *Figure 1*.

The NSCA client program also has a built-in encryption system allowing the user to set a password for encrypting all outgoing traffic and the user can choose from a variety of in-built encryption methods to choose from to encrypt the traffic going across, which is however, used at the risk of the server's processor performance. Because server and client share the same key, the password and method for decrypting the packets at the server end must be the same. Such cryto-algorithms used in NSCA as well as NRPE do provide more protection to the distributed monitoring environment by addressing the three main security issues:

"Confidentiality - no one but the intended recipients is able to intercept and understand the communication. Confidentiality is accomplished by encryption.

Authentication and Integrity - proof for the recipient that the communication was actually sent by the expected sender, and that the data has not been modified in transit. This is accomplished by authentication, often by use of cryptographic keyed hashes.

Non-repudiation - proof that the sender actually sent the data; the sender cannot later deny having sent it. Non-repudiation is usually a benign side-effect of authentication."[24]

⑩Setting up a Central Nagios Monitoring Server

In Nagios.cfg (main configuration file for overall program) :

"This is the main Nagios configuration file, containing global settings for the package. It defines directory locations for the package's various components, the user and group context for the daemon, what items to log, log file rotation settings, various time-outs and other performance-related settings, and additional items related to some of the package's advanced features (such as enabling event handling and defining global event handlers)."[2]

Values of any switch - (Enable - 1 , Disable - 0)

✍ Specifying the location of the resource.cfg.

```
resource_file=/usr/local/Nagios/etc/resource.cfg        #default file location
```

"This file defines macros that may be used within other settings for clarity and security purposes, such as to hide passwords from view in CGI programs."2 In resource.cfg file one can see how usernames and passwords can be hidden, as the actual usernames and passwords can be defined in the macros

[24]http://www.clavister.com/manuals/ver8x/manual/vpn/vpn_overview.htm
[2]http://www.onlamp.com/lpt/a/2787
[2]http://www.onlamp.com/lpt/a/2787

as shown below:

```
$USER3$=someuser
$USER4$=somepassword
```

Macros like $USER3$ and $USER4$ are used to pass arguments to built-in Nagios plugins. The paper shall demonstrate more on the usage of macros later.

✍ Stripping of illegal characters from macros and object names.

One can "specify illegal characters that should be stripped from macros before being used in notifications, event handlers, and other commands. This DOES NOT affect macros used in service or host check commands. You can choose to not strip out the characters shown in the example above, but I recommend you do not do this. Some of these characters are interpreted by the shell (i.e. the backtick) and can lead to security problems."25 For instance, on a Linux x86 platform "several system calls, such as popen(3) and system(3), are implemented by calling the command shell, meaning that they will be affected by shell metacharacters. Similarly, execlp(3) and execvp(3) may cause the shell to be called"26. Obviously, one would want to avoid allowing such potentially dangerous calls to be executed!

```
illegal_macro_output_chars= `~!$%^&*|'"<>?,()=;\[]{}\n\r\t\\v\f.
```

And, when it comes to configuring host names and service descriptions for monitoring, Nagios also practises good programming style disallowing illegal characters to be specified. One can set them in the option shown below:

```
illegal_object_name_chars=`~!$%^&*|'"<>?,()=;\[]{}\n\r\t\\v\f.
```

The paper has added more metacharacters other than the Nagios default, though there are still more types that one can implement. As one can see safe coding is applied within the Nagios development code, which ensures users that running Nagios itself would not open loopholes into network frameworks. The Nagios development tries to "ensure that any call out to another program only permits valid and expected values for every parameter"26.

✍ Specifying the location of the Nagios.tmp file.

```
temp_file=/usr/local/Nagios/var/Nagios.tmp      #default file location
```

"This is a temporary file that Nagios periodically creates to use when updating comment data, status

[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[26]http://new.linuxnow.com/docs/content/Secure-Programs-HOWTO-html/Secure-Programs-HOWTO-7.html
[26]http://new.linuxnow.com/docs/content/Secure-Programs-HOWTO-html/Secure-Programs-HOWTO-7.html

data, etc. The file is deleted when it is no longer needed.25"

✍ Specifying the location of the Nagios.lock file.

```
lock_file=/tmp/Nagios.lock          #default file location
```

"This option specifies the location of the lock file that Nagios should create when it runs as a daemon (when started with the -d command line argument). This file contains the process id (PID) number of the running Nagios process."25

✍ As a good security practice, it is advisable to enable all logging.

```
use_syslog=1
```

"This variable determines whether messages are logged to the syslog facility on your local host"25.

```
log_notifications=1
```

"This variable determines whether or not notification messages are logged. If you have a lot of contacts or regular service failures your log file will grow relatively quickly. Use this option to keep contact notifications from being logged."25

```
log_service_retries=1
```

"This variable determines whether or not service check retries are logged. Service check retries occur when a service check results in a non-OK state, but you have configured Nagios to retry the service more than once before responding to the error. Services in this situation are considered to be in "soft" states."25

```
log_host_retries=1
```

"This variable determines whether or not host check retries are logged."25

```
log_event_handlers=1
```

"This variable determines whether or not service and host event handlers are logged."25

```
log_initial_states=1
```

"This variable determines whether or not Nagios will force all initial host and service states to be logged, even if they result in an OK state. Initial service and host states are normally only logged when there is a problem on the first check. Enabling this option is useful if you are using an application that scans the log file to determine long-term state statistics for services and hosts."25

[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html

```
log_external_commands=1
```

"This variable determines whether or not Nagios will log external commands that it receives from the external command file. Note: This option does not control whether or not passive service checks (which are a type of external command) get logged. To enable or disable logging of passive checks, use the log_passive_service_checks option."25

```
log_passive_service_checks=1
```

"This variable determines whether or not Nagios will log passive service checks that it receives from the external command file."25

✍ Determine the type of log rotation one requires.

In this situation the paper has set logs to be rotated daily.

```
log_rotation_method=d
```

"This is the rotation method that you would like Nagios to use for your log file. Values are as follows:

- ✍ n = None (don't rotate the log - this is the default)
- ✍ h = Hourly (rotate the log at the top of each hour)
- ✍ d = Daily (rotate the log at midnight each day)
- ✍ w = Weekly (rotate the log at midnight on Saturday)
- ✍ m = Monthly (rotate the log at midnight on the last day of the month)"[25]

✍ Specify the location of where the log files are to be kept.

```
log_archive_path=/var/Nagios/archives          #default directory
```

This specifies where all the daily rotated logs will be stored.

```
log_file=/usr/local/Nagios/var/Nagios.log      #default file location
```

"This variable specifies where Nagios should create its main log file. This should be the first variable that you define in your configuration file, as Nagios will try to write errors that it finds in the rest of your configuration data to this file. If you have log rotation enabled, this file will automatically be rotated every hour, day, week, or month."[25]

```
status_file=/usr/local/Nagios/var/status.log   #default file location
```

"This is the file that Nagios uses to store the current status of all monitored services. The status of all hosts associated with the service you monitor are also recorded here. This file is used by the CGIs so that current monitoring status can be reported via a web interface. The CGIs must have read access to this file in order to function properly. This file is deleted every time Nagios stops and recreated

[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html

when it starts."[25]

```
comment_file=/usr/local/Nagios/var/comment.log        #default file location
```

"This is the file that Nagios will use for storing service and host comments. Comments can be viewed and added for both hosts and services through the extended information CGI."[25]

```
downtime_file=/usr/local/Nagios/var/downtime.log      #default file location
```

"This is the file that Nagios will use for storing scheduled host and service downtime information. Comments can be viewed and added for both hosts and services through the extended information CGI."[25]

✍ Specifying the location of object configuration files or directories.

One has to tell Nagios where all the rest of the configuration files that needed for monitoring are stored. The locations and structures chosen to store them are up to individual preference.

The paper has decided to use the 'object configuration directory' structure instead of 'object configuration file' so that the whole configuration work looks clearer. The 'object configuration directory' structure "is used to specify a directory which contains object configuration files that Nagios should use for monitoring. All files in the directory with a .cfg extension are processed as object config files. You can separate your configuration files into different directories and specify multiple cfg_dir= statements to have all config files in each directory processed."[25]

```
# 'object configuration directory' for storing configuration files for 'command definitions'
cfg_dir=/usr/local/Nagios/etc/generic/commands
# 'object configuration directory' for storing configuration files for 'contact definitions'
cfg_dir=/usr/local/Nagios/etc/generic/contacts
# 'object configuration directory' for storing configuration files for 'parent service definitions'
cfg_dir=/usr/local/Nagios/etc/generic/services
# 'object configuration directory' for storing local hosts to be monitored
cfg_dir=/usr/Nagios/etc/networks/localsite
# 'object configuration directory' for storing remote hosts to be monitored e.g Remote site 1 seen in #Figure 5.
cfg_dir=/usr/Nagios/etc/networks/remotesite1
```

The paper has decided to store configuration files of all hosts to be monitored in directory structures

[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html

according to their site locations. In this way, one does not need to alter the main configuration file (i.e Nagios.cfg) everytime a new host is added for monitoring.

✍ Telling Nagios to maintain all hosts and service status information even after Nagios is restarted.

retain_state_information=1

"This option determines whether or not Nagios will retain state information for hosts and services between program restarts. If you enable this option, you should supply a value for the state_retention_file variable. When enabled, Nagios will save all state information for hosts and service before it shuts down (or restarts) and will read in previously saved state information when it starts up again."[25]

use_retained_program_state=1

"This setting determines whether or not Nagios will set various program-wide state variables based on the values saved in the retention file. Some of these program-wide state variables that are normally saved across program restarts if state retention is enabled include the enable_notifications, enable_flap_detection, enable_event_handlers, execute_service_checks, and accept_passive_service_checks options. If you do not have state retention enabled, this option has no effect."25

retention_update_interval=0

"This setting determines how often (in minutes) that Nagios will automatically save retention data during normal operation. If you set this value to 0, Nagios will not save retention data at regular intervals, but it will still save retention data before shutting down or restarting. If you have disabled state retention (with the retain_state_information option), this option has no effect."[25]

state_retention_file=/usr/local/Nagios/var/status.sav

"This is the file that Nagios will use for storing service and host state information before it shuts down. When Nagios is restarted it will use the information stored in this file for setting the initial states of services and hosts before it starts monitoring anything. This file is deleted after Nagios reads in initial state information when it (re)starts. In order to make Nagios retain state information between program restarts, you must enable the retain_state_information option."[25]

It is better to enable status retention as the Nagios program needs to be restarted each time there is a modification to any configuration files. This also helps to save unnecessary CPU processor cycles on the central monitoring server. But, this option can be disabled when necessary if one wants Nagios to re-schedule all checks for processing.

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

✍ Allow the usage of event handlers.

```
enable_event_handlers=1
```

```
#global_host_event_handler=somecommand
#global_service_event_handler=somecommand
```

"This option allows you to specify a host event handler command that is to be run for every host state change. The global event handler is executed immediately prior to the event handler that you have optionally specified in each host definition."[25] The same concept applies for specifying the 'global_service_event_handler' option.

✍ Enable notifications for host and service problems.

```
enable_notifications=1
```

"This option determines whether or not Nagios will send out notifications when it initially (re)starts."[25]

✍ Disable aggressive host checking.

```
use_agressive_host_checking=0
```

"Nagios tries to be smart about how and when it checks the status of hosts. In general, disabling this option will allow Nagios to make some smarter decisions and check hosts a bit faster. Enabling this option will increase the amount of time required to check hosts, but may improve reliability a bit. Unless you have problems with Nagios not recognizing that a host recovered, I would suggest **not** enabling this option."[25] The option is configured as disabled as advised.

✍ Setting the inter_check_delay_method.

```
inter_check_delay_method=s
```

"This option allows you to control how service checks are initially "spread out" in the event queue. Using a "smart" delay calculation (the default) will cause Nagios to calculate an average check interval and spread initial checks of all services out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally *not* recommended unless you are testing the service check parallelization functionality. Using no delay will cause all service checks to be scheduled for execution at the same time. This means that you will generally have large CPU spikes when the services are all executed in parallel."[25]
"Values are as follows:

- ✍ n = Don't use any delay - schedule all service checks to run immediately (i.e. at the same time!)

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

- d = Use a "dumb" delay of 1 second between service checks
- s = Use a "smart" delay calculation to spread service checks out evenly (default)
- x.xx = Use a user-supplied inter-check delay of x.xx seconds"[25]

- Setting service_interleave_factor.

```
service_interleave_factor=s
```

"This variable determines how service checks are interleaved. Interleaving allows for a more even distribution of service checks, reduced load on *remote* hosts, and faster overall detection of host problems. With the introduction of service check parallelization, remote hosts could get bombarded with checks if interleaving was not implemented. This could cause the service checks to fail or return incorrect results if the remote host was overloaded with processing other service check requests. Setting this value to 1 is equivalent to not interleaving the service checks (this is how versions of Nagios previous to 0.0.5 worked). Set this value to **s** (smart) for automatic calculation of the interleave factor unless you have a specific reason to change it. The best way to understand how interleaving works is to watch the status CGI (detailed view) when Nagios is just starting. You should see that the service check results are spread out as they begin to appear."[25] The values for service_interleave_factor are:

- *"x* = A number greater than or equal to 1 that specifies the interleave factor to use. An interleave factor of 1 is equivalent to not interleaving the service checks.
- s = Use a "smart" interleave factor calculation (default)"[25]

One advantage of using "smart" delay calculation is that one can make use of the Nagios program to manage the load on the monitoring server and can get an overall picture of the health of Nagios by executing the following command:

```
# /usr/Nagios/bin/Nagios -s /etc/Nagios/Nagios.cfg

Nagios 1.0
Copyright (c) 1999-2002 Ethan Galstad (Nagios@Nagios.org)
Last Modified: 11-24-2002
License: GPL


        SERVICE SCHEDULING INFORMATION
        ------------------------------
        Total services:       236
        Total hosts:          34
```

[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://Nagios.sourceforge.net/docs/1_0/configmain.html

```
Command check interval:     -1 sec
Check reaper interval:      1 sec

Inter-check delay method:   SMART
Average check interval:     300.000 sec
Inter-check delay:          1.271 sec

Interleave factor method:   SMART
Average services per host:  6.941
Service interleave factor:  7

Initial service check scheduling info:
--------------------------------------
First scheduled check:      1060014867 -> Mon Aug  4 16:34:27 2003
Last scheduled check:       1060015167 -> Mon Aug  4 16:39:27 2003

Rough guidelines for max_concurrent_checks value:
-------------------------------------------------
Absolute minimum value:     1
Recommend value:            3

Notes:
The recommendations for the max_concurrent_checks value
assume that the average execution time for service
checks is less than the service check reaper interval.
The minimum value also reflects best case scenarios
where there are no problems on your network.  You will
have to tweak this value as necessary after testing.
High latency values for checks are often indicative of
the max_concurrent_checks value being set too low and/or
the service_reaper_frequency being set too high.
It is important to note that the values displayed above
do not reflect current performance information for any
```

Nagios process that may currently be running. They are provided solely to project expected and recommended values based on the current data in the config files.

✍ Determining the value of max_concurrent_checks.

```
max_concurrent_checks=
```

"This option allows you to specify the maximum number of service checks that can be run in parallel at any given time. Specifying a value of 1 for this variable essentially prevents any service checks from being parallelized. Specifying a value of 0 (the default) does not place any restrictions on the number of concurrent checks. You'll have to modify this value based on the system resources you have available on the machine that runs Nagios, as it directly affects the maximum load that will be imposed on the system (processor utilization, memory, etc.)."[25]

✍ Determining the value of service_reaper_frequency.

```
service_reaper_frequency=          #In seconds
```

"This option allows you to control the frequency *in seconds* of service "reaper" events. "Reaper" events process the results from parallelized service checks that have finished executing. These events consitute the core of the monitoring logic in Nagios."[25] Generally, this determines how fast Nagios will process the service check results and is recommended to be set below 5 seconds.

✍ Setting the sleep_time.

```
sleep_time=1
```

"This is the number of seconds that Nagios will sleep before checking to see if the next service check in the scheduling queue should be executed. Note that Nagios will only sleep after it "catches up" with queued service checks that have fallen behind."[25] The default is set as 1 and is recommended under the comments of Nagios.cfg file not to be changed.

✍ Setting the timeout values.

```
service_check_timeout=          #In seconds
```

"This is the maximum number of seconds that Nagios will allow service checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned. A timeout error will also be

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

logged."[25]

host_check_timeout=	#In seconds

"This is the maximum number of seconds that Nagios will allow host checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned and the host will be assumed to be DOWN. A timeout error will also be logged."[25]

event_handler_timeout=	#In seconds

"This is the maximum number of seconds that Nagios will allow event handlers to be run. If an event handler exceeds this time limit it will be killed and a warning will be logged."[25]

notification_timeout=	#In seconds

"This is the maximum number of seconds that Nagios will allow notification commands to be run. If a notification command exceeds this time limit it will be killed and a warning will be logged."[25]

ocsp_timeout=	#In seconds

"This is the maximum number of seconds that Nagios will allow an obsessive compulsive service processor command to be run. If a command exceeds this time limit it will be killed and a warning will be logged."[25]

perfdata_timeout=	#In seconds

"This is the maximum number of seconds that Nagios will allow a host performance data processor command or service performance data processor command to be run. If a command exceeds this time limit it will be killed and a warning will be logged."[25] In this case, this particular option is ignored as the paper has disabled 'process_performance_data option' shown in the next bullet.

It is to be noted that most of the options above starting from 'max_concurrent_checks' are left unconfigured as they are configured along with a realistic working environment (i.e. the number of hosts and services that one wants to monitor). If one is interested in knowing how the "smart" delay calculation is calculated to customize for one's network framework, one can refer to http://Nagios.sourceforge.net/docs/1_0/checkscheduling.html. Setting the timeout values really depend on individual preferences of how long Nagios should wait before killing off a particular process.

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

✍ Disable process_performance_data.

```
process_performance_data=0
#host_perfdata_command=process-host-perfdata
#service_perfdata_command=process-service-perfdata
```

The 'process_performance_data' option "determines whether or not Nagios will process host and service check performance data."[25] The paper would prefer to leave this option disabled to avoid going into too much details and focus on its aim of configuring what is necessary for the setup of a working central Nagios monitoring server.

✍ Allow for aggregate_status_updates.

```
aggregate_status_updates=1
```

"This option determines whether or not Nagios will aggregate updates of host, service, and program status data. If you do not enable this option, status data is updated every time a host or service checks occurs. This can result in high CPU loads and file I/O if you are monitoring a lot of services. If you want Nagios to only update status data (in the status file) every few seconds (as determined by the status_update_interval option), enable this option. If you want immediate updates, disable it. I would highly recommend using aggregated updates (even at short intervals) unless you have good reason not to."[25]

```
status_update_interval=5
```

"This setting determines how often (in seconds) that Nagios will update status data in the status file. The minimum update interval is five seconds. If you have disabled aggregated status updates (with the aggregate_status_updates option), this option has no effect."[25] It is recommended to set this to five seconds as the service check results are to be monitored in real-time, which implies as fast as they can be processed.

✍ Enable detection of checks that flap.

```
enable_flap_detection=1
low_service_flap_threshold=5.0
high_service_flap_threshold=20.0
low_host_flap_threshold=5.0
high_host_flap_threshold=20.0
```

"Flapping occurs when a service or host changes state too frequently, resulting in a storm of problem and recovery notifications. Flapping can be indicative of configuration problems (i.e. thresholds set too

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

low) or real network problems."[25] The global thresholds are set to default and can be changed according to real circumstances.

✍ Setup an effective user and group to run the Nagios process (default is 'Nagios').

```
Nagios_user=Nagios
Nagios_group=Nagios
```

✍ Enable execute_service_checks.

```
execute_service_checks=1
```

"This option determines whether or not Nagios will execute service checks when it initially (re)starts. If this option is disabled, Nagios will not actively execute any service checks and will remain in a sort of "sleep" mode (it can still accept passive checks unless you've disabled them). This option is most often used when configuring backup monitoring servers, as described in the documentation on redundancy, or when setting up a distributed monitoring environment. Note: If you have state retention enabled, Nagios will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the state retention file), *unless* you disable the use_retained_program_state option. If you want to change this option when state retention is active (and the use_retained_program_state is enabled), you'll have to use the appropriate external command or change it via the web interface."[25] This option is enabled as the central monitoring server performs active checking on distributed monitoring servers and its own local servers as seen in *Figure 5 for instance step 1 and step 4*.

✍ Setting the interval_length.

```
interval_length=60
```

"This is the number of seconds per "unit interval" used for timing in the scheduling queue, re-notifications, etc. "Units intervals" are used in the host configuration file to determine how often to run a service check, how often of re-notify a contact, etc.
Important: The default value for this is set to 60, which means that a "unit value" of 1 in the host configuration file will mean 60 seconds (1 minute). I have not really tested other values for this variable, so proceed at your own risk if you decide to do so!"[25]

✍ Enable passive checks received.

```
# Enable passive checks received
accept_passive_service_checks=1
```

"This option determines whether or not Nagios will accept passive service checks when it initially (re)starts. If this option is disabled, Nagios will not accept any passive service checks. Note: If you

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://nagios.sourceforge.net/docs/1_0/configmain.html

have state retention enabled, Nagios will ignore this setting when it (re)starts and use the last known setting for this option (as stored in the state retention file), *unless* you disable the use_retained_program_state option. If you want to change this option when state retention is active (and the use_retained_program_state is enabled), you'll have to use the appropriate external command or change it via the web interface."[25] This option is enabled to allow the central Nagios monitoring server to receive passive service results from the distributed monitoring servers as seen in *Figure 5 step 2*.

✍ Enable checking of external commands.

```
check_external_commands=1
```

"This option determines whether or not Nagios will check the command file for internal commands it should execute. This option must be enabled if you plan on using the command CGI to issue commands via the web interface."[25] And, in order for the central Nagios monitoring server to read the external command file written by NSCA server installed on all distributed Nagios monitoring servers, the 'check_external_commands' must be enabled shown in *Figure 7* earlier. And, the location of the command file should be checked that it exists in the right directory as shown below.

```
command_file=/var/Nagios/rw/Nagios.cmd      #default file location
```

"This is the file that Nagios will check for external commands to process. The command CGI writes commands to this file. Other third party programs can write to this file if proper file permissions have been granted as outline in here. The external command file is implemented as a named pipe (FIFO), which is created when Nagios starts and removed when it shuts down. If the file exists when Nagios starts, the Nagios process will terminate with an error message."[25] It is shown in *Figure 7*.

✍ Disable obsess_over_services.

```
obsess_over_services=0
#ocsp_command=somecommand
```

"This value determines whether or not Nagios will "obsess" over service checks results and run the obsessive compulsive service processor command you define."[25] Obsess_over_service is disabled because it is only used when one Nagios monitoring server needs to submit check results to another Nagios monitoring server. According to *Figure 5*, there is no need for the central monitoring server to operate this way.

✍ Enable checking of orphaned service checks.

```
check_for_orphaned_services=1
```

"This option allows you to enable or disable checks for orphaned service checks. Orphaned service

[25]http://nagios.sourceforge.net/docs/1_0/configmain.html
[25]http://nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

checks are checks which have been executed and have been removed from the event queue, but have not had any results reported in a long time. Since no results have come back in for the service, it is not rescheduled in the event queue. This can cause service checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a service check. If this option is enabled and Nagios finds that results for a particular service check have not come back, it will log an error message and reschedule the service check. If you start seeing service checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned services."[25] The paper has enabled this option, as it is always better to be panaroid and log error messages if service checks failed for some reason.

✍ Enable checking of passive services freshness.

```
check_service_freshness=1
freshness_check_interval=300            # in seconds
```

"This option determines whether or not Nagios will periodically check the "freshness" of service checks. Enabling this option is useful for helping to ensure that passive service checks are received in a timely manner."[25] Checking freshness is normally associated with passive checks. In the event that the passive results are not reported back by the distributed monitoring servers for a period of time specified in the freshness_check_interval option in Nagios.cfg, the central monitoring server will conduct active checks on these remote hosts immediately. Nagios will show the checking of stale services in the Nagios.log, for instance:

```
# grep "stale" /usr/local/Nagios/var/Nagios.log | perl -pe 's/(\d+)/localtime($1)/e'
[Sun July  23 23:55:19 2003] Warning: The results of service 'SSH' on host 'RemoteHostA' are stale by 32 seconds (threshold=300 seconds).  I'm forcing an immediate check of the service.
```

✍ Setting the date format.

```
date_format=
```

"This option allows you to specify what kind of date/time format Nagios should use in the web interface and date/time macros."[25] One can choose between a variety of either us (MM/DD/YYYY HH:MM:SS), euro (DD/MM/YYYY HH:MM:SS), iso8601 (YYYY-MM-DD HH:MM:SS) or strict-iso8601 (YYYY-MM-DDTHH:MM:SS). The paper shall leave this option blank as the date format chosen for logging check results is really up to individual preference.

[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html
[25] http://Nagios.sourceforge.net/docs/1_0/configmain.html

✍ Setting the details of notification.

```
admin_email=
admin_pager=
```

Nagios can be set in the above to notify to a particular email address or even to notify through smart messaging service (SMS) using a phone line whenever there is a change of state depending on how one configures notifications, which the paper shall demonstrate later in the section "Setting up the Monitoring of Remote Servers For Security". For more information on how to configure Nagios to send notifications via SMS, one can refer to http://www.Nagios.org/faqs/viewfaq.php?faq_id=65.

Configuration of resource.cfg Configuration file.

"This file defines macros that may be used within other settings for clarity and security purposes, such as to hide passwords from view in CGI programs."[2]

In resource.cfg edit the locations of the macros as needed in our case of the central Nagios monitoring server setup. Also, assuming that 192.168.10.7 is the database host in *Figure 5*, specify the database host and port to log information to:

```
##############################################################################
#
# RESOURCE.CFG - Sample Resource File for Nagios @VERSION@
#
# You can define $USERx$ macros in this file, which can in turn be used
# in command definitions in your host config file(s).  $USERx$ macros are
# useful for storing sensitive information such as usernames, passwords,
# etc.  They are also handy for specifying the path to plugins and
# event handlers - if you decide to move the plugins or event handlers to
# a different directory in the future, you can just update one or two
# $USERx$ macros, instead of modifying a lot of command definitions.
#
# The CGIs will not attempt to read the contents of resource files, so
# you can set restrictive permissions (600 or 660) on them.
#
# Nagios supports up to 32 $USERx$ macros ($USER1$ through $USER32$)
#
```

[2] http://www.onlamp.com/lpt/a/2787

```
# Resource files may also be used to store configuration directives for
# external data sources like MySQL...
#
##############################################################################

# Sets $USER1$ to be the path to the plugins
$USER1$=/usr/local/Nagios/libexec

# Sets $USER2$ to be the path to event handlers
$USER2$=/usr/local/Nagios/libexec/eventhandlers

# Store some usernames and passwords (hidden from the CGIs)
#$USER3$=someuser
#$USER4$=somepassword

# DB STATUS DATA
# Note: These config directives are only used if you compiled
# in database support for status data!
# The user you specify here needs SELECT, INSERT, UPDATE, and
# DELETE privileges on the 'programstatus', 'hoststatus',
# and 'servicestatus' tables in the database.

xsddb_host=192.168.10.7
xsddb_port=3306                 # default mysql port
xsddb_database=Nagios           # database 'Nagios'
xsddb_username=someuser         # This is unset, please put in your own username
xsddb_password=somepassword # This is unset, please put in your own secure password!
xsddb_optimize_data=1
xsddb_optimize_interval=3600

# DB COMMENT DATA
# Note: These config directives are only used if you compiled
# in database support for comment data!
```

```
# The user you specify here needs SELECT, INSERT, UPDATE, and
# DELETE privileges on the 'hostcomments' and 'servicecomments'
# tables in the database.

xcddb_host=192.168.10.7
xcddb_port=3306                 # default mysql port
xcddb_database=Nagios           # database 'Nagios'
xcddb_username=someuser         # This is unset, please put in your own username
xcddb_password=somepassword # This is unset, please put in your own secure password!
xcddb_optimize_data=1

# DB DOWNTIME DATA
# Note: These config directives are only used if you compiled
# in database support for downtime data!
# The user you specify here needs SELECT, INSERT, UPDATE, and
# DELETE privileges on the 'hostdowntime' and 'servicedowntime'
# tables in the database.

xdddb_host=192.168.10.7
xdddb_port=3306                 # default mysql port
xdddb_database=Nagios           # database 'Nagios'
xdddb_username=someuser         # This is unset, please put in your own username
xdddb_password=somepassword # This is unset, please put in your own secure password!
xdddb_optimize_data=1

# DB RETENTION DATA
# Note: These config directives are only used if you compiled
# in database support for retention data!
# The user you specify here needs SELECT, INSERT, UPDATE, and
# DELETE privileges on the 'programretention', 'hostretention',
# and 'serviceretention' tables in the database.
xrddb_host=192.168.10.7
xrddb_port=3306                 # default mysql port
xrddb_database=Nagios           # database 'Nagios'
```

```
xrddb_username=someuser # This is unset, please put in your own username
xrddb_password=somepassword # This is unset, please put in your own secure password!
xrddb_optimize_data=1
```

Configuration of cgi.cfg Configuration file.

"This file holds settings related to the Nagios displays, including paths to Web page items and scripts, and per-item icon and sound selections. The file also defines allowed access to Nagios's data and commands."[2] Assuming that the central Nagios monitoring server (i.e. 192.168.10.8) is where the administrative web interface like *Figure 1* will be held with the url: http://192.168.10.8/Nagios , in cgi.cfg file edit the necessary paths and add in a user (for instance 'Nagios') to allow access:

```
##############################################################################
#
# CGI.CFG - Sample CGI Configuration File for Nagios @VERSION@
#
# Last Modified: 10-29-2002
#
##############################################################################
# MAIN CONFIGURATION FILE
# This tells the CGIs where to find your main configuration file.
# The CGIs will read the main and host config files for any other
# data they might need.

main_config_file=/usr/local/Nagios/etc/Nagios.cfg

# PHYSICAL HTML PATH
# This is the path where the HTML files for Nagios reside.  This
# value is used to locate the logo images needed by the statusmap
# and statuswrl CGIs.

physical_html_path=/usr/local/Nagios/share

# URL HTML PATH
```

[2] http://www.onlamp.com/lpt/a/2787

```
# This is the path portion of the URL that corresponds to the
# physical location of the Nagios HTML files (as defined above).
# This value is used by the CGIs to locate the online documentation
# and graphics.  If you access the Nagios pages with an URL like
# http://www.myhost.com/Nagios, this value should be '/Nagios'
# (without the quotes).

url_html_path=/Nagios

# CONTEXT-SENSITIVE HELP
# This option determines whether or not a context-sensitive
# help icon will be displayed for most of the CGIs.
# Values: 0 = disables context-sensitive help
#         1 = enables context-sensitive help

show_context_help=0

# Nagios PROCESS CHECK COMMAND
# This is the full path and filename of the program used to check
# the status of the Nagios process.  It is used only by the CGIs
# and is completely optional.  However, if you don't use it, you'll
# see warning messages in the CGIs about the Nagios process
# not running and you won't be able to execute any commands from
# the web interface.  The program should follow the same rules
# as plugins; the return codes are the same as for the plugins,
# it should have timeout protection, it should output something
# to STDIO, etc.
#
# Note: If you are using the check_Nagios plugin here, the first
# argument should be the physical path to the status log, the
# second argument is the number of minutes that the status log
# contents should be "fresher" than, and the third argument is the
# string that should be matched from the output of the 'ps'
# command in order to locate the running Nagios process.  That
```

```
# process string is going to vary depending on how you start
# Nagios.  Run the 'ps' command manually to see what the command
# line entry for the Nagios process looks like.

#Nagios_check_command=@libexecdir@/check_Nagios @localstatedir@/status.log 5
'@bindir@/Nagios'

# AUTHENTICATION USAGE
# This option controls whether or not the CGIs will use any
# authentication when displaying host and service information, as
# well as committing commands to Nagios for processing.
#
# Read the HTML documentation to learn how the authorization works!
#
# NOTE: It is a really *bad* idea to disable authorization, unless
# you plan on removing the command CGI (cmd.cgi)!  Failure to do
# so will leave you wide open to kiddies messing with Nagios and
# possibly hitting you with a denial of service attack by filling up
# your drive by continuously writing to your command file!
#
# Setting this value to 0 will cause the CGIs to *not* use
# authentication (bad idea), while any other value will make them
# use the authentication functions (the default).

use_authentication=1

# DEFAULT USER
# Setting this variable will define a default user name that can
# access pages without authentication.  This allows people within a
# secure domain (i.e., behind a firewall) to see the current status
# without authenticating.  You may want to use this to avoid basic
# authentication if you are not using a sercure server since basic
# authentication transmits passwords in the clear.
#
# Important:  Do not define a default username unless you are
```

```
# running a secure web server and are sure that everyone who has
# access to the CGIs has been authenticated in some manner!  If you
# define this variable, anyone who has not authenticated to the web
# server will inherit all rights you assign to this user!

default_user_name=Nagios
# SYSTEM/PROCESS INFORMATION ACCESS
# This option is a comma-delimited list of all usernames that
# have access to viewing the Nagios process information as
# provided by the Extended Information CGI (extinfo.cgi).  By
# default, *no one* has access to this unless you choose to
# not use authorization.  You may use an asterisk (*) to
# authorize any user who has authenticated to the web server.

#authorized_for_system_information=Nagiosadmin,theboss,jdoe

# CONFIGURATION INFORMATION ACCESS
# This option is a comma-delimited list of all usernames that
# can view ALL configuration information (hosts, commands, etc).
# By default, users can only view configuration information
# for the hosts and services they are contacts for. You may use
# an asterisk (*) to authorize any user who has authenticated
# to the web server.

#authorized_for_configuration_information=Nagiosadmin,jdoe

# SYSTEM/PROCESS COMMAND ACCESS
# This option is a comma-delimited list of all usernames that
# can issue shutdown and restart commands to Nagios via the
# command CGI (cmd.cgi).  Users in this list can also change
# the program mode to active or standby. By default, *no one*
# has access to this unless you choose to not use authorization.
# You may use an asterisk (*) to authorize any user who has
# authenticated to the web server.
```

```
#authorized_for_system_commands=Nagiosadmin

# GLOBAL HOST/SERVICE VIEW ACCESS
# These two options are comma-delimited lists of all usernames that
# can view information for all hosts and services that are being
# monitored.  By default, users can only view information
# for hosts or services that they are contacts for (unless you
# you choose to not use authorization). You may use an asterisk (*)
# to authorize any user who has authenticated to the web server.

#authorized_for_all_services=Nagiosadmin,guest
#authorized_for_all_hosts=Nagiosadmin,guest


# GLOBAL HOST/SERVICE COMMAND ACCESS
# These two options are comma-delimited lists of all usernames that
# can issue host or service related commands via the command
# CGI (cmd.cgi) for all hosts and services that are being monitored.
# By default, users can only issue commands for hosts or services
# that they are contacts for (unless you you choose to not use
# authorization).  You may use an asterisk (*) to authorize any
# user who has authenticated to the web server.

#authorized_for_all_service_commands=Nagiosadmin
#authorized_for_all_host_commands=Nagiosadmin

# EXTENDED HOST INFORMATION
# This is all entirely optional.  If you don't enter any extended
# information, nothing bad will happen - I promise...  Its basically
# just used to have pretty icons and such associated with your hosts.
# This is especially nice when you're using the statusmap and
# statuswrl CGIs.  You can also specify an URL that links to a document
# containing more information about the host (location details, contact
```

```
# information, etc).
#
# hostextinfo[<host_name>]=<notes_url>;<icon_image>;<vrml_image>;<gd2_image>;\
#                       <image_alt>;<x_2d>,<y_2d>;<x_3d>,<y_3d>,<z_3d>;
#
# <notes_url>       = Optional URL that points to a document of
#                     some type containing information on the host.
#                     The information (and the document type) can
#                     be anything you want.  Examples include details
#                     on the physical location of the server, info
#                     on how to contact the admins in case of an
#                     emergency, etc.  Relative URLs start in the
#                     same path that is used to access the CGIs.
#                     The link that is created for the host's notes
#                     notes is found in the extinfo CGI.
#                     Note: You may use the $HOSTNAME$ and
#                     $HOSTADDRESS$ macros in this URL.
# <icon_image>      = A GIF, PNG, or JPEG image to associate with
#                     the host.  This is used in the status and
#                     extinfo CGIs.
# <vrml_image>      = An image to use in the statuswrl CGI in the
#                     VRML generation.  Transparent images don't
#                     work so great..
# <gd2_image>       = An image used by the statusmap CGI to
#                     represent the host.  This can be a GIF, PNG,
#                     JPEG, or GD2 image.  GD2 format is recommended,
#                     as it produces the load CPU load.
#                     utility supplied with Boutell's gd library.
# <image_alt>       = ALT tag used with images in various CGIs
# <x_2d>,<y_2d>     = X and Y coordinates used when drawing the
#                     host in the statusmap CGI.  (0,0) is located
#                     in the upper left corner of the screen and is
#                     considered to be the origin.  The coordinates
#                     you supply here are used as the coords of the
```

```
#               upper left hand corner of host icon.  Both
#               numbers should be positive integers.
# <x_3d>,<y_3d>,<z_3d> = X, Y, and Z coordinates used when drawing
#               the host in the statuswrl (VRML) CGI.  All
#               numbers can be positive or negative (anywhere
#               in 3-D space).  The coordinates are used to
#               determine the center of the host "cube" that
#               is drawn. Host "cubes" are drawn with a
#               height, width, and depth of 0.5 (meters).
#
# Note: All images must be placed in the /logos subdirectory under
# the HTML images path (i.e. /usr/local/Nagios/share/images/logos/).
# This path is automatically determined by appending "/images/logos"
# to the path specified by the 'physical_html_path' directive.

#hostextinfo[es-eds]=/serverinfo/es-eds.html;novell40.gif;novell40.jpg;novell40.gd2;IntranetWare
4.11;100,50;3.5,0.0,-1.5;
#hostextinfo[rosie]=/serverinfo/rosie.html;win40.gif;win40.jpg;win40.gd2;NT Server 4.0;;;


# EXTENDED SERVICE INFORMATION
# This is all entirely optional.  If you don't enter any extended
# information, nothing bad will happen - I promise...  Its basically
# just used to have pretty icons and such associated with your services.
# You can also specify an URL that links to a document containing more
# information about the service (location details, contact information,
# etc).
#
# serviceextinfo[<host_name>;<svc_description>]=<notes_url>;<icon_image>;<image_alt>
#
# <notes_url>       = Optional URL that points to a document of
#               some type containing information on the service.
#               The information (and the document type) can
#               be anything you want.  Examples include details
#               on the physical location of the server, info
#               on how to contact the admins in case of an
```

```
#                   emergency, etc.  Relative URLs start in the
#                   same path that is used to access the CGIs.
#                   The link that is created for the service's
#                   notes URL is found in the extinfo CGI.
#                   Note: You may use the $HOSTNAME$, $HOSTADDRESS$,
#                   and $SERVICEDESC$ macros in this URL.
# <icon_image>      = A GIF, PNG, or JPEG image to associate with
#                   the service.  This is used in the status and
#                   extinfo CGIs.
# <image_alt>       = ALT tag used with image
#
# Note: All images must be placed in the /logos subdirectory under
# the HTML images path (i.e. /usr/local/Nagios/share/images/logos/).
# This path is automatically determined by appending "/images/logos"
# to the path specified by the 'physical_html_path' directive.

#serviceextinfo[es-eds;PING]=http://www.somewhere.com?traceroute to=$HOSTADDRESS$;;PING rate
#serviceextinfo[rosie;Security Alerts]=;security.gif;Security alerts


# STATUSMAP BACKGROUND IMAGE
# This option allows you to specify an image to be used as a
# background in the statusmap CGI.  It is assumed that the image
# resides in the HTML images path (i.e. /usr/local/Nagios/share/images).
# This path is automatically determined by appending "/images"
# to the path specified by the 'physical_html_path' directive.
# Note:  The image file may be in GIF, PNG, JPEG, or GD2 format.
# However, I recommend that you convert your image to GD2 format
# (uncompressed), as this will cause less CPU load when the CGI
# generates the image.

#statusmap_background_image=smbackground.gd2


# DEFAULT STATUSMAP LAYOUT METHOD
# This option allows you to specify the default layout method
```

```
# the statusmap CGI should use for drawing hosts.  If you do
# not use this option, the default is to use user-defined
# coordinates.  Valid options are as follows:
#       0 = User-defined coordinates
#       1 = Depth layers
#       2 = Collapsed tree
#       3 = Balanced tree
#       4 = Circular
#       5 = Circular (Marked Up)

default_statusmap_layout=5

# DEFAULT STATUSWRL LAYOUT METHOD
# This option allows you to specify the default layout method
# the statuswrl (VRML) CGI should use for drawing hosts.  If you
# do not use this option, the default is to use user-defined
# coordinates.  Valid options are as follows:
#       0 = User-defined coordinates
#       2 = Collapsed tree
#       3 = Balanced tree
#       4 = Circular

default_statuswrl_layout=4

# STATUSWRL INCLUDE
# This option allows you to include your own objects in the
# generated VRML world.  It is assumed that the file
# resides in the HTML path (i.e. /usr/local/Nagios/share).

#statuswrl_include=myworld.wrl

# PING SYNTAX
# This option determines what syntax should be used when
# attempting to ping a host from the WAP interface (using
```

```
# the statuswml CGI.  You must include the full path to
# the ping binary, along with all required options.  The
# $HOSTADDRESS$ macro is substituted with the address of
# the host before the command is executed.
ping_syntax=/bin/ping -n -U -c 5 $HOSTADDRESS$

# REFRESH RATE
# This option allows you to specify the refresh rate in seconds
# of various CGIs (status, statusmap, extinfo, and outages).

refresh_rate=90

# SOUND OPTIONS
# These options allow you to specify an optional audio file
# that should be played in your browser window when there are
# problems on the network.  The audio files are used only in
# the status CGI.  Only the sound for the most critical problem
# will be played.  Order of importance (higher to lower) is as
# follows: unreachable hosts, down hosts, critical services,
# warning services, and unknown services. If there are no
# visible problems, the sound file optionally specified by
# 'normal_sound' variable will be played.
#
#
# <varname>=<sound_file>
#
# Note: All audio files must be placed in the /media subdirectory
# under the HTML path (i.e. /usr/local/Nagios/share/media/).

#host_unreachable_sound=hostdown.wav
#host_down_sound=hostdown.wav
#service_critical_sound=critical.wav
#service_warning_sound=warning.wav
#service_unknown_sound=warning.wav
```

```
#normal_sound=noproblem.wav

# DG EXTENDED DATA
# Note: These config directives are only used if you compiled
# in database support for extended data!
# The user you specify here only needs SELECT privileges on the
# 'hostextinfo' table in the database.

#xeddb_host=somehost
#xeddb_port=someport
#xeddb_database=somedatabase
#xeddb_username=someuser
#xeddb_password=somepassword

# DB STATUS DATA (Read-Only For CGIs)
# Note: These config directives are only used if you compiled
# in database support for status data!
# The user you specify here only needs SELECT privileges on the
# 'programstatus', 'hoststatus', and 'servicestatus' tables
# in the database, as these values are only used by the CGIs.
# The core program will read the directives you specify in
# in a resource file.

#xsddb_host=somehost
#xsddb_port=someport
#xsddb_database=somedatabase
#xsddb_username=someuser
#xsddb_password=somepassword

# DB COMMENT DATA (Read-Only For CGIs)
# Note: These config directives are only used if you compiled
# in database support for comment data!
# The user you specify here only needs SELECT privileges on the
# 'hostcomments', and 'servicecomments' tables in the database,
```

```
# as these values are only used by the CGIs.  The core program
# will read the directives you specify in a resource file.

#xcddb_host=somehost
#xcddb_port=someport
#xcddb_database=somedatabase
#xcddb_username=someuser
#xcddb_password=somepassword

# DB DOWNTIME DATA (Read-Only For CGIs)
# Note: These config directives are only used if you compiled
# in database support for downtime data!
# The user you specify here only needs SELECT privileges on the
# 'hostdowntime', and 'servicedowntime' tables in the database,
# as these values are only used by the CGIs.  The core program
# will read the directives you specify in a resource file.

#xdddb_host=somehost
#xdddb_port=someport
#xdddb_database=somedatabase
#xdddb_username=someuser
#xdddb_password=somepassword
```

From the above cgi.cfg file, one could see that Nagios implement strict access to the type of users allowed in accessing the cgis giving security analysts even more assurance that a Nagios deployment will not pose a threat to their company resources but actually take care of the confidentiality of information. Also, because Nagios makes use of cgi scripts found in /usr/local/Nagios/share to generate trends, availability and so forth as shown in *Figure 1*, it has also ensured to protect the confidentiality of company resources by fixing "a single directory, such as `cgi-bin`, for serving CGI scripts only, and abide by it."[27] This is a good practice as that 'cgi-bin' directory resides within the web directory and nowhere else. It can serve as a form of restriction to limit the type of systems data that an attacker can get to even when his malicious cgi script attack works. In other words, it resembles a secure chroot environment except that in this case, the attacker is aware that he cannot gain access to root directory at all.

[27] http://tech.irt.org/articles/js184/

Configuration of Object Configuration files.

Object Configuration files are defined as follows:

"This class of files specifies which hosts and services are monitored. In addition, they can be used to define host and service test commands, host groups, alerts and their recipients, event handlers, and other object-specific settings used by Nagios."[2]

By default, Nagios comes together with a set of sample object configuration files. They include the hostgroups.cfg-sample, hosts.cfg-sample, misccommands.cfg-sample, checkcommands.cfg-sample, contactgroups.cfg-sample, contacts.cfg-sample, dependencies.cfg-sample, services.cfg-sample, escalations.cfg-sample, timeperiods.cfg-sample and minimal.cfg-sample. These files are just named according to the definitions they store and can be changed according to individual preference. The definitions stored consists of the following items:

"Hosts: Computers and other network devices

Host Groups: Named groups of hosts

Services: Important daemons providing specific network services

Contacts: User to be contacted in the event of a problem

Contact Groups: Named groups of contacts

Time Periods: Day and/or time ranges within a week, used to specify when checks are to be performed, notifications are to be sent, and the like

Commands: Commands to be run for all purposes (host/service checking, notifications, event handling, and so on). Nagios provides two files containing many predefined commands: checkcommands.cfg and misccommands.cfg.

Host Dependencies: Specifications of host reachability dependencies. When an intermediate host is down, checks are skipped for all hosts that are dependent on that one.

Service Dependencies: Specifications of service dependency requirements. When a service host is down, checks are skipped for all other services that are dependent on it.

Host Escalations: Definitions of optional escalation levels for host problems

Host Group Escalations: Definitions of optional escalation levels for host groups

Service Escalations: Definitions of optional escalation levels for failed services"[2]

The dependencies.cfg-sample and escalations.cfg-sample configuration files that store the host and services dependencies and escalations definitions respectively are not a necessity for configuration, and Nagios will still be able to start running without them, as long as the rest of the items specified above are defined. Because there are quite a few files to begin configuration with, Nagios has provided the minimal.cfg-sample for new users to start with. That file combines a backbone of all the definitions used in configuring Nagios including hosts, services, contacts, and so forth.

One main advantage of the improved Nagios over traditional Netsaint is that it makes use of template-

[2] http://www.onlamp.com/lpt/a/2787

[2]http://www.onlamp.com/lpt/a/2787

based object data, which makes it easier for configuration! Templates work like the concept of 'inheritance', where child objects can share similar features of the parent object defined without having to repeat the specifications of these features again in the child objects. In other words, a child object inherits all features from its parent object that are defined.

In minimal.cfg-sample, for instance one may see the following host definition templates:

```
# This is the parent host definition
define host {
        name                            generic-host    ; The name of this host template
        notifications_enabled           1       ; Host notifications are enabled
        event_handler_enabled           1       ; Host event handler is enabled
        flap_detection_enabled          1       ; Flap detection is enabled
        failure_prediction_enabled      1       ; Failure prediction is enabled
        process_perf_data               1       ; Process performance data
        retain_status_information       1       ; Retain status information across program resta
rts
        retain_nonstatus_information    1       ; Retain non-status information across program restarts
        register                        0       ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL
 HOST, JUST A TEMPLATE!
        }



# Since this is a simple configuration file, we only monitor one host - the
# local host (this machine).
# This is the child host definition.
define host{
        use                             generic-host            ; Name of host template to use
        host_name                       localhost
        alias                           localhost
        address                         127.0.0.1
        check_command                   check-host-alive
        max_check_attempts              10
        notification_interval           120
        notification_period             24x7
        notification_options            d,r
        }
```

The above shows the flexibility of template-based object data, where the parent object is called 'generic-host' with its child object inheriting all of its features through the key option 'use'.

The paper stores object configuration files in self-customizable directory structures which in this case are set according to those directories as specified in Nagios.cfg file shown earlier. Now, the paper shall demonstrate the contents of the files located in each of the directory.

✍ In /usr/local/Nagios/etc/generic/services/ directory:

To begin configuration, it is noted that in order for all the set up of these object configuration files to work in run-time, one has to set up the configuration necessary for the remote end servers as well, which will be demonstrated in the later section of "Setting Up Distributed Nagios Monitoring Server for Remote Site 1". In addition, other than having service definitions defined for checking servers on the local site seen in *Figure 5*, "the central server must have service definitions for all services that are being monitored by all the distributed servers. Nagios will ignore passive check results if they do not correspond to a service that has been defined"[27]. And, because of that the services to be configured for monitoring on the central Nagios monitoring server are neither all passive nor active, enabling the overall switches in main configuration file Nagios.cfg seen earlier is insufficient to determine which services belong to the passive and active group. There is a need to specify options including active_checks_enabled, passive_checks_enabled, parallelize_check, obsess_over_service and check_freshness for each service definition even though they are not compulsory options. The paper has named a configuration file e.g. 'parent-services.cfg' to be added within this directory.

The 'parents-services.cfg' will contain parent templates called 'active-service' and 'passive-service' storing unregistered template for inheritance so as to save the need for disabling and enabling these options for each service definition. Options including max_check_attempts, normal_check_interval, retry_check_interval, check_period, notification_interval, and so forth are also specified in the parent templates. Because these options are general and compulsory options to all services, specifying them in the parent templates also save the need to configure them for each and every single service definition. This not only saves time in configuration but minimises human errors.

In 'parent-services.cfg' file:

```
#Parent template 1 -- active-service (inherited by services that are actively checked by central
#monitoring server)
define service{
        name                    active-service          ; The 'name' of this service template,
referenced in other service definitions
        active_checks_enabled   1                       ; Active service checks are enabled
        passive_checks_enabled  0                       ; Passive service checks are enabled/accepted
        parallelize_check       1                       ; Active service checks should be parallelized
(disabling this can lead to major performance problems)
        obsess_over_service     0                       ; We should not obsess over this service (if
```

[27] http://nagios.sourceforge.net/docs/1_0/distributed.html

```
necessary)
        check_freshness         0               ; Active service so no need to check service 'freshness'
        max_check_attempts      1               ; Service is checked 1 time
        normal_check_interval   5               ; Service is checked every 5 minutes
        retry_check_interval3                   ; Service is re-checked every 3 minutes
        check_period        24x7                ; 24 hours by 7 days checked
        notification_interval   0               ; Services not acknowledged is notified only once, remove duplicates of notifications reporting the same type of error for a particular process.
        notification_period     24x7            ; 24 hours by 7 days notified.
        notification_options    u,c,r           ;  states allowed for notifications.
        contact_groups          admin           ; contact us


        register                0               ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!
}
```

Parallelized checks are needed in active checks because the monitoring server is to be able to run a few checks at one go to minimise load / reduce latency. Obsess_over_service is disabled because it is only used when the central monitoring server is to submit service results to another monitoring server. Checking freshness is normally associated with passive checks explained later in parent template 2, so it is disabled as well.

The normal_check_interval is related to the interval_length specified in Nagios.cfg. If the interval_length (in seconds) is set to 60, it will mean in this case that the central monitoring server will initially schedule a service to be checked every 5 minutes. And, when the first service check executed returns a change in state, it is checked again by the number of times specified in the max_check_attempts at an interval specified in retry_check_interval before Nagios will actually confirm the change. Depending on the notification_options; warning (w), unknown (u), critical (c) and recovery (r) specified, it will then inform the point-of-contact (i.e. contact_groups) every 10 minutes about the service until the service recovers. When the service recovers, Nagios will also inform the point-of-contact because (r) is specified. The template is not registered because it is not an actual service to be monitored.

Within the same 'parent-services.cfg' file:

```
#Parent template 2 -- passive-service (inherited by services that are actively monitored by distributed #monitoring servers and not actively checked by central monitoring server)
define service{
```

```
        name                    passive-service     ; The 'name' of this service template, referenced in other service definitions
        active_checks_enabled   1                   ; Active service checks are enabled so that passive services still get alerted (i.e. sound is not disabled) on the web interface but they are not actually actively checked as defined in the check_period option later.
        passive_checks_enabled  1                   ; Passive service checks are accepted.
        parallelize_check       0                   ; There are no actual active service checks. (disabling this can lead to major performance problems)
        obsess_over_service     0                   ; We should not obsess over this service (if necessary)
        check_freshness         1                   ; Check service 'freshness' is enabled.
        max_check_attempts      0                   ; Not checked because check_period is set to none
        normal_check_interval   0                   ;
        retry_check_interval0                   ;
        check_period            none                ; look at 'none' time definition found in 'timeperiods.cfg' file
        notification_interval   0                   ; Services not acknowledged is notified only once, remove duplicates of notifications reporting the same type of error for a particular process.
        notification_period     24x7                ; 24 hours by 7 days notified.
        notification_options    u,c,r               ;  states allowed for notifications.
        contact_groups          admin               ; contact us
        register                0                   ; DONT REGISTER THIS DEFINITION - ITS NOT A REAL SERVICE, JUST A TEMPLATE!
}
```

Note that in template 2 the active_checks_enabled is disabled and passive_checks_enabled is enabled, indicating that the services that inherit these are all services that are not checked actively by the central monitoring server. Also, "processing of service check results has not been parallelized. This has been done to prevent situations where multiple notifications about host problems or recoveries may be sent out if a host goes down, becomes unreachable, or recovers"[28].

One might ponder about the above mentioned options and wonder why certain options like 'max_check_attempts' and 'normal_check_interval' exist in the file when they are disabled anyway. Or how does one know what are the options to specify in the first place? All these boil down to finding out the minimal requirements and how does a full service definition template look like. And, it is as shown below:

```
# A full skeleton of a service definition template.
# Note: This is not included in the /usr/local/Nagios/etc/generic/services/ directory.
```

[28] http://nagios.sourceforge.net/docs/1_0/parallelization.html

```
define service{
        host_name                       host_name
        service_description             service_description
        is_volatile[0/1]
        check_command                   command_name
        max_check_attempts              #
        normal_check_interval           #
        retry_check_interval            #
        active_checks_enabled[0/1]
        passive_checks_enabled[0/1]
        check_period                    timeperiod_name
        parallelize_check[0/1]
        obsess_over_service[0/1]
        check_freshness[0/1]
        freshness_threshold             #
        event_handler                   command_name
        event_handler_enabled[0/1]
        low_flap_threshold              #
        high_flap_threshold             #
        flap_detection_enabled[0/1][1]
        process_perf_data[0/1]
        retain_status_information[0/1]
        retain_nonstatus_information[0/1]
        notification_interval           #
        notification_period             timeperiod_name
        notification_options[w,u,c,r]
        notifications_enabled[0/1]
        contact_groups                  contact_groups
        stalking_options[o,w,u,c]
}[29]
```

It is to be noted that those options in italics are the minimal requirements of a service definition template and part of them are stated in the parent-services.cfg file to be inherited by the child service definition templates found in the configuration files for hosts to be monitored, which the paper will demonstrate later in the section "Setting Up the Monitoring of Remote Servers for Security". The

[29] http://nagios.sourceforge.net/docs/1_0/xodtemplate.html

combination of a parent service template together with a child service template will form at least the minimum options that are required.

Next, one has to add another file named e.g. 'timeperiods.cfg' containing timeperiod definition templates. As seen earlier, check_period is set to 'none' in the 'parent-services.cfg' file where no specific time period is set because passive services are not checked directly by the central server. Therefore,

In 'timeperiods.cfg' file:

```
# 'none' timeperiod definition template
Define timeperiod{
        timeperiod_name         none            # No time is specified.
        alias                   zero-time-period
}
```

Notice how it is different from the full timperiod definition template as shown below but yet it fulfilled the minimal requirements needed.

```
# A full skeleton of a timeperiod definition template
# Note: This is not included in the /usr/local/Nagios/etc/generic/contacts/ directory.
define timeperiod{
        timeperiod_name         timeperiod_name
        alias                   alias
        sunday                  timeranges
        monday                  timeranges
        tuesday                 timeranges
        wednesday               timeranges
        thursday                timeranges
        friday                  timeranges
        saturday                timeranges
}
```
[30]

Again, the options that are italics are the minimal requirements for a timeperiod definition template.

After setting up the parent service templates, one can begin configuring the hosts needed for the central Nagios monitoring server to do monitoring. For instance, one wants to configure the central Nagios monitoring server to monitor local host (IP Address: 192.168.10.9), the distributed Nagios monitoring server (IP Address: 192.168.20.3) and remote host (IP Address: 192.168.20.1) in *Figure 5*. One will have to add in hosts, services and hostgroups definition templates. Each host configuration file (i.e. 192.168.10.9.cfg, 192.168.20.3.cfg and 192.168.20.1.cfg) will consist of a single host

[30] http://nagios.sourceforge.net/docs/1_0/xodtemplate.html

definition template as well as all its service definitions templates.

The following shows a full hostgroup definition template and host definition template:

```
# A full 'hostgroup' definition template
define hostgroup{
        hostgroup_name          hostgroup_name
        alias                   alias
        contact_groups          contact_groups
        members                 members
}
# A full 'host' definition template
# Note: This is not included in the /usr/local/Nagios/etc/networks/ directory.
define host{
        host_name               host_name
        alias                   alias
        address                 address
        parents                 host_names
        check_command           command_name
        max_check_attempts      #
        checks_enabled[0/1]
        event_handler           command_name
        event_handler_enabled[0/1]
        low_flap_threshold      #
        high_flap_threshold     #
        flap_detection_enabled[0/1][2]
        process_perf_data[0/1]
        retain_status_information[0/1]
        retain_nonstatus_information[0/1]
        notification_interval   #
        notification_period     timeperiod_name
        notification_options[d,u,r]
        notifications_enabled[0/1]
        stalking_options[o,d,u]
```

```
}[30]
```

The options that are italics are the minimal requirements that needed to be defined for a hostgroup template or a host definition template.

✍ In /usr/local/Nagios/etc/networks/localsite/ directory:

In localsite-hostgroup.cfg:

```
# local site 'hostgroup' definition template
define hostgroup{
        hostgroup_name                  Local-site
        alias                           Local site servers
        contact_groups                  admin
        members                         server-192.168.10.9
}
```

In 192.168.10.9.cfg:

```
# 192.168.10.9 'host' definition template
define host{
        host_name                       server-192.168.10.9
        alias                           local site server-192.168.10.9
        address                         192.168.10.9
         check_command                  check-host-alive
          max_check_attempts            1
        notification_interval           10
        notification_period             24x7
        notification_options            d,u,r
}

define service{
        use                             active-service
        host_name                       192.168.10.9
        service_description             FTP
```

[30]http://nagios.sourceforge.net/docs/1_0/xodtemplate.html

```
        check_command                   check_ftp!40!60!60
}
```

As one can see in 192.168.10.9.cfg file, the central Nagios monitoring server is now configured to monitor an active FTP service of the local server 192.168.10.9. When this check is executed in run-time, the central Nagios monitoring server will do a standard tcp three-way handshake connection on port 21; one can think of it as a full telnet session on port 21. Monitoring of services require self-customized or built-in Nagios-plugins. This is also reflected in *Figure* 6. The built-in Nagios plugins include 'check_ping', 'check_ftp', 'check_http', 'check_load', 'check_procs' and so forth. The plugin to use, as implied by the name of the plugin, really depends on the type of services being monitored. One advantage about using such built-in Nagios plugins is that they reveal little yet useful information sufficient enough to help facilitate the process of incident response. For instance, information showing the name of software and its version used by a service installed on a particular system. Sometimes such information allows security analysts to diagnose quickly whether or not an attack signalled by real-time NIDS alerts, is vulnerable to that version of the running service installed. By knowing which version of software is installed on where, system administrators can also run software updates or patches faster. This is especially so when security analysts have too many servers to look after, and cannot remember details of their servers' running services at the top of their heads. In the case of monitoring FTP service, one can see the version of the File Transfer Protocol (FTP) service installed as displayed in *Figure 8* below.

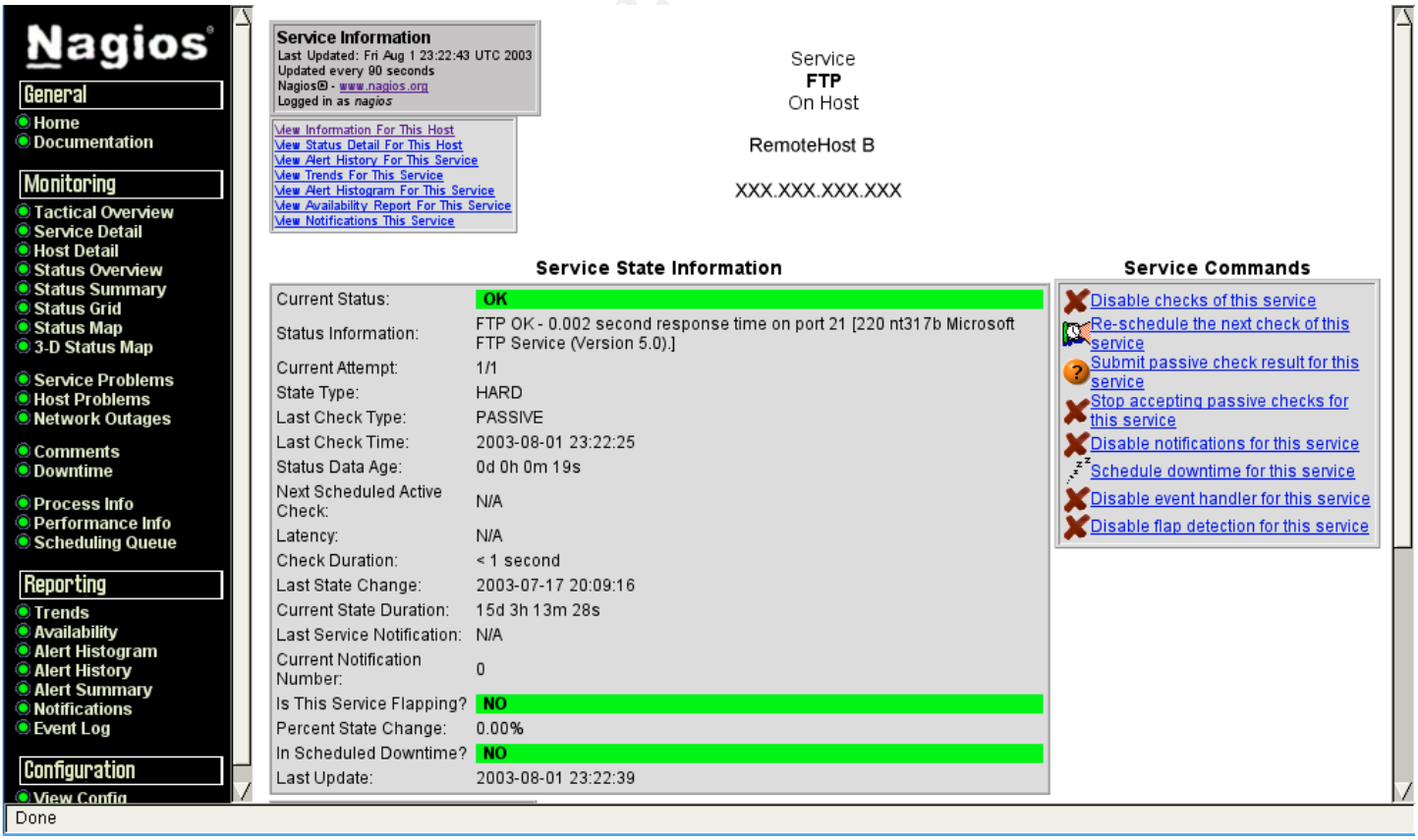


Figure 8. An FTP service that shows its type of software used and its version.

And depending on how one configures Nagios and more specifically what one monitors with Nagios, one can also use it to detect anomalous behaviour. To detect anomalous behaviour, Nagios can be configured to monitor the health of the operating systems. The paper took it one step further in terms of keeping track of the states of health.

✍ In /usr/local/Nagios/etc/networks/remotesite1/ directory:

In remotesite1-hostgroup.cfg:

```
# remote site 1 'hostgroup' definition template
define hostgroup{
        hostgroup_name                  remote-site1
        alias                           remote site 1 servers
        contact_groups                  admin
        members                         distributedserver1, firewall-192.168.20.1
}
```

In 192.168.20.3.cfg:

```
# 192.168.20.3 distributed monitoring server 'host' definition template
define host{
        host_name                       distributedserver1
        alias                           distributed monitoring server 1
        address                         192.168.20.3
        max_check_attempts              1
        notification_interval           10
        notification_period             24x7
        notification_options            d,u,r
}

define service{
        use                             active-service
        host_name                       192.168.20.3
        service_description             USERS
```

```
        check_command                   check_nrpe!check-users
}
```

With the above service definition set in 192.168.20.3.cfg, the central Nagios monitoring server will schedule an active check during run-time. Upon execution of the active check, it will trigger the plugin 'check_nrpe' that calls the right commands recognised by the NRPE daemon installed on distributed Nagios monitoring server located at remote site 1. This is an example of the process *step 4* of *Figure 5*. In this case, the NRPE daemon is told to run the command 'check-users' which if configured properly on the other remote end, will run a built-in Nagios plugin named 'check_users' that does a check on the total number of users logged in as shown in further details in *Figure 6*.

Therefore, Nagios can actually monitor for the number of users logged in to a particular server. In this case, assuming that no one is supposed to access that particular distributed monitoring server without authorized permissions, a security analyst will recognize immediately that something is unusual through an alert generated in *Figure 9*. This information can come in handy especially when a highly-planned compromise happens and a hacker gains a shell on one of the servers secretly.

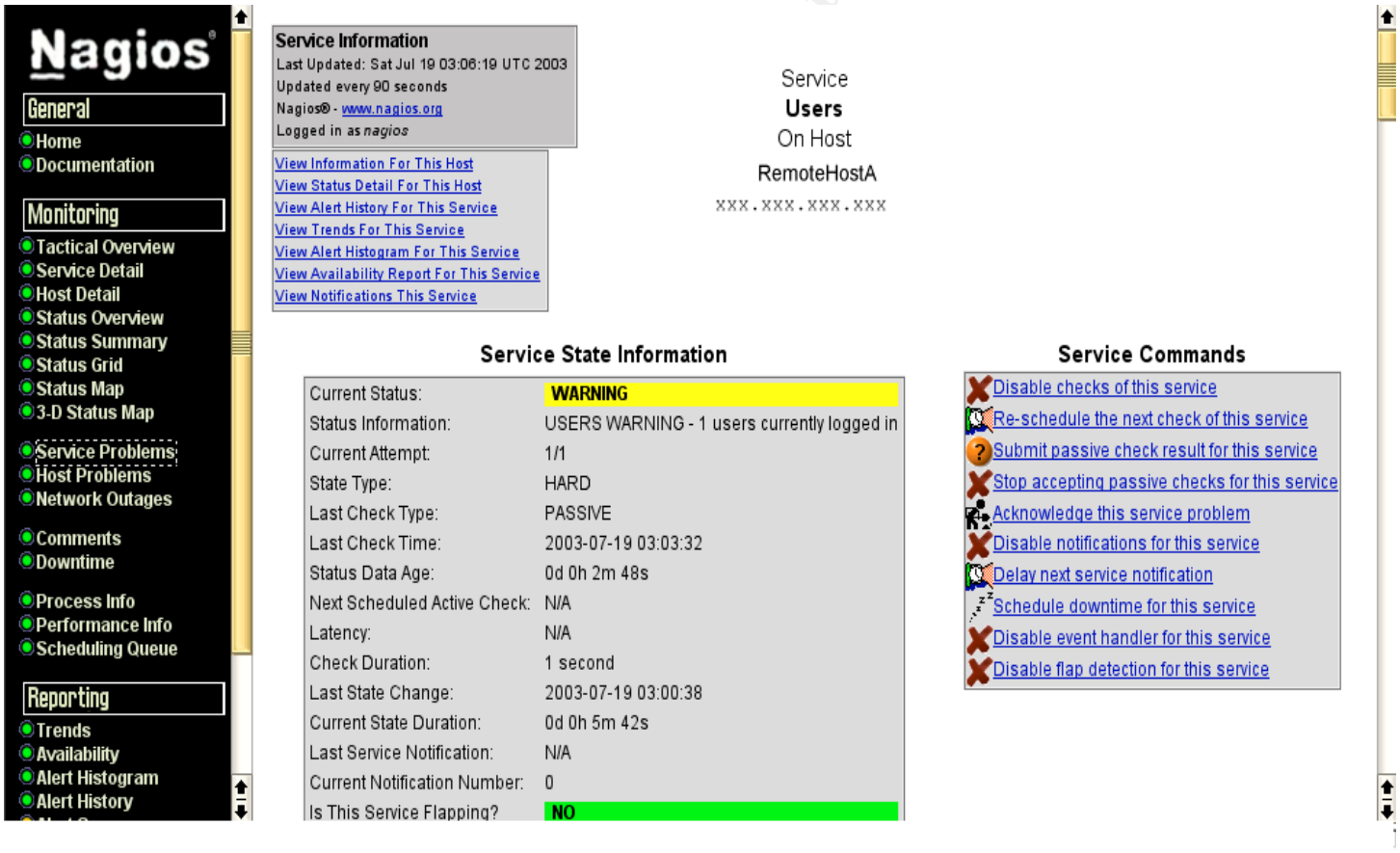


Figure 9. The diagram above shows the possibility of an anonymous user logged into RemoteHostA.

Take another example when an attacker has planted a trojan onto the distributed Nagios monitoring

system itself and opened up FTP port 21 on the server, allowing any anonymous user to upload files onto it. Configuring for monitoring of disk space can further help to protect the system internally when disk space on a particular partition fills up strangely at a faster rate than usual. For instance, as portrayed in the next *Figure*, where one observes that the disk space is 30% warning with only 5233420KB free on disk /var partition.

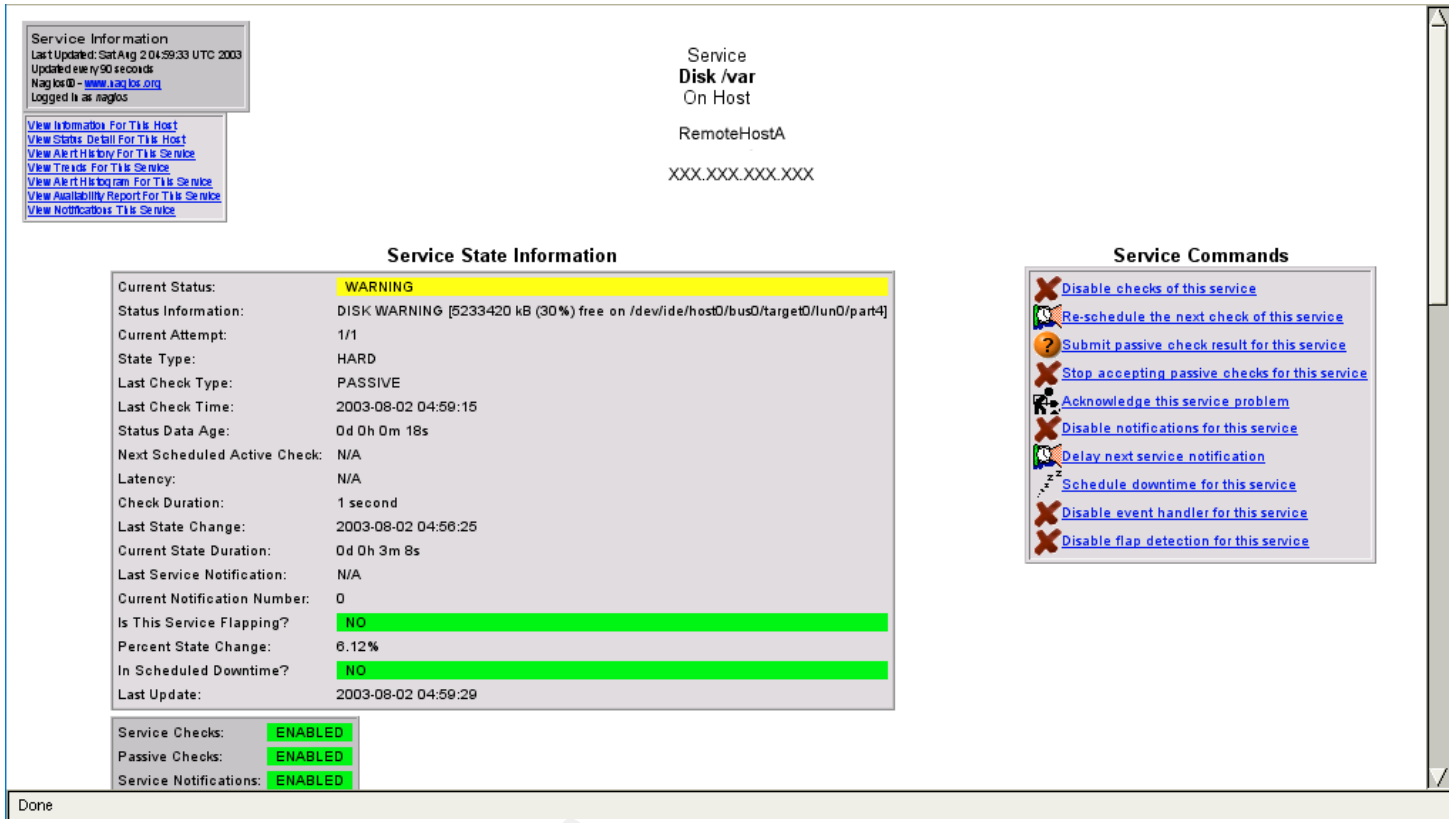


Figure 10. A Nagios alert sounding a disk space warning.

To configure for disk space monitoring of the distributed monitoring server (for instance 192.168.20.3), one simply append the 192.168.20.3.cfg file with another service definition:

```
define service{
        use                             active-service
        host_name                       192.168.20.3
        service_description             Disk /var
        check_command                   check_nrpe!check-disk-var
}
```

In a distributed monitoring environment, checks are allocated to other distributed Nagios monitoring servers. Therefore, one can also configure the distributed Nagios monitoring server to monitor the load of a particular remote system for example, the remote firewall with an internal IP Address 192.168.20.1. The check on its load is an example of the process *Step 3* of *Figure 5*. However, for the central Nagios monitoring server to recognize that such a check service exists on the remote network segment as well, one must first define that the appropriate host and service definitions for the object (i.e. 192.168.20.1). Therefore, within the same directory, one can add in an object configuration file named '192.168.20.1.cfg'. Then, add in the host name as a member into the 'remotesite1-hostgroup.cfg' file earlier.

In 192.168.20.1.cfg:

```
# 192.168.20.1 'host' definition template
define host{
        host_name                       firewall-192.168.20.1
        alias                           remote site1 firewall-192.168.20.1
        address                         192.168.20.1
        max_check_attempts              1
        notification_interval           10
        notification_period             24x7
        notification_options            d,u,r
}
# 'Load' service definition template
define service{
        use                             passive-service
        host_name                       192.168.20.1
        service_description             Load
        check_command                   check-freshness
}
```

From the above configuration set up for the two hosts (i.e. 192.168.20.3 and 192.168.20.1) to be monitored, one would have noticed that each of the host definition template is missing a host 'check_command' option whose value is normally set to 'check-host-alive', as seen earlier in the default host definition templates of 'sample-minimal.cfg' file. "If you leave this argument blank, the host will *not* be checked - Nagios will always assume the host is up."[30] This is because in this case, one would certainly not want the central Nagios monitoring server to run the 'check-host-alive' command on internal IP remote network addresses, which actually triggers the 'check_ping' plugin that will conduct a standard tcp ping check on them automatically. Also, it is to be noted that the 'check_command' option in the 'Load' service definition defined in 192.168.20.1.cfg will never be scheduled to run by the central Nagios monitoring server because the 'Load' service is passive with its check results fed in by the distributed Nagios monitoring server seen as an example of the process *step 2* of *Figure 5*.

[30] http://nagios.sourceforge.net/docs/1_0/xodtemplate.html

The 'Load' service check will only get scheduled for execution when the central Nagios monitoring server does not receive any passive check results from the distributed Nagios monitoring server for longer than freshness_check_interval (i.e. five minutes), set in the main configuration 'Nagios.cfg' file. And, if that happens the incident response team will be alerted by a critical alert containing the message "No updates from distributed monitoring server 1!" and will then proceed to diagnose the service problem immediately. This is because the 'check-freshness' check_command is configured to run a self-customizable script known as 'check_freshness' plugin, found in the default installation directory (i.e. /usr/local/Nagios/libexec) together with the rest of the built-in Nagios plugins.

```
# 'check_freshness' plugin script
# !/bin/sh


echo "No updates from distributed monitoring server 1!"
exit 2
```

The point to note here is the exit code number which is the standard compliant return code for 'critical' state alerts. Nagios even provide a set of good documentation for writing your own self-customizable plugins in two popular scripting languages: bash and perl. For more information one can go to http://Nagiosplug.sourceforge.net/developer-guidelines.html. Thus, due to the fact that the executable plugins that the Nagios monitor could communicate with are so flexible in their design, the security incident response teams who deploy Nagios need not worry about poor future scalability that results in monitoring issues. Nagios indeed does support future scalability, capable of monitoring many variety of services running on different platforms!

The monitoring of 'Load' is hard when there are probably scenarios in which one has machines backed up at 2AM where the load spikes. Most people would simply tune the load average up to the peak load in a given day. What one could have done is to sample the data on an hourly basis for a week all stored in a database. So now one can monitor the system load more closely during the day. Take an encounter where a user's laptop was sending a rather large amount of traffic through the firewall out of remote site 1. Turns out it was trojaned (probably when the user took the laptop home. The point here is that the symptom as seen in *Figure 11* below on the firewall of remote site 1 was the abnormally high load average. When one gets suspicious of the sudden misbehaving traffic and investigates, one would have noticed the traffic spike from one machine. The paper would like to stress that a NIDS requires that a signature be in place to detect this. In the scenario above, this was something new and was not being detected by the NIDS sensor (yet).

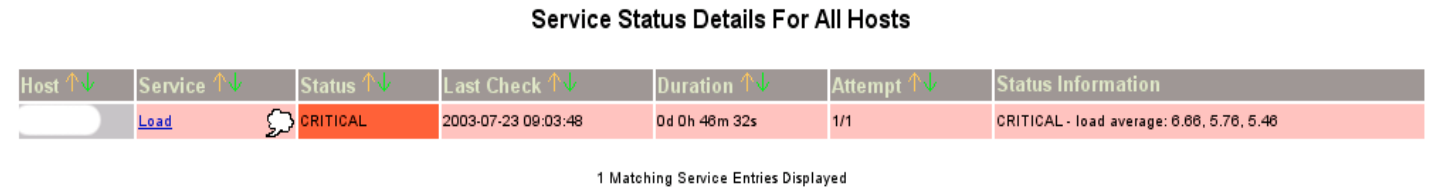


Figure 11. A critical Nagios alert sounding abnormal high load averages.

Therefore, a quick remedy for an NIDS that fails to detect zero-day incidents like the one mention earlier is to incorporate smarter security management, which can be done through integrating the distributed Nagios monitoring environment together with other security applications such as Firewalls/VPN, Snort IDS or the Prelude IDS system. This will provide better event correlation for security analysts. In secure corporate networks, a combination of security applications to be used together is needed (discussed in earlier section of "The Usefulness of Nagios Distributed Monitoring for Security"). The paper shall explain now how such integration can be done. For instance, Snort alerts can be configured to output to syslog files for the distributed Nagios monitoring server to monitor in real-time, assuming that Snort is installed on the same server as well. Or the distributed Nagios monitoring environment can 'move out' to fit into the distributed Prelude IDS architecture as shown below in *Figure 12*.

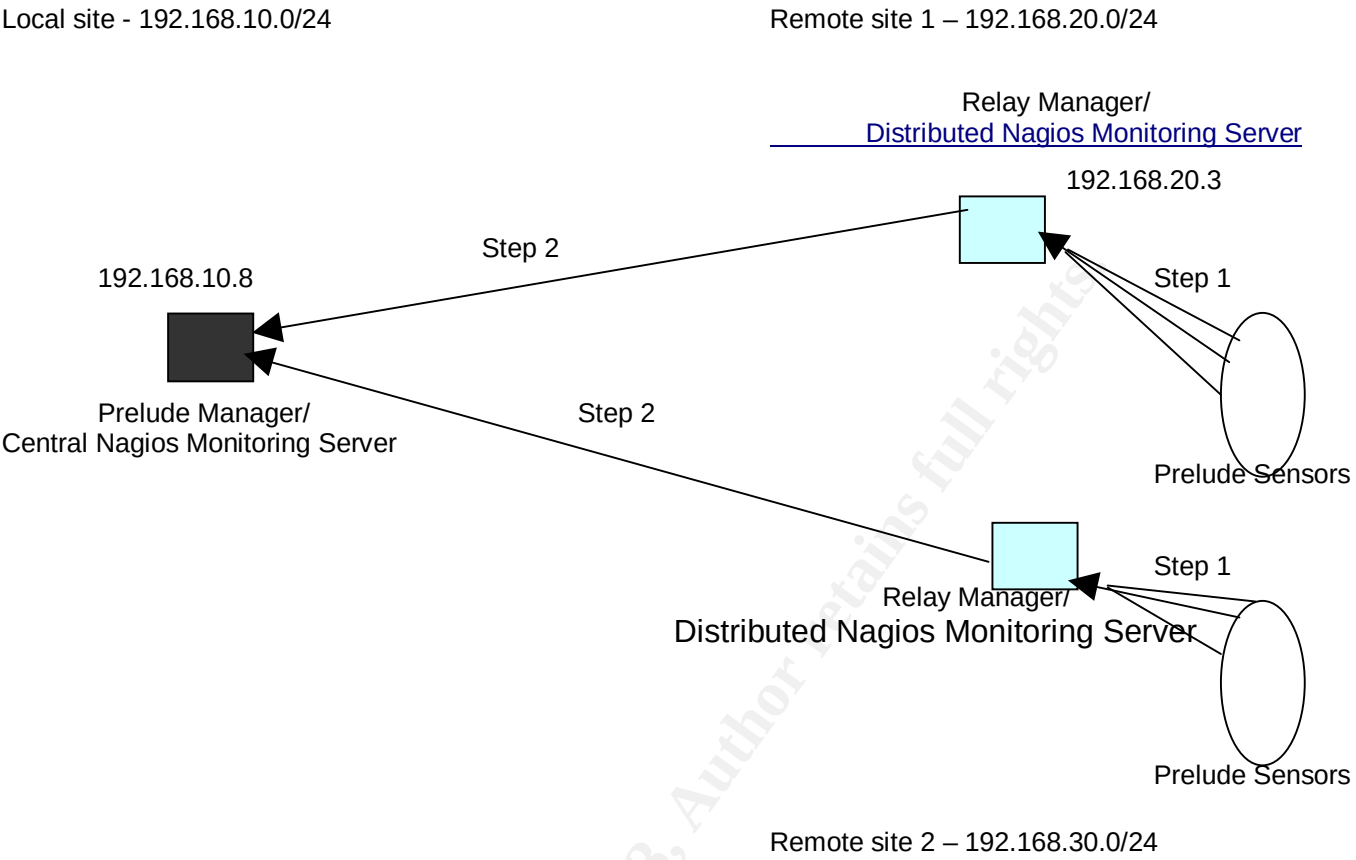


***Figure 12*. An example of merging Distributed Nagios Monitoring Environment into distributed Prelude IDS Architecture.**

As shown in the *Figure 12* above, the prelude manager sits on the central Nagios monitoring server as it is "the central logging point. It receives alerts from sensors and logs them using one or several plugins."[30] And to be able to receive alerts, the prelude manager on the central server have to communicate with all the relay managers located on remote network segments, which in this case happens to be the Nagios distributed monitoring servers as well. The relay managers are almost similar to the Prelude manager except that they are configured to pass messages to Prelude manager in real-time. These messages are generated by Prelude sensors including Prelude nids and Prelude Nagios and so forth. For more details of the managers and sensors, one can go to Prelude's main site found on http://www.prelude-ids.org.

The integrated architecture described above is made possible because of the ease of customising a particular external plugin to be used for monitoring by Nagios. In this situation, Prelude Nagios is actually just a type of customised external plugin made for Nagios, which can log all Nagios data to IDMEF format[i]. However, the plugin's usefulness should not be under-estimated and because with the Prelude Nagios, it is extremely convenient and more importantly, beneficial for security analysts that have already deployed the Prelude distributed IDS architecture to monitor distributed networks round-

[30] http://www.prelude-ids.org/article.php3?id_article=6

the-clock, to set up a distributed Nagios monitoring environment as well. Prelude Nagios allows Nagios data to be used in conjunction with the data produced from IDS alerts, vulnerability assessment (VA) and firewalls. This boosts accurate event correlations and especially when one discover next that alerts generated by Nagios can be made available via the same administrative interface as the IDS alerts. This then provides a summarized yet useful view of real-time analysis for the information security department in practice as shown below.

***Figure 13*. Prelude hybrid IDS using Piwi central administrative web Interface to report Nagios alerts.**

Therefore, one can see that all security, management and systems alerts can be monitored at the same time by Security Analysts via an administrative interface such as piwi as shown in the diagram above. Piwi is actually a "P(erl|relude) IDS Web Interface - A frontend to your Prelude database"[31] (assuming that all logs goes into the database). The colours seen under column 'P' come in three colours; green, yellow and red. They represent low, medium and high severity of real-time sensor events involved respectively. The column 'Timestamp' of the sensor events are in UTC format. And in the diagram above, one can see alerts triggered by the Prelude Nagios sensor. Such an interface

[31] http://www.prelude-ids.org/rubrique.php3?id_rubrique=6

allows security analysts to draw conclusions faster regarding the alerts being generated and speeds up their incident response time. For example, if there are "DoS attack" IDS alerts to a particular host and Nagios is reporting "green" for that remote host (i.e connectivity is fine), then it is most probably a false positive. However, if there are "DoS attack" IDS alerts and Nagios is also informing of flaky connections to that particular remote host as well, security analysts can be certain that there is a remote DoS happening! Flaky connections happen when the states of services start flapping or changes quickly within a short period of time. And, to further prove the usefulness of the customised Nagios external plugin 'Prelude Nagios' in communicating with the Prelude relay manager, imagine another situation whereby there are completely no new IDS alerts coming in, but Prelude Nagios displays a series of critical Nagios alerts of flaky service status connectivity to a remote network. Security analysts, who are always taught to maintain a 'panaroid' attitude, are at least suspicious and will start to doubt and think of possible causes; the daily functionalities of that relay manager guarding that remote network, a real zero-day DoS attack happening. All of which will pose threats to the perimeter of that remote network.

Early detection of DoS attacks maybe possible when Nagios alerts show up in warning states with slow response times returning from service ports at the beginning, and eventually turning critical when the critical thresholds specified are exceeded and timed out on connections. This view is based upon that "network monitoring has the potential to detect DoS attacks in early stages before they severely harm the victim. Our conjecture is that a DoS attack injects a huge amount of traffic into the network, which may alter the internal characteristics (e.g., delay and loss ratio) of the network"[32]. As an example back in Figure 13, by clicking on the Id 12358 for instance, will bring out another web interface in Figure 14 allowing the security analysts to view full details of the Nagios event report showing "FTP, CRITICAL HARD, 1, (Service Check Timed Out)", which in this case may represent a server damage scenario in terms of DoS that disrupts the proper functioning of running FTP service.

[32] http://216.239.39.104/search?q=cache:m4uV_rBx9hIJ:www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/12.pdf+detecting+service+violations+and+dos+attacks&hl=en&ie=UTF-8

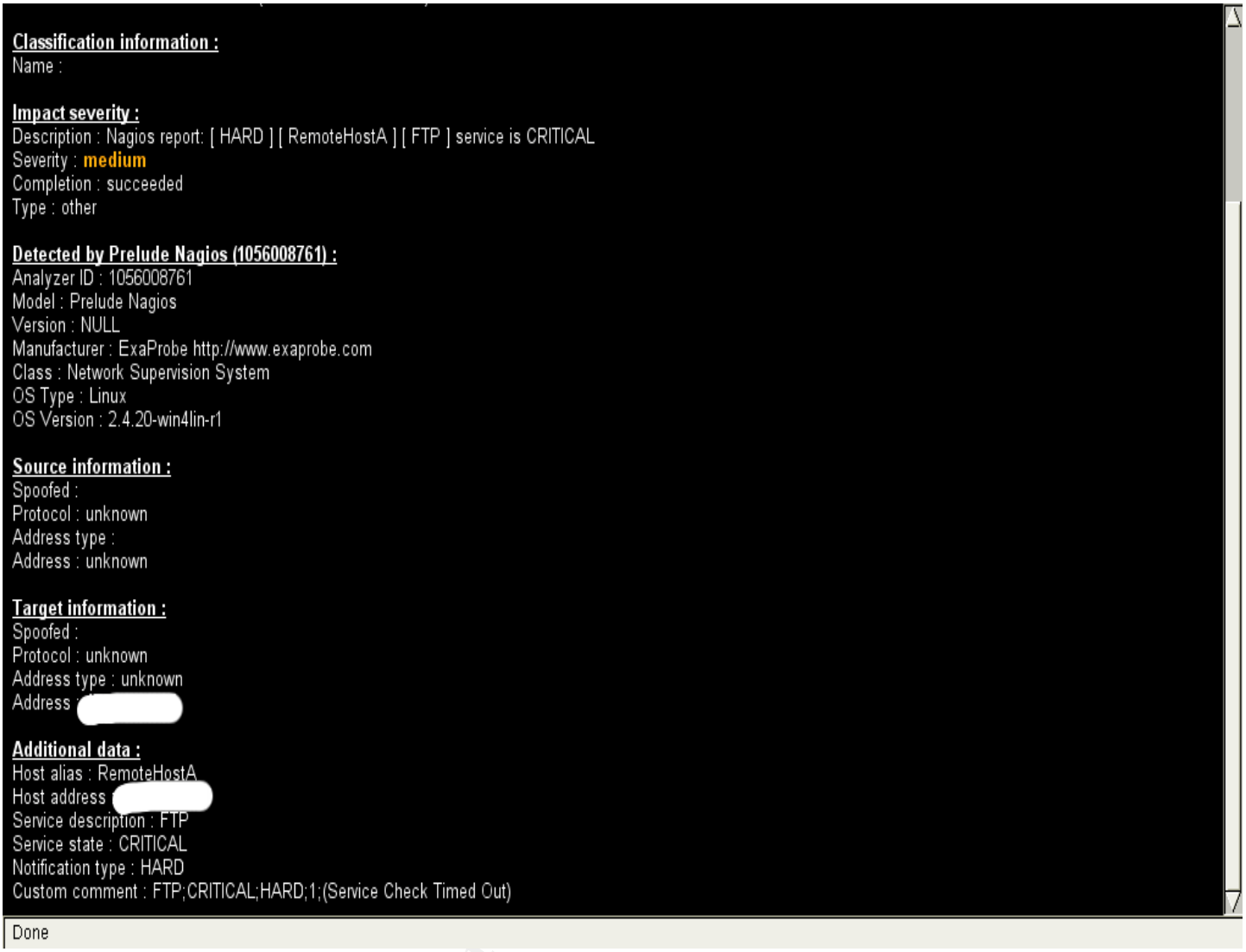


Figure 14. A medium severity alert of a FTP port closed on a particular server.

Alternatively, the scenario of ports closed can also be displayed on the Nagios web interface as shown below:

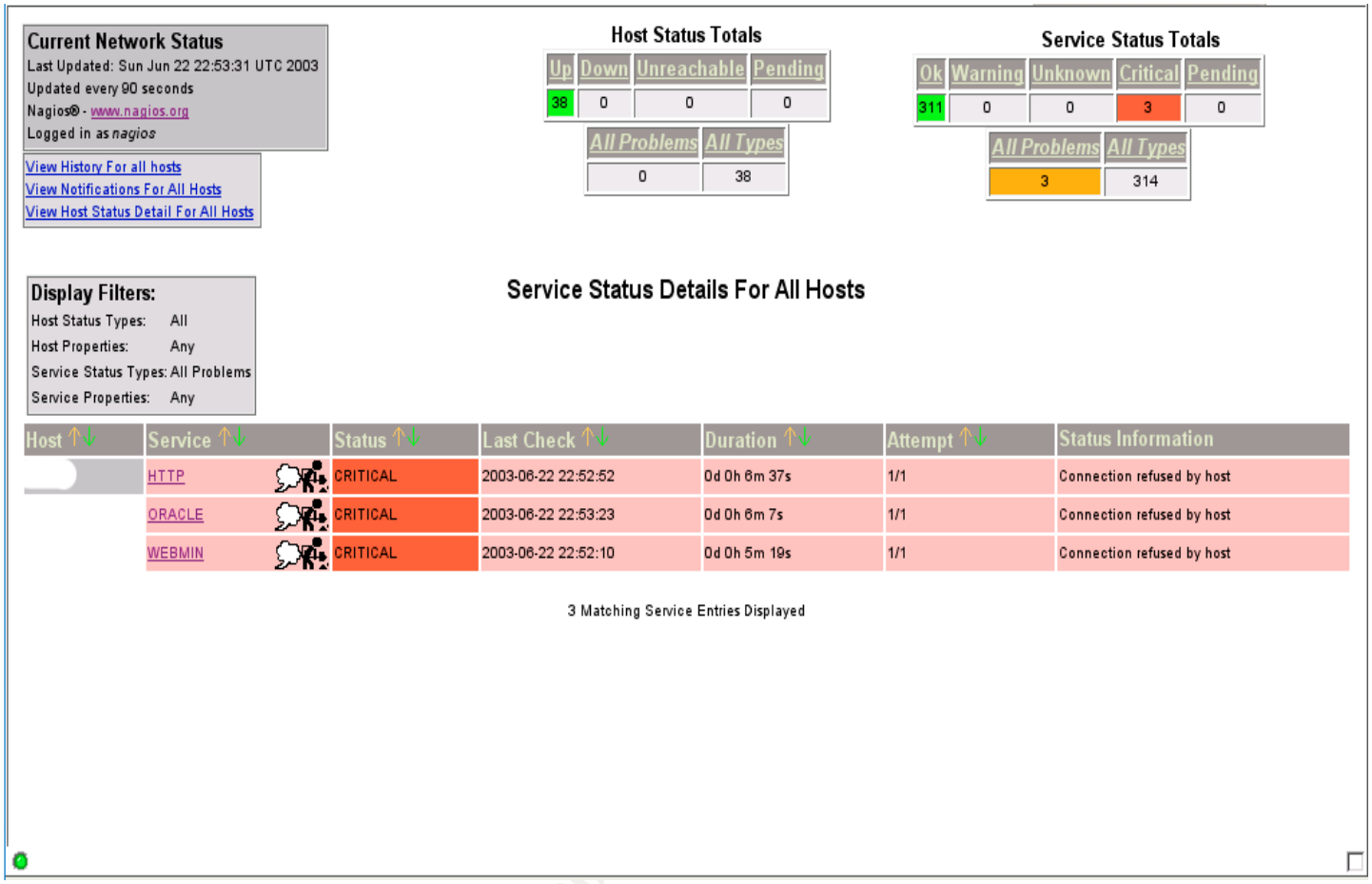


Figure 15. Scenario critical red alerts showing ports closed on a particular server displayed on Nagios web interface.

To complement the whole environment of event correlation, there is also a need to monitor log files of Firewalls/Virtual Private Networks. And, this can be done through Nagios. Because there is no sliver bullet to protect systems fully in the computer security world, a security department must at least ensure that when something bad happens, they know how to react with the right procedures to follow. Most importantly after a successful attack, the forsenics team has the right tools to help them investigate when the attack happened, discover how the systems got penetrated, the motive of the attack, and digs out the lost data and so on. And. "network-based tools are probably the most useful diagnosis and analysis available today to the incident response practitioner."[33] In this case, the forsenics team can also apply the Nagios network monitoring tool to check system log files. Using Nagios to check for system log files is better than doing periodic log reviews as it provides real-time alerts whenever an intrusion attempt is detected. "With log reviews, there's an inherent delay between when an attack occurs and when it's discovered. Even if you're reviewing logs daily, an attack can go unnoticed for hours -- which leaves a lot of time for a hacker to try to find the right opening in your systems. That's where intrusion detection comes in. An intrusion-detection system (IDS) constantly watches your network and alerts you or takes other actions when an intruder is

[33]http://www.oreilly.com/catalog/incidentres/chapter/ch07.html

detected."[7] The combination of managing security using Nagios together with an Prelude IDS alerts therefore proves almost informidable.

There is a built-in Nagios plugin called 'check_log.sh' which enables Nagios to be configured to monitor log files of a particular firewall (i.e. 192.168.20.1). For instance, a security analyst is interested in knowing events of attempted login failures on the firewall. One can configure as such:

Append the following service definition into 192.168.20.1.cfg file

```
# 'Log' Service definition template
define service{
        use                             passive-service
        host_name                       192.168.20.1
        service_description             Firewall Log
        check_command                   check_freshness
}
```

One can execute check_log Nagios plugin manually on 192.168.20.1 to find out the output to expect from Nagios in real-time:

```
# /usr/local/Nagios/libexec/check_log
Usage: check_log -F logfile -O oldlog -q query
Usage: check_log --help
Usage: check_log -version

# /usr/local/Nagios/libexec/check_log -F messages -O auth.log -q "authentication failure"
Log check ok - 0 pattern matches found\n
```

In the above case for instance, there are no recent login failures.

```
# /usr/local/Nagios/libexec/check_log -F messages -O auth.log -q "authentication failure"
1  > July  29 04:07:25 firewall-192.168.20.1 sudo(pam_unix)[11687]: authentication failure; logname=
   uid=0 euid=0 tty=pts/2 ruser= rhost=  user=kenneth
```

On the contrary in the situation above, analysts will be informed that there is an attempted user 'kenneth' trying to log into the firewall.

[7] http://insight.zdnet.co.uk/hardware/chips/0,39020436,2135496,00.htm

✍ In /usr/local/Nagios/etc/generic/commands/ directory:

Next, all 'check_command' options specified in the host configuration files above (i.e. 192.168.10.9.cfg, 192.168.20.3.cfg and 192.168.20.1.cfg) must be defined somewhere on the central Nagios monitoring server in order to be run. Thus, we can have a file called 'checkcommands.cfg'. Additional argument macros should be used over fixed values in this file so that one can be flexible towards customising the same service defined. Values passed into the arguments are specified in the check_command option with an argument separator (!).

To find out the relevant arguments to pass in and what the built-in Nagios plugins are about, one can always do a (--help). For instance:

```
# /usr/Nagios/libexec/check_ftp --help
check_ftp (Nagios-plugins 1.3.0) 1.13
The Nagios plugins come with ABSOLUTELY NO WARRANTY. You may redistribute
copies of the plugins under the terms of the GNU General Public License.
For more information about these matters, see the file named COPYING.
Copyright (c) 1999 Ethan Galstad (Nagios@Nagios.org)

This plugin tests FTP connections with the specified host.

Usage: check_ftp -H host -p port [-w warn_time] [-c crit_time] [-s send]
       [-e expect] [-W wait] [-t to_sec] [-v]
Options:
 -H, --hostname=ADDRESS
   Host name argument for servers using host headers (use numeric
   address if possible to bypass DNS lookup).
 -p, --port=INTEGER
   Port number
 -s, --send=STRING
   String to send to the server
 -e, --expect=STRING
   String to expect in server response -W, --wait=INTEGER
   Seconds to wait between sending string and polling for response
 -w, --warning=DOUBLE
   Response time to result in warning status (seconds)
```

```
-c, --critical=DOUBLE
   Response time to result in critical status (seconds)
-t, --timeout=INTEGER
   Seconds before connection times out (default: 10)
-v    Show details for command-line debugging (do not use with Nagios server)
-h, --help
   Print detailed help screen
-V, --version
   Print version information
```

In checkcommands.cfg:

```
#All plugins are stored in the default directory /usr/local/Nagios/libexec set to $USER1$ macro in
#resource.cfg earlier.
# 'check-host-alive' command definition
define command{
        command_name    check-host-alive
        command_line    $USER1$/check_ping -H $HOSTADDRESS$ -w 3000.0,80% -c 5000.0,100% -
p 1
}


# 'check_ftp' command definition
define command{
        command_name    check_ftp
        command_line    $USER1$/check_ftp -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$ -t \
$ARG3$
}

# 'check_nrpe' command definition
define command{
        command_name    check_nrpe
        command_line    $USER1$/check_nrpe -H $HOSTADDRESS$ -c $ARG1$
}

# 'check-freshness' command definition
```

```
define command{
        command_name    check-freshness
        command_line    $USER1$/check_freshness
}
```

✍ In /usr/local/Nagios/etc/generic/contacts/ directory:

Within this directory, one has to define all contact definition templates. These are meant for notifying Nagios events through email or pager besides the central Nagios web interface. Therefore, in a file e.g. 'contacts.cfg':

```
# 'SecurityAnalysts' contact definition
define contact{
        contact_name                    SecurityAnalysts
        alias                           Security Department Analysts
        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                           <email address>         ; as defined in Nagios.cfg
      }
```

Note that there is no need to define 'notify-by-email' and 'host-notify-by-email' in the checkcommands.cfg file because they are already defined properly in the misccommands.cfg-sample file by default. One can just simply move that file to misccommands.cfg and put it in the /usr/local/Nagios/etc/generic/commands directory. A full contact definition template looks like this:

```
# A full skeleton of a contact definition template.
# Note: This is not included in the /usr/local/Nagios/etc/contacts/ directory.
define contact{
        contact_name                    contact_name
        alias                           alias
        host_notification_period        timeperiod_name
        service_notification_period timeperiod_name
        host_notification_options[d,u,r,n]
        service_notification_options[w,u,c,r,n]
        host_notification_commands      command_name
```

```
        service_notification_commands   command_name
        email                           email_address
        pager                           pager_number or pager_email_gateway
}[30]
```

Another configuration file for contacts needed by Nagios is the contactgroup definition template.

Therefore, the paper adds in another file e.g. 'contactgroups.cfg' into the directory of /usr/local/Nagios/generic/contacts/ too.

In 'contactgroups.cfg' file:

```
# 'admin' contact group definition
define contactgroup{
        contactgroup_name       admin
        alias                   Security Department
        members                 SecurityAnalysts
    }
```

The above will have to follow the settings of a full contactgroup definition template as shown below because all options are compulsory.

```
# Skeleton of a full contactgroup definition template
# Note: This is not included in the /usr/local/Nagios/etc/contacts/ directory.
define contactgroup{
        contactgroup_name       contactgroup_name
        alias                   alias
        members                 members
}[30]
```

With the setup of the central Nagios monitoring server, one can see that Nagios actually provides a few alternatives to monitor for unusual events. It can be configured to notify Security Analysts via an interactive web Interface as seen in *Figure 1,* email or SMS alerts. In fact, Nagios is aware of the need for more alternatives of notifications in time to come and always welcomes other projects, for instance NagMIN and Nagat. Users may not want to use the interactive web interface provided by default. One can go to http://Nagiosplug.sourceforge.net/projects/sourceforge.php for more information regarding these projects.

⑩Setting up Distributed Nagios Monitoring Server For Remote Site 1

[30]http://nagios.sourceforge.net/docs/1_0/xodtemplate.html
[30]http://nagios.sourceforge.net/docs/1_0/xodtemplate.html

A distributed Nagios monitoring server works like a central Nagios monitoring server except that it is "usually a bare-bones installation of Nagios. It doesn't have to have the web interface installed, send out notifications, run event handler scripts, or do anything other than execute service checks if you don't want it to"[34]. Therefore, other than the switches mention below in the configuration of the overall program Nagios.cfg file, the rest of the switches are to be set the same as the configuration of the central monitoring server. Because there is no need for a web interface, the configuration of cgi.cfg and resource.cfg files are unnecessary. Overall, the main task of a distributed Nagios monitoring server is to execute service checks on remote servers and report back to the central server as seen in *Figure 5*.

In Nagios.cfg (main configuration file for overall program) :

✍ Enable active checking.

```
# Enable active checking
execute_service_checks=1
```

This option is turned on as the distributed Nagios monitoring server has to monitor the servers in its own network segment. The process is seen for instance in *step 3* of *Figure 5*.

✍ Disable accept passive service checks.

```
# Disable passive checks received
accept_passive_service_checks=0
```

This option is disabled because according to the distributed monitoring setup in *Figure 5,* there is no need for the distributed Nagios monitoring server to accept any passive check results.

✍ Enable Obsessive compulsive service processor (OCSP) option.

```
# Enable obsess_over_services
obsess_over_services=1
# Set ocsp_command=<name of script>
ocsp_command=submit_check_result
```

There is a need to turn on obsess_over_service because the distributed servers have to submit passive service results to the central server seen in for instance *step 2* of *Figure 5*. The distributed monitoring servers made use of the OCSP command to call for a 'submit_check_result' script as shown below. The script will pipe all check results to send_nsca program, which communicates to the

[34] http://nagios.sourceforge.net/docs/1_0/distributed.html

nsca daemon on the central Nagios monitoring server.

```
# The 'Submit_check_result' shell script
#!/bin/sh

# Arguments:
#  $1 = host_name (Short name of host that the service is
#       associated with)
#  $2 = svc_description (Description of the service)
#  $3 = state_string (A string representing the status of
#       the given service - "OK", "WARNING", "CRITICAL"
#       or "UNKNOWN")
#  $4 = plugin_output (A text string that should be used
#       as the plugin output for the service checks)
#
# In global_vars, set Nagios_CENTRAL_SERVER=<hostaddress>
source /etc/Nagios/global_vars

# Convert the state string to the corresponding return code
return_code=-1

case "$3" in
        OK)
                return_code=0
                ;;
        WARNING)
                return_code=1
                ;;
        CRITICAL)
                return_code=2
                ;;
        UNKNOWN)
                return_code=-1
```

```
        ;;
    esac


# pipe the service check info into the send_nsca program, which
# in turn transmits the data to the nsca daemon on the central
# monitoring server
#Assume that the send_nsca program and its configuration file is located in the
#/usr/local/Nagios/bin and /usr/local/Nagios/var directories, respectively


/bin/echo -e "$1\t$2\t$return_code\t$4\n" | /usr/local/Nagios/bin/send_nsca \
$Nagios_CENTRAL_SERVER -c /usr/local/Nagios/etc/send_nsca.cfg
```

✍ Disable freshness checking.

```
check_service_freshness=0
```

There is no need for the distributed monitoring server to receive passive results so disable the check_freshness option.

Configuration of Object Configuration files.

✍ In /usr/local/Nagios/etc/generic/services directory:

In the Parent-services.cfg:

```
#Parent template 2 -- passive-service (inherited by services that are actively monitored by distributed
#monitoring servers and not actively checked by central monitoring server)
define service{
        name                    passive-service     ; The 'name' of this service template,
referenced in other service definitions
        active_checks_enabled   1                   ; Active service checks are enabled.
        passive_checks_enabled  0                   ; Passive service checks are disabled.
        parallelize_check       1                   ; There are active service checks. (disabling
this can lead to major performance problems)
        obsess_over_service     1                   ; We should obsess over this service (if
necessary)
        check_freshness         0                   ; Check service 'freshness' is disabled.
```

```
        max_check_attempts      1               ; Checked once.
        normal_check_interval   5               ; Service is checked every 5 minutes.
        retry_check_interval3                   ; Service is re-checked every 3 minutes.
        check_period            24x7            ; look at '24x7' time definition found in
'timeperiods.cfg' file
        notification_interval   0               ; Notifications sent out once only.
        notification_period     24x7            ; Notifications 24x7.
        notification_options    u,c,r           ; Notifications .
        contact_groups          admin           ; contact us
        register                0               ; DONT REGISTER THIS DEFINITION - ITS
NOT A REAL SERVICE, JUST A TEMPLATE!
}
```

Not to be confused by the name of the parent template above, it is important to note that although the 'passive-service' definition template is configured for the distributed Nagios monitoring server to perform active service checks, these are actually passive service checks on the central Nagios monitoring server. Thus, the parent template is configured to enable active checking only.

✍ In /usr/local/Nagios/etc/networks/remotesite1 directory:

In remotesite1-hostgroup.cfg:

```
# remote site 1 'hostgroup' definition template
define hostgroup{
        hostgroup_name                  remote-site1
        alias                           remote site 1 servers
        contact_groups                  admin
        members                         firewall-192.168.20.1
}
```

In 192.168.20.1.cfg:

```
# 192.168.20.1 'host' definition template
define host{
```

```
        host_name                       firewall-192.168.20.1
        alias                           remote site1 firewall-192.168.20.1
        address                         192.168.20.1
          check_command                 check-host-alive          #Add this in here
        max_check_attempts              1
        notification_interval           10
        notification_period             24x7
        notification_options            d,u,r
}
# 'Load' service definition template
# Note that unlike the central monitoring server, the check_command is not 'check_freshness'
#anymore.
define service{
        use                             passive-service
        host_name                       192.168.20.1
        service_description             Load
        check_command                   check_nrpe!check-load
}
# 'Log' Service definition template
define service{
        use                             passive-service
        host_name                       192.168.20.1
        service_description             Firewall Log
        check_command                   check_nrpe!check-loginattempts-firewall-log
}
```

Nagios is 'fussy' about having the same type of services defined. Therefore, it is important that both values of the 'service_description' option seen above are defined exactly the same way on the central and distributed Nagios monitoring server. The configuration needed for a setup of a distributed Nagios monitoring server would end here, as the configuration of the commands and contacts definitions on the distributed Nagios monitoring server works in the same way as described in the setup of the central Nagios monitoring server.

After discussing the usefulness of systems monitoring together with IDS alerts, how do security analysts go about configuring the distributed Nagios monitoring server to communicate with a Prelude relay manager then? For further details, one can refer to the additional appendix attached at the back of the paper.

Setting up NRPE on a particular server e.g. 192.168.20.1

The configuration of NRPE on a particular host really depends on how one wants NRPE to be run. There are several ways of running NRPE, including running as inetd or xinetd or as a daemon. If running as inetd is preferred, one has to comment out the following options in the nrpe.cfg.

In nrpe.cfg:

```
# server_port=
# server_address=
# allowed_hosts=
# nrpe_user=
# nrpe_group=
```

Configure the following if NRPE is to be run under inetd:

Remember to separate them by tab

```
nrpe stream tcp nowait Nagios /usr/sbin/tcpd/usr/local/Nagios/libexec/n\rpe -i /usr/local/Nagios/nrpe.cfg
```

Configure the following if NRPE is to be run as a daemon.

In nrpe.cfg:

```
# Port number assigned for NRPE daemon
server_port=5666

# Binding to a particular interface; the IP address of the host that NRPE is sitting on or
#comment out this option if NRPE is bind to multiple interfaces i.e. 0.0.0.0.
server_address=192.168.20.1
```

```
# The hosts that are allowed to talk to the NRPE daemon. If there are more than 1 host,
#separate their IP addresses by a comma. In this case, it would be the IP address of the
#distributed monitoring server for the Remote site 1.
allowed_hosts=192.168.20.3
```

```
# The name or UID of the user that NRPE daemon is running
nrpe_user=nrpe

# The name or GID of the group that NRPE daemon is running
nrpe_group=nrpe

# command[<command_name>]=<command_line>
# To configure for checking of load service
command[check-load]=/usr/local/Nagios/libexec/check_load -w 0.80,0.80,0.80 -c 1.00,1.00,1.00 -t 90
# To configure for checking of failed login attempts
command[check-loginattempts-firewall-log]=/usr/local/Nagios/libexec/check_log -F messages -O auth.log -q "authentication failure"
```

The most important point of configuring the NRPE is to make sure that the distributed Nagios monitoring server (i.e. 192.168.20.3) calls the right commands recognised by the NRPE daemon when executing the check_nrpe plugin, which then executes the local check_commands that addresses the right directory of where the actual plugins are located (i.e. <command_line>). In other words, the command argument (in the case of 192.168.20.1.cfg, the command argument is 'check-load') passed into the check_nrpe plugin must match the <command_name>. Therefore, the 'Load' service definition in 192.168.20.1.cfg is configured to allow the Nagios distributed monitoring server to check the load averages of the host 192.168.20.1.

⑩Setting up NSCA

The configuration for NSCA comes in two files:

nsca.cfg - Configuration file for the NSCA server.

send_nsca.cfg - Configuration file for the NSCA client.

On the Central Nagios monitoring server (i.e. 192.168.10.8 in this case)

Like NRPE, NSCA can also be run as a service under inetd or as a daemon on its own. In the nsca.cfg file, there are also options like server_port, server_address, allowed_hosts, nsca_user and nsca_group. All of these options should be comment out if NSCA is to be run under inetd. With everything set up properly on the central Nagios monitoring server as demonstrated in earlier, all that is left to be configured for NSCA lies mainly in the rest of the options in the nsca.cfg as shown:

```
# DEBUGGING OPTION
# This option determines whether or not debugging
```

```
# messages are logged to the syslog facility.
# Values: 0 = debugging off, 1 = debugging on

debug=1                                 # Enable debugging
# COMMAND FILE
# This is the location of the Nagios command file that the daemon
# should write all service check results that it receives.

command_file=/usr/local/Nagios/var/rw/Nagios.cmd        # Default file location

# ALTERNATE DUMP FILE
# This is used to specify an alternate file the daemon should
# write service check results to in the event the command file
# does not exist.  It is important to note that the command file
# is implemented as a named pipe and only exists when Nagios is
# running.  You may want to modify the startup script for Nagios
# to dump the contents of this file into the command file after
# it starts Nagios.  Or you may simply choose to ignore any
# check results received while Nagios was not running...

alternate_dump_file=/usr/local/Nagios/var/rw/nsca.dump  # Default file location

# AGGREGATED WRITES OPTION
# This option determines whether or not the nsca daemon will
# aggregate writes to the external command file for client
# connections that contain multiple check results.  If you
# are queueing service check results on remote hosts and
# sending them to the nsca daemon in bulk, you will probably
# want to enable bulk writes, as this will be a bit more
# efficient.
# Values: 0 = do not aggregate writes, 1 = aggregate writes

aggregate_writes=0                              # Disable bulk writes
```

```
# APPEND TO FILE OPTION
# This option determines whether or not the nsca daemon will
# will open the external command file for writing or appending.
# This option should almost *always* be set to 0!
# Values: 0 = open file for writing, 1 = open file for appending

append_to_file=0

# MAX PACKET AGE OPTION
# This option is used by the nsca daemon to determine when client
# data is too old to be valid.  Keeping this value as small as
# possible is recommended, as it helps prevent the possibility of
# "replay" attacks.  This value needs to be at least as long as
# the time it takes your clients to send their data to the server.
# Values are in seconds.  The max packet age cannot exceed 15
# minutes (900 seconds).

max_packet_age=30

# DECRYPTION PASSWORD
# This is the password/passphrase that should be used to descrypt the
# incoming packets.  Note that all clients must encrypt the packets
# they send using the same password!
# IMPORTANT: You don't want all the users on this system to be able
# to read the password you specify here, so make sure to set
# restrictive permissions on this config file!

#password=                              # This is unset, please put in your own secure password!

# DECRYPTION METHOD
# This option determines the method by which the nsca daemon will
# decrypt the packets it receives from the clients.  The decryption
# method you choose will be a balance between security and performance,
# as strong encryption methods consume more processor resources.
```

```
# You should evaluate your security needs when choosing a decryption
# method.
#
# Note: The decryption method you specify here must match the
#       encryption method the nsca clients use (as specified in
#       the send_nsca.cfg file)!!
# Values:
#
#       0 = None        (Do NOT use this option)
#       1 = Simple XOR  (No security, just obfuscation, but very fast)
#
#       2 = DES
#       3 = 3DES (Triple DES)
#       4 = CAST-128
#       5 = CAST-256
#       6 = xTEA
#       7 = 3WAY
#       8 = BLOWFISH
#       9 = TWOFISH
#       10 = LOKI97
#       11 = RC2
#       12 = ARCFOUR
#
#       14 = RIJNDAEL-128
#       15 = RIJNDAEL-192
#       16 = RIJNDAEL-256
#
#       19 = WAKE
#       20 = SERPENT
#
#       22 = ENIGMA (Unix crypt)
#       23 = GOST
#       24 = SAFER64
#       25 = SAFER128
```

```
#       26 = SAFER+
decryption_method=3             # Set to 3DES here
```

On the Distributed Nagios Monitoring Server (i.e. 192.168.20.3 in this case)

The most important part about configuring the nsca.cfg is to make sure that the password used and decryption method chosen are similar to the send_nsca remote servers configuration of password and encryption method, otherwise NSCA will not work! It has a variety of in-built encryption methods to choose from, but is used at the risk of performance.

Conclusion

All in all, it is the people behind the scenes that play a crucial role in managing security in today's networks. Nagios is just a network monitoring tool. And although it is useful to further enhance the confidentiality, integrity and availability of resources, it really depends on how intelligently the plugins work and how security analysts configure them to work on crucial company's servers. "Monitoring provides immediate security in a way that just doing a vulnerability assessment or dropping a firewall into a network can never provide. Monitoring provides dynamic security in a way that yet another security product can never provide. And, as security products are added into a network - firewalls, IDSs, specialized security devices - monitoring only gets better"[35]. This is because faster detection and response can be achieved through active network monitoring of events from a number of security components. It is always crucial for security analysts to maintain monitoring 24x7 with an 'active' mind. In other words, they must work intelligently. For instance, if there are no new alerts generated for some time, security analysts should always make it a habit to check around to make sure things are still moving fine. With GNU license open source software like Nagios, it is also advisable for one to subscribe to relevant mailing lists to keep in touch with new bug fixes and patch to the latest version of software applications used. Currently, Nagios is at version 1.1 and is expected to roll into 2.0 by summer of 2003 with further enhancements including passive host checks, which enable the central monitoring server to be able to receive not only remote services results but host status as well. For more interests in the next version, one can view http://www.Nagios.org/upcoming.php. "It takes constant monitoring. It's not one tool over another; it's the mind-set of the staff who review our systems, read information, put in proper patches and do proper testing."[36]

[35] http://www.counterpane.com/msm.html
[36] http://www.computerworld.com/news/1999/story/0,11280,35031,00.html

End Notes

Nagios version 1.1

Nagios-Plugins 1.3.0

NRPE version of 1.8 and 2.0Beta4.

NSCA version of 2.4

Nagios Main Website:

Galstad, Ethan. "Nagios Version 1.0" 1999-2002. URL: http://www.Nagios.org/

References

[1] Lamsal, P. "Management of the Next Generation IP Core Network." 16th April 1999. URL: http://www.tml.hut.fi/Opinnot/Tik-110.551/1999/papers/12ManagementOfIPngCore/ipcore.html

[2] Frisch, Æ. "Top Five Open Source Packages for System Administrators." 12th May 2002. URL: http://www.onlamp.com/lpt/a/2787

[3] Winkler, Ira. "Ounce of Prevention." November 1999. URL: http://www.infosecuritymag.com/articles/1999/winkler.shtml

[4] Smith, B. "Thinking about Security Monitoring and Event Correlation." 3rd November 2000. URL: http://www.securityfocus.com/infocus/1231

[5] Walker, L. "The View From Symantec's Security Central." 9th January 2003. URL: http://www.washingtonpost.com/wp-dyn/articles/A28625-2003Jan8.html

[6] Huston, B. "Protect Against Zero Day Exploits." 30th October 2002. URL: http://www.itworld.com/nl/security_strat/10302002/

[7] Bogue, R. L. "You've Been Hacked: Now Prevent Future Attacks." 3rd June 2003. URL: http://insight.zdnet.co.uk/hardware/chips/0,39020436,2135496,00.htm

[8] Messmer, E. and Pappalardo, D. "A Year After Meltdown: No Silver Bullet for DoS." 2nd May 2001. URL: http://www.nwfusion.com/news/2001/0205ddos.html

[9] Briguet, C. "Smart Management of Security Alerts." 14th April 2003. URL: http://www.exaprobe.com/labs/manage/

[10] Metadigm, Ltd. "Firewalls." 1989. URL: http://www.metadigm.co.uk/solutions/firewall.shtml

[11] Vandoorselaere, Y. "Welcome to the Prelude Website." (Prelude Homepage). 1998. URL: http://www.prelude-ids.org/

[12] Sun Microsystems Inc. "Java Distributed Computing Backgrounder." 1998. URL: http://java.sun.com/products/javaspaces/backgrounder/

[13] Galstad, E. "Nagios is Hogging Memory or Causing the Server to Swap Heavily." FQ115. 20th November 2003. URL: http://www.Nagios.org/faqs/viewfaq.php?faq_id=115&expand=false&showdesc=true

[14] Kohlhepp, R. J. "NetSaint: A Prayer Answered for Network Managers in A Budget." 15th May 2000. URL: http://www.networkcomputing.com/1109/1109sp5.html

[15] Galstad, Ethan. "Nagios Version 1.0 Documentation" 14th April 2003. URL: http://Nagios.sourceforge.net/docs/1_0/

[16] Smith, R. "VPN vs SSL." 3rd September 1999. URL: http://lists.shmoo.com/pipermail/vpn/1999-September/000223.html

[17] Red Hat.,Inc. "RedHat Linux 7.2: The Official Red Hat Linux Reference Guide." *Chapter 9. TCP Wrappers and XINETD.* 2001. URL: http://www.redhat.com/docs/manuals/linux/RHL-7.2-Manual/ref-guide/ch-tcpwrappers.html

[18] Lothos. "An Introduction To Tcp Wrappers." 14 November 2002. URL: http://www.undergroundnews.com/files/texts/underground/hacking/keenveracity5.htm

[19] Internet Security Systems. 1994-2003. URL: http://www.iss.net/security_center/advice/Underground/Hacking/Methods/Technical/Spoofing/default.htm

[20] Refer to [15].

[21] CERT/CC "CERT Advisory CA-2002-03 Multiple Vulnerabilities in Implementations of the Simple Network Management Protocol (SNMP)." 4th August 2003. URL: http://www.cert.org/advisories/CA-2002-03.html

[22] Hurley, E. "SNMP Flaw is Serious, Fix isn't Easy." 13th February 2002. URL: http://searchsecurity.techtarget.com/originalContent/0,289142,sid14_gci802127,00.html

[23] Refer to [15].

[24] Clavister, AB. "VPN Overview." 2002. URL: http://www.clavister.com/manuals/ver8x/manual/vpn/vpn_overview.htm

[25] Refer to [15].

[26] Wheeler, D. A. "Secure Programming for Linux HOWTO." 9th February 2002. URL: http://new.linuxnow.com/docs/content/Secure-Programs-HOWTO-html/Secure-Programs-HOWTO-7.html

[27] Kamthan, P. "CGI Security: Better Safe Than Sorry." 19th September 1999. URL: http://tech.irt.org/articles/js184/

[28] - [30] Refer to [15].

[31] Polombo, D. "Prelude HOWTO." 16th September 2002. URL: http://www.prelude-ids.org/article.php3?id_article=6

[32] Refer to [11].

[33] Habib, A., Hefeeda, M. M. and Bhargava, B. K. "Detecting Service Violations and DoS Attacks." 2002. URL : http://216.239.39.104/search?q=cache:m4uV_rBx9hIJ:www.isoc.org/isoc/conferences/ndss/03/proceedings/papers/12.pdf+detecting+service+violations+and+dos+attacks&hl=en&ie=UTF-8

[34] VanWyk, K. R. and Fomo, R. "Incident Response." *Chapter 7. Tools of the Trade.* August 2001. URL: http://www.oreilly.com/catalog/incidentres/chapter/ch07.html

[35] Refer to [20].

[36] CounterPane Internet Security, Inc. "Managed Security Monitoring: Network Security for the 21st Century." 2003. URL: http://www.counterpane.com/msm.html

[37] Harrison, A. "When Good Scanners Go Bad." 22 March 1999. URL: http://www.computerworld.com/news/1999/story/0,11280,35031,00.html

Additional Appendix

Configuration of Distributed Nagios server to communicate with Prelude Relay Manager

One has to first configure and add Prelude Nagios sensor to the Prelude Relay Manager installed on the distributed Nagios monitoring server (i.e. 192.168.20.3 in this case) seen in *step 1* of *Figure 12*.

In prelude-Nagios.conf:

# Configuration file for Prelude Nagios sensor	
node-name=distributedserver1	# Host Name
node-location=remotesite1.com	# Domain Name
manager-addr=192.168.20.3	# IP Address

```
heartbeat-time=60                    # In seconds
```

The options defined in the prelude-Nagios.conf file are pretty self-explanatory except for the 'heartbeat-time', which is actually simply a way for Prelude Nagios sensor to inform the Prelude Relay Manager that it is working and one can think of it like a human heart that beats every 60 seconds!

After configuration is done, in order to add the Prelude Nagios sensor to the Prelude Relay Manager, one has to find out the encrypted password from last entry of the Manager-adduser log as shown in the following for instance:

```
waiting for install request from Prelude sensors...
Generated one-shot password is "pD:$*49"".

This password will be requested by "sensor-adduser" in order to connect.
Please remove the first and last quote from this password before using it.
waiting for install request from Prelude sensors...
```

This adds the Prelude Nagios sensor to the Prelude Relay Manager listening on the default port #5554, replacing <user-id> with the actual ID of the 'Nagios' user.

```
su -l Nagios
sensor-adduser -s prelude-Nagios -m 192.168.20.3:5554 -u <user-id>
```

The next step is the configuration of Prelude Nagios sensor on the distributed Nagios monitoring server.

In /usr/local/Nagios/etc/generic/commands directory:

In the misccommands.cfg file add in the following:

```
define command{
        command_name    host-notify-by-prelude
        command_line    /usr/Nagios/bin/prelude-Nagios  -N "$HOSTNAME$" -S "$HOSTSTATE$"\
-A "$HOSTADDRESS$" -d "$TIMET$" -C "$OUTPUT$" --manager-addr 192.168.20.3:5554 \        --
heartbeat-time 60
        }

define command{
        command_name    service-notify-by-prelude
        command_line    /usr/Nagios/bin/prelude-Nagios  -t "$NOTIFICATIONTYPE$" -L \
"$HOSTALIAS$" -A "$HOSTADDRESS$" --service -l "$SERVICEDESC$" -m "$SERVICESTATE$"\ -
d "$TIMET$" -C  "$OUTPUT$" --manager-addr 192.168.20.3:5554 --heartbeat-time 60
```

```
        }
```

In /usr/local/Nagios/etc/generic/contacts directory:

Edit the following in the 'SecurityAnalysts' contact definition contacts.cfg to call for the above commands instead of 'service-notify-by-email' and 'host-notify-by-email':

```
service_notification_commands    service-notify-by-prelude
host_notification_commands       host-notify-by-prelude
```

Finally, make sure that the connection between the Prelude Relay Manager and the Prelude Manager already installed on the central Nagios monitoring server is working when the Nagios re-started on the distributed Nagios monitoring server as seen in *step 2* of *Figure 12.* One should see the following in Nagios.log indicating that the service_notification_commands for instance are executed:

```
[Thu Jul 24 22:28:19 2003] SERVICE NOTIFICATION: SecurityAnalysts;server-
192.168.20.1;Load;CRITICAL;service-notify-by-prelude;CRITICAL - load average: 4.02, 3.01, 4.30
```

Followed by a series of Prelude Nagios alerts portrayed in *Figure 13* !

[i] Definition of IDMEF - Refer to article in footnote [9].