Table of Contents

Using NSClient++ from nagios **1**
- Choosing a transport 1
- NSClient (check_nt) 1
- NRPE (check_nrpe) 2
- NRPE and NSClient Server 2
- NSCA (nsca-server) 3
- Make Your Own 3

Using NSClient++ from nagios with check_nt **4**
- 1. Nagios command line 4
- 2. NSClient++ configuration 5
- 3. Solving problems 6
- 4. Nagios configuration 8
- 5. Final words 9

Using NSClient++ from nagios with check_nrpe **10**
- 1. Overview of NRPE 10
- 2. Nagios command line 11
- 3. NSClient++ configuration 11
- 4. Finding and solving problems 12
- 5. NSClient++ configuration (revisited) 14
  - 5.1 NRPE specific setting in NSClient++ 14
  - 5.2 Modules 15
- 6. Nagios command line (revisited) 15
- 7. Nagios configuration 17
  - 7.1 Introduction 17
  - 7.2 Template 18
  - 7.3 Host definition 18
  - 7.4 Service definitions 18
- 8. Where to go next 19

Using NSClient++ from nagios with NSCA **20**
- 1. Overview of NSCA 20
- 2. NSClient++ configuration 20
  - 2.1 Modules 21
  - 2.2 NSCA Configuration 21
  - 2.3 NSCA Commands 23
- 3. NSCA Server 23
- 4. Testing and Debugging 23
- 5. Configure Nagios 25
  - 5.1 Introduction 25
  - 5.2 Passive Checks 26
  - 5.3 Template 27
  - 5.4 Host definition 27
  - 5.5 Service definitions 27
- 6. Where to go next 28

Using NSClient++ from nagios

This is a quick guide over how to use NSClient++ with Nagios. It is divided into four sections (so don't miss the three other pages but this one).

1. Overview (this page)
2. NSClient (check_nt)
3. NRPE (check_nrpe)
4. NSCA

Choosing a transport

NSClient++ supports several transports and you can use either one or several of these or you can create your own cusom transport. Transports are methods which facilitates communication between Nagios and your server. You can look at this much like for instance HTTP (which you are using now) and FTP. They both support transferring files but they have slightly different approaches so things work differently but the end result is the same. A file gets transfered. In our case the end result is that a monitoring result gets submitted to Nagios.

1. **NSClient (check_nt)** Only has some basic checks and is intended for backwards compatibility.

2. **NRPE (check_nrpe)** This is what I would think of as the "normal" or preferred way to use NSClient++. Most examples are intended to be used in this mode.

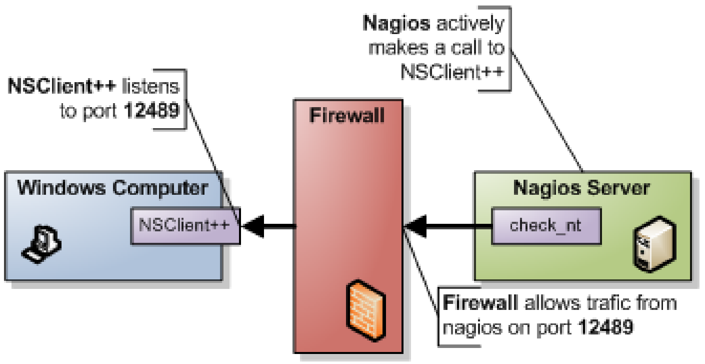
3. **NRPE (check_nrpe) and NSClient (check_nt)** This is what I would think of as the "legacy" way to use NSClient++ in addition to all advanced features you have support for older legacy checks. (in other words ability to migrate without to much change).

4. **NSCA (nsca server)** If you are an "advanced" Nagios user you might want to do passive checking (which is supported from NSClient++). If you don't know what NSCA is you probably don't want to do this.

5. **Make your own** The spirit of NSClient++ is to allow you do decide what you want to do so you can make any combination of the above and even use some other third party protocols or what not...

I would recommend nagios-beginners to starting out with NSClient++ to go with NSClient (since it is simplest to setup) and everyone else (NSClient++ beginner but nagios intermediates) go with NRPE (unless you have specific needs in which case you most likely know enough to choose for you self). And advanced users feel free pick and choose.

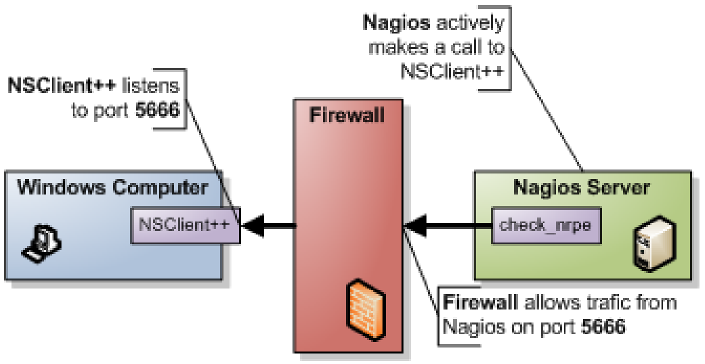
NSClient (check_nt)


Nagios actively makes a call to NSClient++
NSClient++ listens to port 12489
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

This is the simplest and most locked in way to use NSClient++ you are limited to a handful of checks and there is no way to exploit the power of NSClient++ from here. The good though is that it is very simple to use and setup and the configuration is included with nagios so it might be a good way to start. It is also the "only" way to have password protection. But note that since there is no encryption the password is sent as clear text so if you are compromised it will be easy to find. Another option in favor of this is since check_nt is distributed in the "normal plugin kit" you undoubtedly already have everything you need on the Nagios side.

For details on configuring this go to the NSClient setup guide.

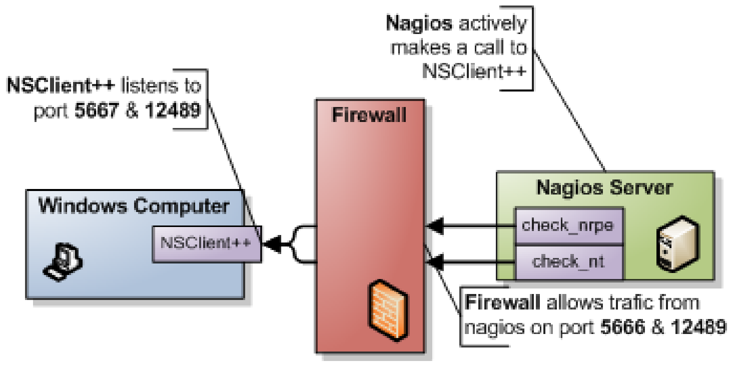
NRPE (check_nrpe)


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

NRPE is the preferred way and, if you ask me, you get the most out of NSClient++ choosing this mode. NRPE works much like NRPE for unix (if you are familiar with it) and in short you can say it relays a plugin request to a remote server. NRPE acts like a simple transport layer allowing remote execution. The difference between regular NRPE and NSClient++ is that NSClient++ has several built-in checks and thus does not require scripts for basic checks. So with NSClient++ you get a lot of ready-to-use checks that wont require you to have scripts. But if you choose you can disable all "modules" and stick with a pure NRPE installation and only external scripts or just use the external script to extend NSClient++ when you need.

For details on configuring this go to the NRPE setup guide.

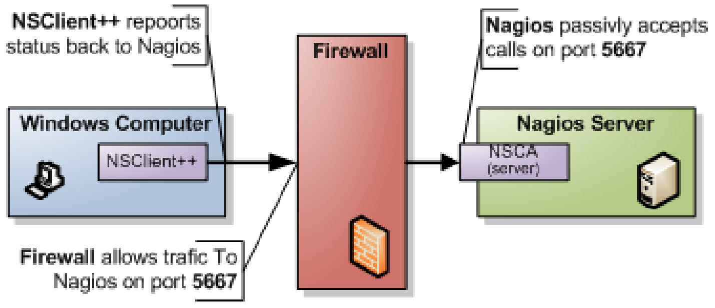
NRPE and NSClient Server


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5667 & 12489
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
check_nt
Firewall allows trafic from nagios on port 5666 & 12489

NSClient++ is built around choice this you can naturally use both NRPE (check_nrpe) and NSClient (check_nt) if you wish and this is the preferred way to migrate from old legacy setup to a more modern one.

For details on configuring this go to both the NRPE setup guide and the NSClient setup guide.

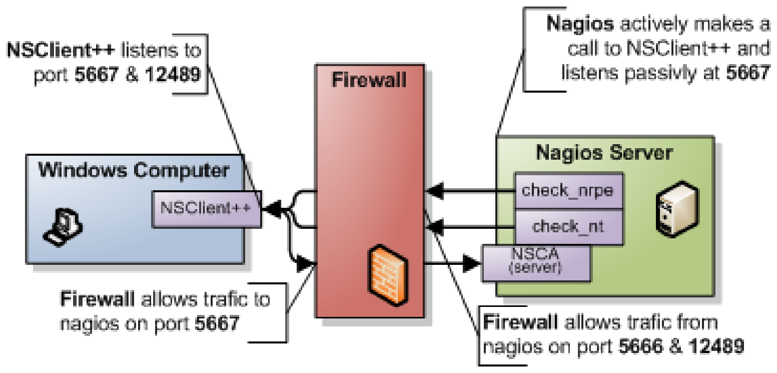
NSCA (nsca-server)


NSClient++ repoorts status back to Nagios
Firewall
Nagios passivly accepts calls on port 5667
Windows Computer
NSClient++
Nagios Server
NSCA (server)
Firewall allows trafic To Nagios on port 5667

Passive checking is the "reversed" of active checking this means that instead of Nagios "calling out" to your (windows) server the (windows) server will phone home to the Nagios server instead. The best way to illustrate this is to compare the NRPE and the NSCA pictures above.

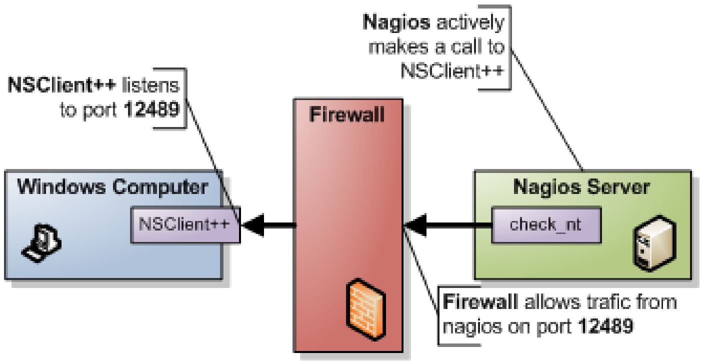
For details on configuring this go to the NSCA setup guide.

Make Your Own


NSClient++ listens to port 5667 & 12489
Firewall
Nagios actively makes a call to NSClient++ and listens passivly at 5667
Windows Computer
NSClient++
Nagios Server
check_nrpe
check_nt
NSCA (server)
Firewall allows trafic to nagios on port 5667
Firewall allows trafic from nagios on port 5666 & 12489

There are several other monitoring protocols out there and you can quite easily with for instance Lua make your own how to do this is outside the scope of this guide but you should know that the "sky is the limit".

Using NSClient++ from nagios with check_nt

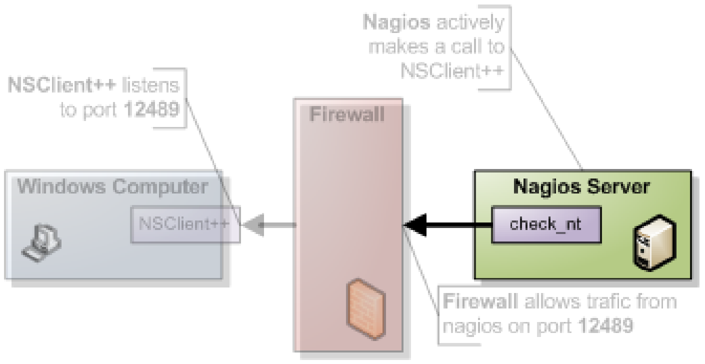

Nagios actively makes a call to NSClient++
NSClient++ listens to port 12489
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

NOTICE The check_nt client support is available for compatibility mode and it is not recommended unless you already have an infrastructure around it. There are several features you will not be able to use with this scheme. I would recommend using NRPE instead.

This is the simplest and most locked in way to use NSClient++ you are limited to a handful of checks and there is no way to exploit the power of NSClient++ from here. The good though is that it is very simple to use and setup so it might be a good way to start. It is also the "only" way to have password protection. But note that since there is no encryption the password is sent as clear text so if you are compromised it will be easy to find. Also since check_nt is distributed in the "normal plugin kit" you undoubtedly already have everything you need on the nagios side.

Nagios have their own guide for setting this up here
http://nagios.sourceforge.net/docs/3_0/monitoring-windows.html

1. Nagios command line


Nagios actively makes a call to NSClient++
NSClient++ listens to port 12489
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

Using check_nt from the command line of your Nagios server is usually the bast place to start. If you are not familiar with it I would recommend you try this out as it will save you a lot of time when you are getting started or trying out new things. It is a good way to eliminate errors and you wont have to bother with restarting/waiting on Nagios when you need to make changes. To access NSClient++ from the Nagios server via the NSClient protocol you use a program (comes with the default plugins) called check_nt.

```
check_nt -H <client ip> -p <port> -v <command> ...
```

- client ip = the IP of the server you want to monitor (i.e. where NSClient++ i installed).

- port = the port you are using for the NSClientListener (defaults to 12489)
- command = is the various things you can monitor. The various commands all take different additional arguments which are all showed in the help.

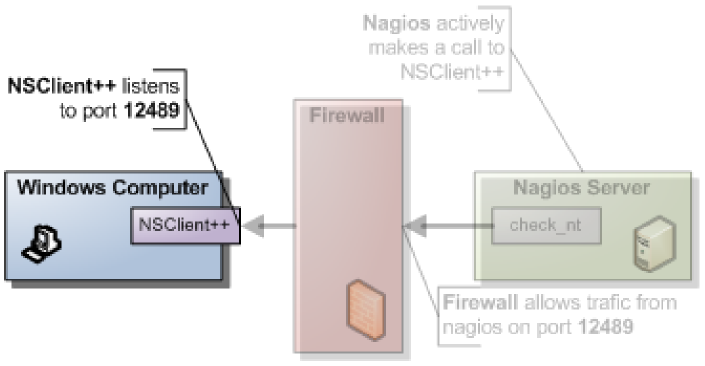
To check the CPU load you can for instance run the following (assuming your windows server has 10.0.0.1 as ip address)

```
check_nt -H 10.0.0.1 -p 12489 -v CPULOAD -w 80 -c 90 -l 5,80,90,10,80,90
CPU Load 0% (5 min average) 0% (10 min average)
   |'5 min avg Load'=0%;80;90;0;100 '10 min avg Load'=0%;80;90;0;100
```

If you instead got the following don't worry, it is because your NSClient++ is not configured properly and, we will solve that in the next section.

```
CRITICAL - Socket timeout after 10 seconds
```

2. NSClient++ configuration


NSClient++ listens to port 12489
Nagios actively makes a call to NSClient++
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

The first thing you need to do is decide which modules you want to use. NSClient++ is modular by design this means you only use the features you want (and if you want you can use all of them). The modules can be roughly divided into two kinds.

1. check commands
2. protocols (and utility modules).

The first kind is the one you *use* it responds to your commands and "finds" monitored data for you. The second kind is the one that allows you to talk to the first kind. When it comes to modules for the NSClient mode you will need the following:

Module	Description	Commands
CheckSystem.dll	Handles many system checks	CPU, MEMORY, COUNTER etc
CheckDisk.dll	Handles Disk related checks	USEDDISKSPACE
FileLogger.dll	Logs errors to a file so you can see what is going on	N/A
NSClientListener.dll	Listens and responds to incoming requests from nagios	N/A

To enable modules you edit the [modules] section in the nsc.ini file and your section should look something like this:

```
[modules]
CheckSystem.dll
CheckDisk.dll
FileLogger.dll
NSClientListener.dll
```

The other things you need to configure is who is allowed to ask questions (which ip addresses) this is done either under the [Settings] section (globally) or under the [NSClient] (locally). I would recommend using the [Settings] section as it will simplify things when you start using NRPE. The keys you need to change are allowed_hosts and password. And the value should be:

- allowed_hosts = A list of addresses that is allowed to ask questions (i.e. your nagios ip).
- password = The password to use.

The result should look like this (assuming you don't use a password and the nagios ip address is 10.0.0.2):

```
[Settings]
;password=secret-password
allowed_hosts=10.0.0.2
```

Notice that since you don't use a password that key is commented out (;).

Don't forget to restart NSClient++ after you make changes to the NSC.ini file.

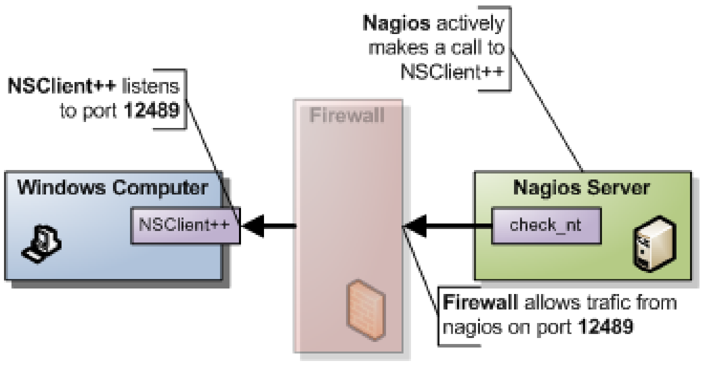
```
nsclient++ /stop
nsclient++ /start
... or ...
net stop nsclientpp
net start nsclientpp
```

Now feel free to try the command line agent again and hopefully things should work out perfectly. Run the following command from your nagios server.

```
check_nt -H 10.0.0.1 -p 12489 -v CPULOAD -w 80 -c 90 -l 5,80,90,10,80,90
CPU Load 0% (5 min average) 0% (10 min average)
  |'5 min avg Load'=0%;80;90;0;100 '10 min avg Load'=0%;80;90;0;100
```

```
check_nt -H 10.0.0.1 -p 12489 -v USEDDISKSPACE -d SHOWALL -l c
c:\ - total: 149.00 Gb - used: 12.93 Gb (9%) - free: 136.07 Gb (91%)
  |'c:\ Used Space'=12.93Gb;0.00;0.00;0.00;149.00
```

3. Solving problems



Nagios actively makes a call to NSClient++
NSClient++ listens to port 12489
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

A good way to find and solve problems is to run nsclient++ in "test" mode this is done by stopping the service and starting it in "test" mode.

```
nsclient++ /stop
nsclient++ /test
... test mode ... (quit with: exit)
nsclient++ /start
```

When in test mode you will get a lot of interesting log messages when things are happening so it is fairly simple to figure out what is wrong. To try this out do the following:

```
nsclient++ /stop
nsclient++ /test
```

What you will see is the following output (or something similar):

```
Launching test mode - client mode
d NSClient++.cpp(1106) Enabling debug mode...
d NSClient++.cpp(494) Attempting to start NSCLient++ - 0.3.7.7 2009-07-05
d NSClient++.cpp(897) Loading plugin: CheckSystem...
d NSClient++.cpp(897) Loading plugin: NSClient server...
d \PDHCollector.cpp(66) Autodetected w2k or later, using w2k PDH counters.
l NSClient++.cpp(600) NSCLient++ - 0.3.7.7 2009-07-05 Started!
d \PDHCollector.cpp(103) Using index to retrive counternames
d \Socket.h(675) Bound to: 0.0.0.0:12489
l NSClient++.cpp(402) Using settings from: INI-file
l NSClient++.cpp(403) Enter command to inject or exit to terminate...
d \PDHCollector.cpp(123) Found countername: CPU:    \Processor(_total)\% processortid
d \PDHCollector.cpp(124) Found countername: UPTIME: \System\Tid sedan systemstart
d \PDHCollector.cpp(125) Found countername: MCL:    \Minne\Dedikationsgrõns
d \PDHCollector.cpp(126) Found countername: MCB:    \Minne\Dedicerade byte
```

Then when you run the check from Nagios again:

```
check_nt -H 10.0.0.1 -p 12489 -v USEDDISKSPACE -d SHOWALL -l c
c:\ - total: 149.00 Gb - used: 12.93 Gb (9%) - free: 136.07 Gb (91%)
  |'c:\ Used Space'=12.93Gb;0.00;0.00;0.00;149.00
```

If you check the log from NSCLient++ you should see (amongst other):

```
d \NSClientListener.cpp(146) Data: None&2&5
d \NSClientListener.cpp(171) Data: 5
d NSClient++.cpp(1034) Injecting: checkCPU: 5, nsclient
d NSClient++.cpp(1070) Injected Result: OK '0'
```

```
d NSClient++.cpp(1071) Injected Performance Result: ''
```

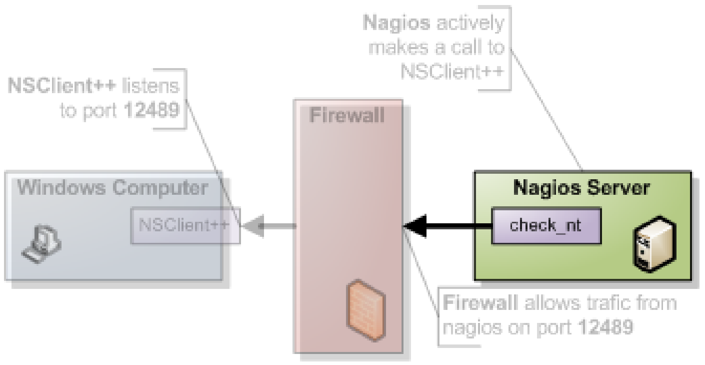
When you are don you can exit NSClient++ using the exit command:

```
exit
```

Then don't forget to start NSClient++ again:

```
nsclient++ /start
```

4. Nagios configuration


NSClient++ listens to port 12489
Nagios actively makes a call to NSClient++
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

Nagios comes pre-configured for many of the NSClient checks. in windows.cfg you will find many entries along the lines of:

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     NSClient++ Version
        check_command           check_nt!CLIENTVERSION
}
```

The interesting part here is: **'check_nt!CLIENTVERSION**' which will run a check against check_nt. In commands.cfg the check_nt command is defined like so:

```
# 'check_nt' command definition
define command{
        command_name    check_nt
        command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -v $ARG1$ $ARG2$
}
```

So you can see most things are already setup for you so it is quite simple to get started. The more "advanced" checks (which takes parameters) looks like this if you recall the CPULOAD we tried from the command line:

```
define service{
        use                     generic-service
        host_name               winserver
        service_description     CPU Load
        check_command           check_nt!CPULOAD!-l 5,80,90
}
```

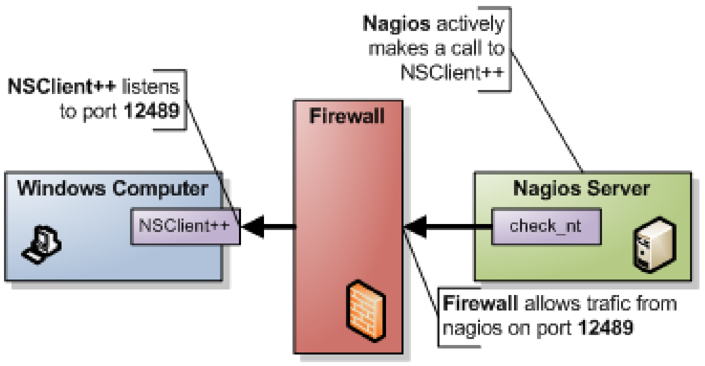
the command is now defined as **'check_nt!CPULOAD!-l 5,80,90**' which translates directly into:

```
<plugin dir>/check_nt -H <ip of client> -p 12489 -v CPULOAD -l 5,80,90
```

which if you recall is exactly what we used when we tried the command from the command line. If you want to add a password the simplest way is to add it in command.cfg (if you want to have the same password on all your clients) like so:

```
# 'check_nt' command definition
define command{
        command_name    check_nt
        command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -s <password> -v $ARG1$ $ARG
}
```

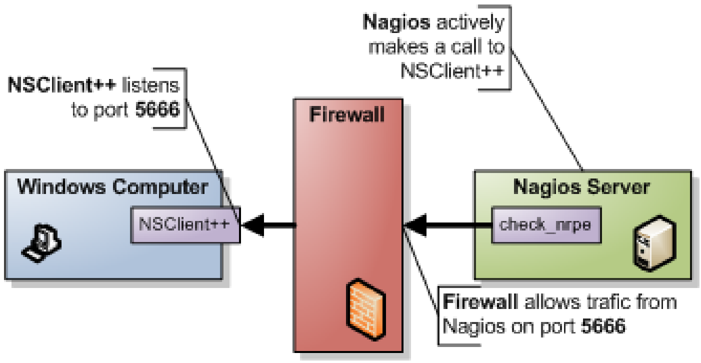
5. Final words


Nagios actively makes a call to NSClient++
NSClient++ listens to port 12489
Firewall
Windows Computer
NSClient++
Nagios Server
check_nt
Firewall allows trafic from nagios on port 12489

As I stated initially using check_nt is limited and many checks (for instance EventLog?) wont work this way. So a good idea is probably to start checking out the NRPE Guide as well.

And remember if you experience problems don't "debug" from nagios, run your command from the command line while having nsclient++ running in /test mode and you should be fine!

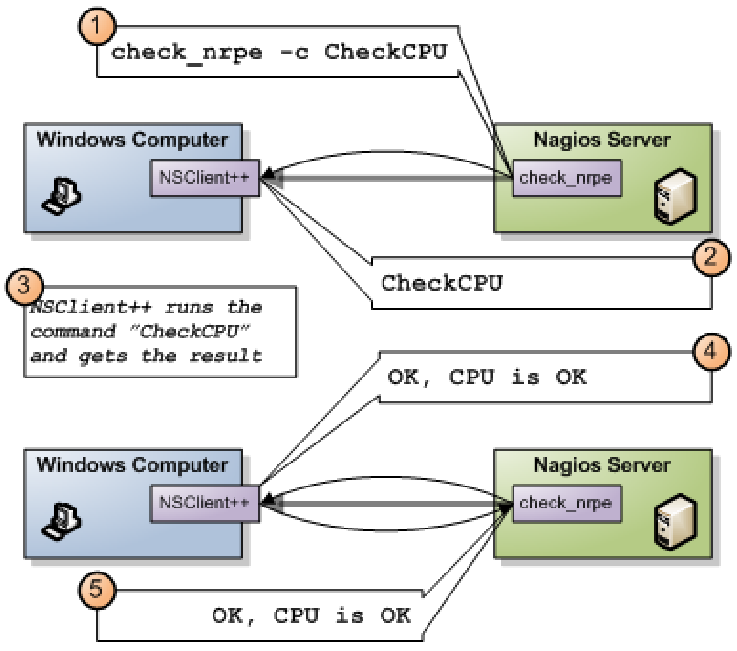
Using NSClient++ from nagios with check_nrpe


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

NRPE is the preferred way over NSClient (check_nt) and you get the most out of NSClient++ choosing this mode (NSCA and what not will support the same commands but are more complex to setup). NRPE works much like NRPE for unix (if you are familiar with it) and in short you can say it relays a plugin request to a remote server. NRPE acts like a simple transport layer allowing remote execution. The difference between regular NRPE and NSClient++ is that NSClient++ has built-in checks. So with NSClient++ you get a lot of ready-to-use checks that wont require you to have scripts. But if you choose you can disable all "modules" and stick with a pure NRPE installation and only external scripts.

1. Overview of NRPE

For those not familiar with NRPE (Nagios Remote Plugin Execution) here is a quick introduction.



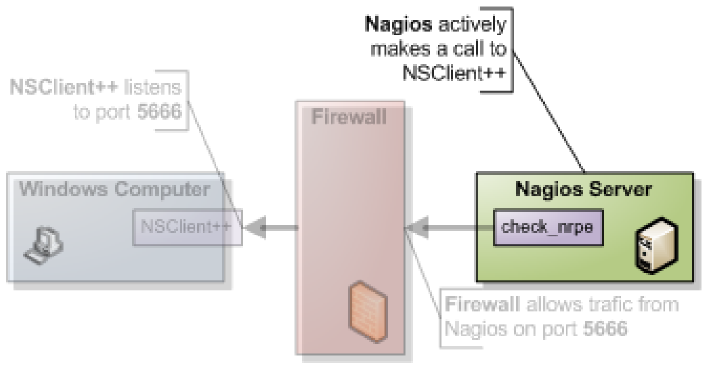
NRPE works much like SSH or telnet etc. It relays a command and awaits the result. In the above diagram what happens is:

1. Nagios executes check_nrpe with the proper arguments.
2. NSClient++ receives the command to be executed

3. NSClient++ will execute the command and get a result on the form of <status>, <message> and optionally <performance data>
4. NSClient++ sends the result back to Nagios
5. Nagios gets the result from check_nrpe (and uses it much like any other plugin)

So in essence NRPE is merely a transport mechanism to send the result of a check command over the network.

2. Nagios command line



Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

NRPE require you to install a special plug-in on your nagios server called NRPE. The unix-side of NRPE consists of a server and a client on nagios you only need the client so you can skip any "servers" or what not that it want to start when you install it.

The client is (generally) called check_nrpe and works like so:

```
./check_nrpe -H <nsclient++ server ip> -c <command> [-a <a> <list> <of> <arguments>]
```

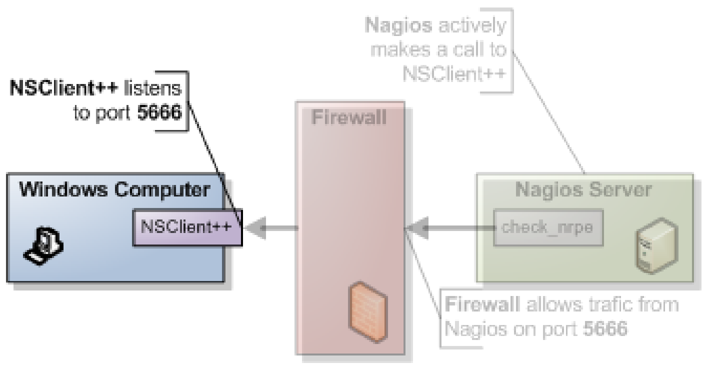
- <command> = The command (script) you want to run (often this is a pre-built command from within NSClient++)
- <a> <list> <of> <arguments> = a list of arguments for the command.

So the simplest way to see if things are a-working just run it without a command and you should get a response specifying the version of "NRPE" (in this case NSClient++) like so:

```
./check_nrpe -H <nsclient++ server ip>
I (0.3.3.19 2008-07-02) seem to be doing fine...
```

And again like in the NSClient example don't worry if you get a timeout here since we have to configure NSClient++ before it actually works so this is expected.

3. NSClient++ configuration


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

Configuring NRPE is a bit more involved but not overly so. The first thing you need to do to get things working is add the NRPEListener module.

```
[modules]
...
NRPEListener.dll
...
```

If you have not already done so (above) you also need to set which computers are allowed to query the agent. This is set either under the [Settings] section (globally) or under the [NRPE] section (locally). If you when you configured NSClient above set this globally you are already set to go. If not the key you need to change is the allowed_hosts. There is no password for NRPE.

- allowed_hosts = A list of addresses that is allowed to ask questions (i.e. your nagios ip).

The result should look like this (assuming your nagios server ip address is 10.0.0.2):

```
[Settings]
allowed_hosts=10.0.0.2
```

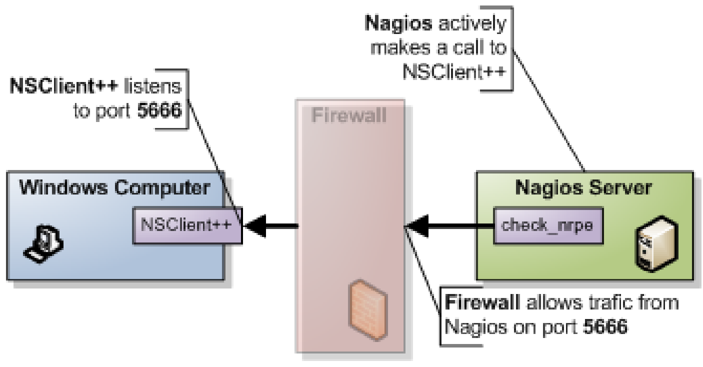
After this restart the service.

```
nsclient++ /stop
nsclient++ /start
... or ...
net stop nsclientpp
net start nsclientpp
```

Now feel free to try the command line agent again and hopefully things should work out perfectly. Run the following command from your nagios server.

```
./check_nrpe -H 10.0.0.1
I (0.3.3.19 2008-07-02) seem to be doing fine...
```

4. Finding and solving problems


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

A good way to find and solve problems is to run nsclient++ in "test" mode this is done by stopping the service and starting it in "test" mode.

```
nsclient++ /stop
nsclient++ /test
... test mode ... (quit with: exit)
nsclient++ /start
```

When in test mode you will get a lot of interesting log messages when things are happening so it is fairly simple to figure out what is wrong. So lets try this now: Start NSClient++ in test mode like so:

```
nsclient++ /stop
nsclient++ /test
```

And you should see something along the following lines (it will look different depending on your setup):

```
Launching test mode - client mode
d NSClient++.cpp(1106) Enabling debug mode...
d NSClient++.cpp(494) Attempting to start NSCLient++ - 0.3.7.7 2009-07-05
d NSClient++.cpp(897) Loading plugin: NRPE server (w/ SSL)...
d \NRPEListener.cpp(91) Loading all commands (from NRPE)
d \NRPEListener.cpp(121) Starting NRPE socket...
l NSClient++.cpp(600) NSCLient++ - 0.3.7.7 2009-07-05 Started!
d \Socket.h(675) Bound to: 0.0.0.0:5666
l NSClient++.cpp(402) Using settings from: INI-file
l NSClient++.cpp(403) Enter command to inject or exit to terminate...
```

Now you can run the the command again from Nagios like so:

```
./check_nrpe -H 10.0.0.1
I (0.3.7.7 2009-07-05) seem to be doing fine...
```

And if you check the log of NSClient++ /test you will this time not see anything and this is because the "check version" is an internal command so lets try with something slightly more interesting:

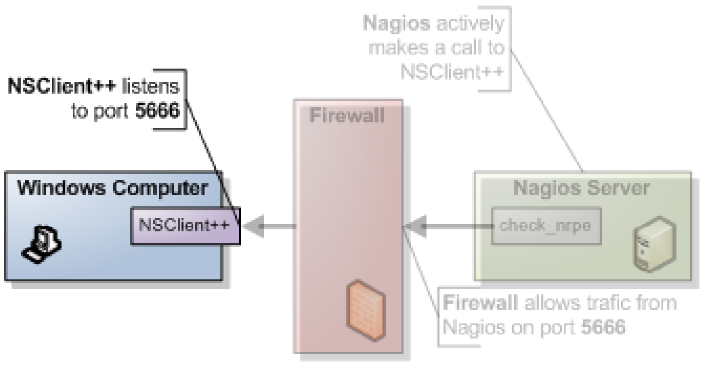
```
./check_nrpe -H 10.0.0.1 -c foobar
UNKNOWN: No handler for that command
```

And don't worry there is no foobar command but we will see how this looks in NSClient++

```
d NSClient++.cpp(1034) Injecting: foobar:
l NSClient++.cpp(1085) No handler for command: 'foobar'
l \NSCHelper.cpp(238) No handler for command 'foobar'.
```

We shall get back a bit to this later on when we have configure NSClient++ more so lets leave this for now.

5. NSClient++ configuration (revisited)


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

As we said before it is a bit more involved to configure NRPE and yet thus far it has actually been simpler? This is because we have not configured anything yet all we can do now is talk to NSClient++ but not actually use it. So in this section we shall cover the basics and first off are some of the configuration options available for NRPE

5.1 NRPE specific setting in NSClient++

- use_ssl

  If this is 1 (true) we will use SSL encryption on the transport. **Notice** this flag has to be the same on both ends or you will end up with strange errors. The flag is set on check_nrpe with the -n option (if you use -n no SSL will be used).

- allow_arguments

  Since arguments can be potentially dangerous (it allows your users to control the execution) there is a flag (which defaults to off) to enable arguments. So if you plan on configure NSClient++ from the Nagios end you need to enable this. **But be warned this is a security issue you need to think about**. If you do not want to allow arguments you can instead configure all checks in the NSC.ini file and just execute the aliases from nagios.

  One important issue with the **allow_arguments** is that there are more then one! **Yes, more then one!** The reason for this is that you can allow arguments from NRPE and you can allow arguments for external scripts (it is not the same option) which might seem a bit confusing at first.

- allow_nasty_meta_chars

  This flag allows arguments to contain "dangerous" characters such as redirection and pipe (<>|) and makes things a tad more dangerous. But if you decide to use arguments you most likely want to use this flag as well. **But again this is a security risk**

So this if you enable this in the INI file you will end up with something like this (extract):

```
[NRPE]
```

```
;# COMMAND ARGUMENT PROCESSING
;  This option determines whether or not the NRPE daemon will allow clients to specify
;   arguments to commands that are executed.
allow_arguments=1
;
;# COMMAND ALLOW NASTY META CHARS
;  This option determines whether or not the NRPE daemon will allow clients
;    to specify nasty (as in |`&><'"\[]{}) characters in arguments.
allow_nasty_meta_chars=1
;
;# USE SSL SOCKET
;  This option controls if SSL should be used on the socket.
use_ssl=1
```

There are a lot of other options as well but these are the most used ones.

5.2 Modules

The other thing which you should configure is which modules to use. There is (at time of writing) 16 modules to choose from of which 9 will give you more "checks to run" so choosing which you need can be a bit of work. Here we shall start out with the basic ones and for details on the rest check out the Modules section in the wiki.

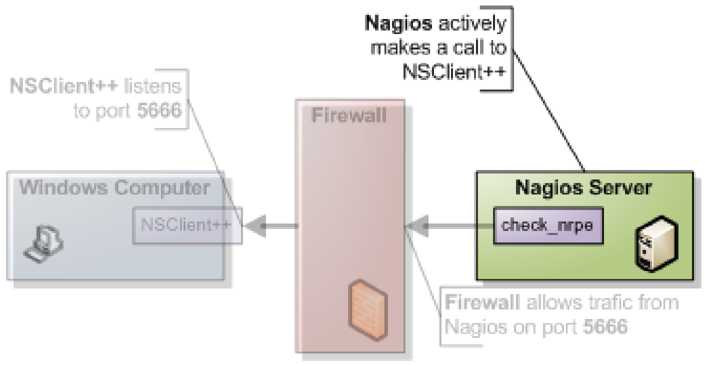
Module	Description	Commands
CheckSystem.dll	Handles many system checks	CheckCPU, CheckMEM etc
CheckDisk.dll	Handles Disk related checks	CheckDisk
CheckExternalScripts.dll	Handles aliases (which is what we will use) and external scripts.	N/A
FileLogger.dll	Logs errors to a file so you can see what is going on	N/A
NRPEListener.dll	Listens and responds to incoming requests from Nagios via NRPE	N/A

The finished modules section from the INI file will look like so:

```
[modules]
CheckSystem.dll
CheckDisk.dll
CheckExternalScripts.dll
FileLogger.dll
NRPEListener.dll
```

Now we have done some basic setup of NSClient++ and we can continue to try using it a bit more before we continue with configuring Nagios.

6. Nagios command line (revisited)


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

Now that we have the agent up and running (if not probably want to go back over the previous sections to get it up and running before reading on) what can we do with it?. From here on we will assume you have allow arguments and metchars enabled since it makes it simpler to try things out **BEWARED** that there are security implication to this so might wanna read up before rolling this configuration into production.

As we stated before check_nrpe is a lot more powerful then the legacy check_nt and there is a lot of built in commands as well as a lot of external ones you can use. The built in ones are listed below.

- CheckAlwaysCRITICAL (check)
- CheckAlwaysOK (check)
- CheckAlwaysWARNING (check)
- CheckCPU (check)
- CheckCRITICAL (check)
- CheckCounter (check)
- CheckEventLog/CheckEventLog (check)
- CheckFile (check)
- CheckFileSize (check)
- CheckMem (check)
- CheckMultiple (check)
- CheckOK (check)
- CheckProcState (check)
- CheckServiceState (check)
- CheckTaskSched/CheckTaskSched (check)
- CheckUpTime (check)
- CheckVersion (check)
- CheckWARNING (check)
- CheckWMI/CheckWMI (check)
- CheckWMIValue (check)

Lets start with a simple one CheckCPU and see how to use it.

If we check the docs for it it has an example like so:

```
checkCPU warn=80 crit=90 time=20m time=10s time=4
CPU Load ok.|'20m average'=11%;80;90;
  '10s average'=7%;80;90; '4 average'=10%;80;90;
```

Now this is a "NSCLient++ /test mode command" so it is not usable in it self for you instead you need to change it slightly. The first word is the command and the rest are arguments. check_nrpe has two options for settings commands (-c) and arguments (-a) and is used like so:

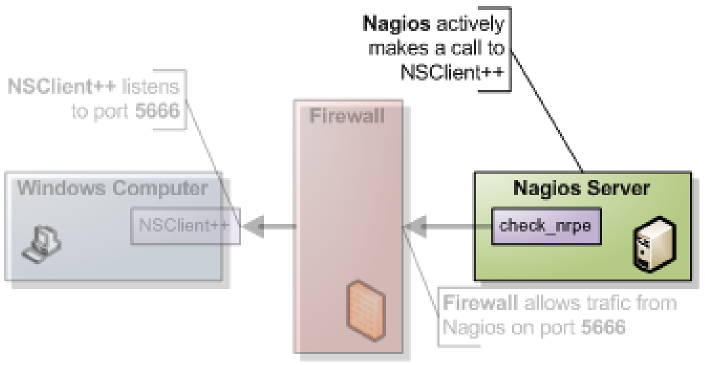
```
check_nrpe ... -c <command> [-a <argument> <argument> <argument>]
```

in this case (CheckCPU) this translates to:

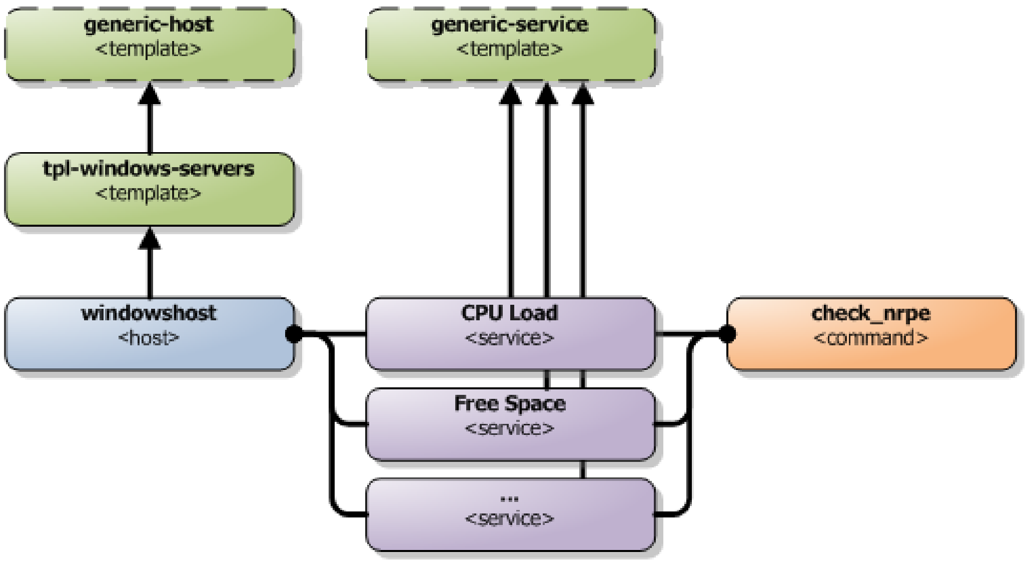
```
check_nrpe ... -c CheckCPU -a warn=80 crit=90 time=20m time=10s time=4
CPU Load ok.|'20m average'=11%;80;90; '10s average'=7%;80;90; '4 average'=10%;80;90;
```

And that is as hard as it gets all you need to do is figure out which arguments you want to use for the command and stack them all in a long line.

7. Nagios configuration


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

7.1 Introduction



Nagios configuration is in itself a whole chapter and this is just a quick peek on how you can do things. First off there are a few concepts to understand:

- templates are the same as the corresponding item but they have a flag register = 0 which makes them "unlistable items"
- services are essentially checks (is check CPU)
- hosts are essentially computers
- groups are an important concept which I ignore here for simplicity (I recommend you use it)

The configuration is at the end layer quite simple you have a "check" and a "host" and you connect them with a service. Like I show at the bottom line in the diagram above. Whats makes this a tad more complicated is that you can inherit things from a "parent" definition. Which is what I show with arrows (bottom to top) above. The templates with dashed lines are the base templates which all services and hosts inherit.

7.2 Template

First, its best practice to create a new template for each different type of host you'll be monitoring. Let's create a new template for windows servers.

```
define host{
        name                    tpl-windows-servers ; Name of this template
        use                     generic-host ; Inherit default values
        check_period            24x7
        check_interval          5
        retry_interval          1
        max_check_attempts      10
        check_command           check-host-alive
        notification_period     24x7
        notification_interval   30
        notification_options    d,r
        contact_groups          admins
        register                0 ; DONT REGISTER THIS - ITS A TEMPLATE
}
```

Notice that the tpl-windows-servers template definition is inheriting default values from the generic-host template, which is defined in the sample localhost.cfg file that gets installed when you follow the Nagios quickstart installation guide.

7.3 Host definition

Next we need to define a new host for the remote windows server that references the newly created tpl-windows-servers host template.

```
define host{
        use             tpl-windows-servers ; Inherit default values from a template
        host_name       windowshost ; The name we're giving to this server
        alias           My First Windows Server ; A longer name for the server
        address         10.0.0.2 ; IP address of the server
}
```

Defining a service for monitoring the remote Windows server. These example service definitions will use the sample commands that are defined in the default NSC.ini file which ships with NSClient 0.3.7 or newer.

7.4 Service definitions

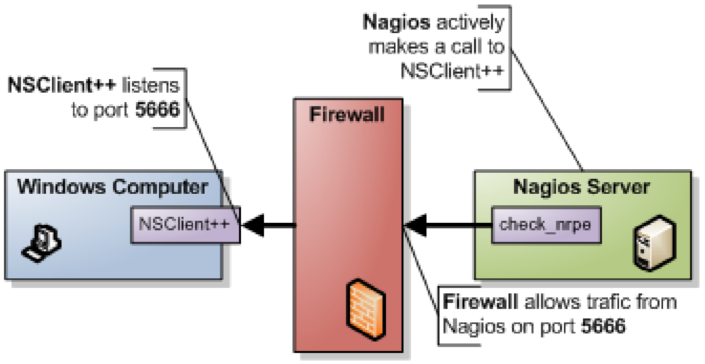
The following service will monitor the CPU load on the remote host. The "alias_cpu" argument which is passed to the check_nrpe command definition tells NSClient++ to run the "alias_cpu" command as defined in the alias section of the NSC.ini file.

```
define service{
        use                     generic-service
        host_name               windowshost
        service_description     CPU Load
        check_command           check_nrpe!alias_cpu
}
```

The following service will monitor the free drive space on /dev/hda1 on the remote host.

```
define service{
        use                     generic-service
        host_name               windowshost
        service_description     Free Space
        check_command           check_nrpe!alias_disk
}
```

8. Where to go next

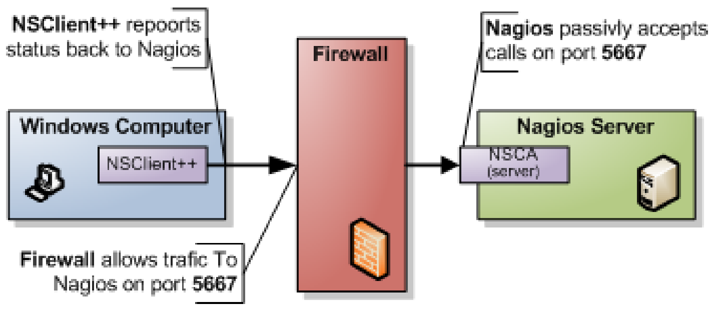

Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

This is of cores not the end now you need to check out what checks you want to use run on your servers. There is a lot of built-in checks but there are a lot more external scripts you can use and download from for instance monitoring exchange or the new nagios exchange.

Built in checks:

- CheckAlwaysCRITICAL (check)
- CheckAlwaysOK (check)
- CheckAlwaysWARNING (check)
- CheckCPU (check)
- CheckCRITICAL (check)
- CheckCounter (check)
- CheckEventLog/CheckEventLog (check)
- CheckFile (check)
- CheckFileSize (check)
- CheckMem (check)
- CheckMultiple (check)
- CheckOK (check)
- CheckProcState (check)
- CheckServiceState (check)
- CheckTaskSched/CheckTaskSched (check)
- CheckUpTime (check)
- CheckVersion (check)
- CheckWARNING (check)
- CheckWMI/CheckWMI (check)
- CheckWMIValue (check)

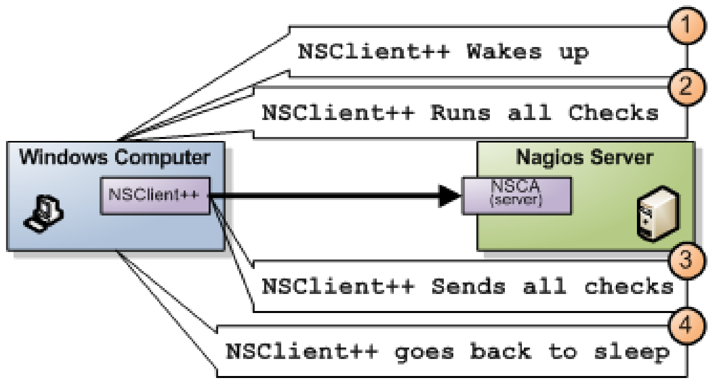
Using NSClient++ from nagios with NSCA


NSClient++ repoorts status back to Nagios
Firewall
Nagios passivly accepts calls on port 5667
Windows Computer
NSClient++
Nagios Server
NSCA (server)
Firewall allows trafic To Nagios on port 5667

NSCA (Nagios Service Check Acceptor) is a server which runs on the Nagios server and accepts passive checks results from various servers.

Passive in this context means that Nagios is not the initiator of the actual check commands above. Instead the client (when it is configure to do so) will submit the results to Nagios (thus it will initiate the data transfer). If you compare the above image with the one used with NRPE you will notice that the arrow points from the client to Nagios whereas the NRPE one points from Nagios to your client.

1. Overview of NSCA

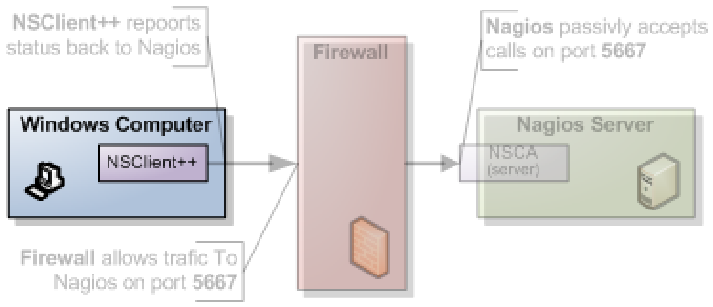

1
NSClient++ Wakes up
2
NSClient++ Runs all Checks
Windows Computer
NSClient++
Nagios Server
NSCA (server)
3
NSClient++ Sends all checks
4
NSClient++ goes back to sleep

As I stated before NSCA is "sort of the reverse" of NRPE and the diagram above illustrates the process by which Nagios receives the check results.

1. NSClient++ decides it is time to send the results
2. NSClient++ gathers all results
3. NSClient++ connect to NSCA (server) and sends all results
4. NSClient++ goes back to sleap

So in essence NSCA is (again) merely a transport mechanism to send the result of a check command over the network. But the big change is that this time it is NSClient++ who decides when it is time to do so.

2. NSClient++ configuration


NSClient++ repoorts status back to Nagios
Windows Computer
NSClient++
Firewall
Nagios passivly accepts calls on port 5667
Nagios Server
NSCA (server)
Firewall allows trafic To Nagios on port 5667

Since NSCA is a server we shall start by configuring NSClient++ as thats were most things will happen. Also since this is an "advanced" guide it is assumed you have read at least the NRPE guide and are familiar with the basic working of both Nagios and NSClient++.

2.1 Modules

The first thing you do is to make sure you have all the proper modules loaded. The basic ones we will need for basic checks in addition to the NSCAAgent. One important thing to notice is that once the NSCAAgent is loaded it will start (attempting) to submit passive check results. This means that if it not properly configured it will result in a lot of error messages.

So lets start with the following modules:

Module	Description	Commands
CheckSystem.dll	Handles many system checks	CheckCPU, CheckMEM etc
CheckDisk.dll	Handles Disk related checks	CheckDisk
CheckExternalScripts.dll	Handles aliases (which is what we will use) and external scripts.	N/A
CheckHelpers.dll	Handles various "utility" checks like CheckOK	CheckOK (amongst others)
FileLogger.dll	Logs errors to a file so you can see what is going on	N/A
NSCAAgent.dll	Submits passive checks results to NSCA (server) on Nagios	N/A

The resulting modules section in NSC.ini will look like so:

```
[modules]
CheckSystem.dll
CheckDisk.dll
CheckExternalScripts.dll
CheckHelpers.dll
FileLogger.dll
NSCAAgent.dll
```

2.2 NSCA Configuration

Then we move on to configure NSCA which is not that hard a quick overview of the basic settings you need to edit:

interval

Perhaps the most important option. It controls the interval which NSClient++ will use when it runs the checks in essence this is the amount of time between a check will be submitted to Nagios (via NSCA). Since there is only one of these it will not be possible to have individual intervals for various checks instead all checks will be submitted using this interval. It is a good idea to set this **LOW** when you are debugging things as you will have to wit for this to fire before anything happens.

encryption_method

The encryption algorithm to use. It is often a good idea to set this to 0 (None) when you try this out as it will reduce the number things which might be broken. If you have the incorrect one it will be hard to know what is wrong. For production I would recommend using 14 (AES) at is it a fairly strong algorithm.

password

The password is the "secret" you share with NSCA it has to be the same on both ends (or again like with encryption) nothing will work.

nsca_host

This is the IP address of the NSCA server (often the same as the Nagios server). This will **not** default to the allowed_hosts directive so you HAVE to specify this option.

The resulting configuration will look something like this:

```
;# CHECK INTERVALL (in seconds)
;   How often we should run the checks and submit the results.
interval=10
;
;# ENCRYPTION METHOD
;   This option determines the method by which the send_nsca client will encrypt the packets it sends
;   to the nsca daemon. The encryption 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 an encryption method.
;
; Note: The encryption method you specify here must match the decryption method the nsca daemon uses
;       (as specified in the 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
;       6 = xTEA
;       8 = BLOWFISH
;       9 = TWOFISH
;       11 = RC2
;       14 = RIJNDAEL-128 (AES)
;       20 = SERPENT
encryption_method=0
;
;# ENCRYPTION PASSWORD
;  This is the password/passphrase that should be used to encrypt the sent packets.
password=secret-password
;
;# NAGIOS SERVER ADDRESS
;  The address to the nagios server to submit results to.
nsca_host=192.168.0.1
```

2.3 NSCA Commands

Now we (hopefully) have configure NSCA which will work splendidly but untill we add some checks it wont actually do anything. Checks for NSCA is added under the NSCA Commands section. The syntax of this section is <service definition>=<check command>.

service definition
: The service definition is the name of the service **IN Nagios**.

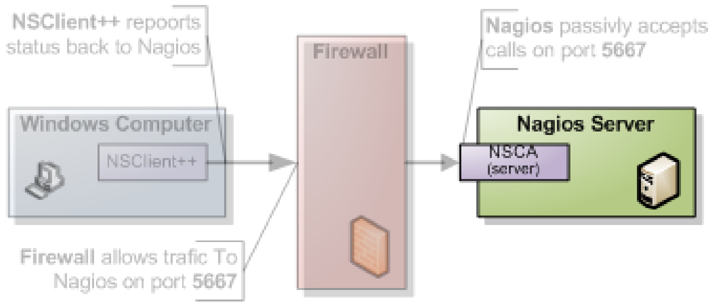
check command
: The check command is the command to run inside NSClient++

There is also a special check called host_check which will correspond to the "host" check command. All commands supported by NSClient++ can be used here which (apart from the commands listed on this site) includes all external scripts you define using the ExternalScripts? module.

The resulting section will look something like this:

```
[NSCA Commands]
CPU Load=alias_cpu
host_check=check_ok
```

3. NSCA Server

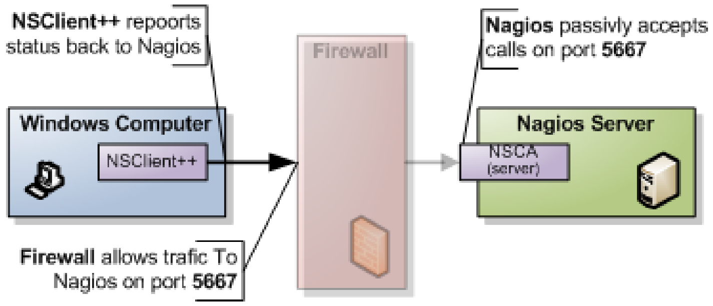

NSClient++ repoorts status back to Nagios
Windows Computer
NSClient++
Firewall
Nagios passivly accepts calls on port 5667
Nagios Server
NSCA (server)
Firewall allows trafic To Nagios on port 5667

How to configure NSCA falls a bit outside the scope of this tutorial but it is pretty straight forward and a quick walk through is provided here.

Don't forget the "debug=1" in /etc/nsca.conf

TODO

4. Testing and Debugging


NSClient++ repoorts status back to Nagios
Windows Computer
NSClient++
Firewall
Nagios passivly accepts calls on port 5667
Nagios Server
NSCA (server)
Firewall allows trafic To Nagios on port 5667

Now lets fire this baby up and see what it can do. As always we will start with with running NSClient++ in /test mode like so:

```
NSClient++ /stop
NSClient++ /test
```

The usual output when NSClient++ boots:

```
Launching test mode - client mode
d NSClient++.cpp(1106) Enabling debug mode...
d NSClient++.cpp(494) Attempting to start NSCLient++ - 0.3.7.7 2009-07-05
d NSClient++.cpp(897) Loading plugin: Helper function...
d NSClient++.cpp(897) Loading plugin: NSCAAgent (w/ encryption)...
d \NSCAThread.cpp(77) Time difference for NSCA server is: 0
d \NSCAThread.cpp(84) Only reporting: ok,warning,critical,unknown
d \NSCAThread.cpp(102) Autodetected hostname: DESKTOP
l NSClient++.cpp(600) NSCLient++ - 0.3.7.7 2009-07-05 Started!
d \NSCAThread.cpp(171) Drifting: 0
l NSClient++.cpp(402) Using settings from: INI-file
l NSClient++.cpp(403) Enter command to inject or exit to terminate...
```

Here we will have to wait as NSClient++ (in my example I set the interval to 10 second so I will wait for 10 seconds. Then we get something along the following lines:

```
d \NSCAThread.cpp(252) Looked up 192.168.0.1 to 192.168.0.1
d \NSCAThread.cpp(297) Finnished sending to server...
d \NSCAThread.cpp(189) Executing (from NSCA): CPU Load
d NSClient++.cpp(1034) Injecting: alias_cpu:
d NSClient++.cpp(1034) Injecting: checkCPU: warn=80, crit=90, time=5m, time=1m, time=30s
d NSClient++.cpp(1070) Injected Result: OK 'OK CPU Load ok.'
d NSClient++.cpp(1071) Injected Performance Result: ''5m'=1%;80;90; '1m'=3%;80;90; '30s'=2%;80;90; '
d NSClient++.cpp(1070) Injected Result: OK 'OK CPU Load ok.'
d NSClient++.cpp(1071) Injected Performance Result: ''5m'=1%;80;90; '1m'=3%;80;90; '30s'=2%;80;90; '
d \NSCAThread.cpp(189) Executing (from NSCA):
d NSClient++.cpp(1034) Injecting: check_ok:
d NSClient++.cpp(1034) Injecting: CheckOK: Everything is fine
d NSClient++.cpp(1070) Injected Result: OK 'Every thing is fine'
d NSClient++.cpp(1071) Injected Performance Result: ''
d NSClient++.cpp(1070) Injected Result: OK 'Every thing is fine'
d NSClient++.cpp(1071) Injected Performance Result: ''
d \NSCAThread.cpp(245) Sending to server...
```

And everything looks like it went super... **BUT** and this is a bit but. the NSCA protocol does not support any result checking. We submit the result and we are done there is no "returned information" so everything could have gone terribly wrong and we would not see anything at all.

And here is where we need to start debugging on the Nagios (or NSCA) side.

```
sudo tail -f /var/log/syslog
```

will result in the following:

```
Jul 12 19:35:20 localhost nsca[27093]: Connection from 192.168.0.104 port 26117
Jul 12 19:35:20 localhost nsca[27093]: Handling the connection...
Jul 12 19:35:21 localhost nsca[27093]: Received invalid packet type/version from client
   - possibly due to client using wrong password or crypto algorithm?
```

And this is clue that we have indeed miss configured NSCA. Most often it is either invalid password or the wrong encryption so if we make sure these are correct we will end up with the following instead:

```
Jul 12 19:42:54 localhost nsca[27157]: Connection from 192.168.0.104 port 60421
Jul 12 19:42:54 localhost nsca[27157]: Handling the connection...
Jul 12 19:42:55 localhost nsca[27157]: Dropping packet with stale timestamp - packet was 57 seconds old.
```

This is another issue you might sometime need to resolve it means the clocks of the machines are not in perfect syncronization. This can be solved in three ways:

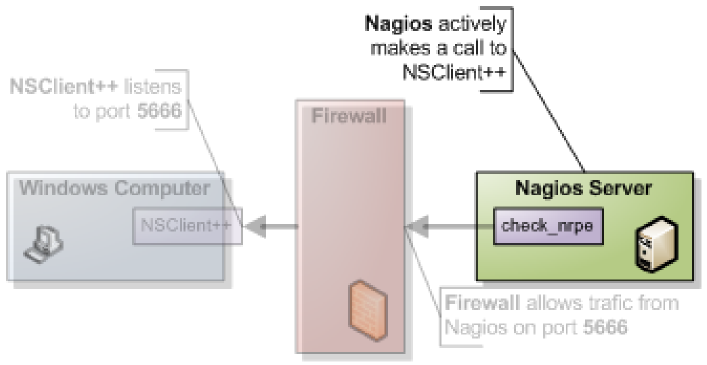
1. Sync the clocks
2. Use the time_delay to change the "local time" in NSClient++
3. Change the max_packet_age in NSCA.cfg

When we have fixed this we end up with the following:

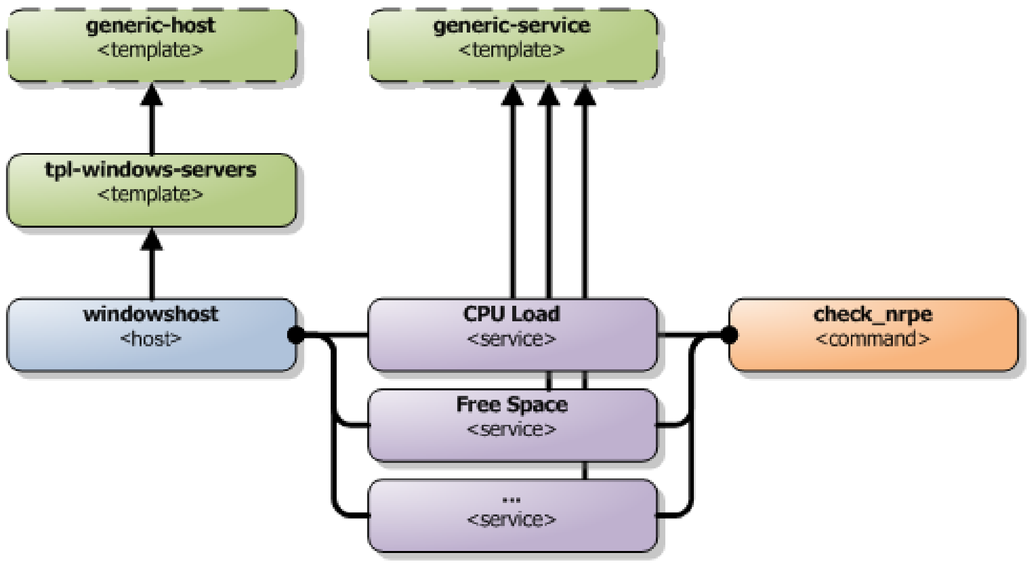
```
Jul 12 19:47:01 localhost nsca[27207]: Connection from 192.168.0.104 port 8198
Jul 12 19:47:01 localhost nsca[27207]: Handling the connection...
Jul 12 19:47:02 localhost nsca[27207]: SERVICE CHECK -> Host Name: 'DESKTOP',
  Service Description: 'CPU Load', Return Code: '0',
  Output: 'OK CPU Load ok.|'5m'=0%;80;90; '1m'=1%;80;90; '30s'=3%;80;90; '
Jul 12 19:47:02 localhost nsca[27207]: HOST CHECK -> Host Name: 'DESKTOP',
  Return Code: '0', Output: 'Everything is fine|'
Jul 12 19:47:02 localhost nsca[27207]: End of connection...
```

And this means (hopefully) that communication is all working and all you need to do now is configure the checks in Nagios.

5. Configure Nagios


Nagios actively makes a call to NSClient++
NSClient++ listens to port 5666
Firewall
Windows Computer
NSClient++
Nagios Server
check_nrpe
Firewall allows trafic from Nagios on port 5666

5.1 Introduction



Nagios configuration is in itself a whole chapter and this is just a quick peek on how you can do things. First off there are a few concepts to understand:

- templates are the same as the corresponding item but they have a flag register = 0 which makes them "unlistable items"
- services are essentially checks (is check CPU)
- hosts are essentially computers
- groups are an important concept which I ignore here for simplicity (I recommend you use it)

The configuration is at the end layer quite simple you have a "check" and a "host" and you connect them with a service. Like I show at the bottom line in the diagram above. Whats makes this a tad more complicated is that you can inherit things from a "parent" definition. Which is what I show with arrows (bottom to top) above. The templates with dashed lines are the base templates which all services and hosts inherit.

5.2 Passive Checks

The main difference between passive checks and active checks are the following two flags:

active_checks_enabled
Active service checks are enabled

passive_checks_enabled
Passive service checks are enabled/accepted

So adding the following will "change" an active check to a passive check.

```
        active_checks_enabled   0 ; Active service checks are enabled
        passive_checks_enabled  1 ; Passive service checks are enabled/accepted
```

So you say what shall I enter for command for my passive checks?

There are several options for this depending on what you want I wont (as always) go into the details in this quick guide but the short of it is either you use check_dummy or you use the actual command and setup freshness checks. With freshness checks active it means that if a result is not submitted Nagios will actively go out and seek the information (this is what I would recommend for host checks at least).

5.3 Template

First, its best practice to create a new template for each different type of host you'll be monitoring. Let's create a new template for windows servers.

```
define host{
        name                    tpl-windows-servers ; Name of this template
        use                     generic-host ; Inherit default values
        check_period            24x7
        check_interval          5
        retry_interval          1
        max_check_attempts      10
        check_command           check-host-alive
        notification_period     24x7
        notification_interval   30
        notification_options    d,r
        contact_groups          admins
        register                0 ; DONT REGISTER THIS - ITS A TEMPLATE
}
```

Notice that the tpl-windows-servers template definition is inheriting default values from the generic-host template, which is defined in the sample localhost.cfg file that gets installed when you follow the Nagios quickstart installation guide.

5.4 Host definition

Next we need to define a new host for the remote windows server that references the newly created tpl-windows-servers host template.

```
define host{
        use             tpl-windows-servers ; Inherit default values from a template
        host_name       windowshost ; The name we're giving to this server
        alias           My First Windows Server ; A longer name for the server
        address         10.0.0.2 ; IP address of the server
        active_checks_enabled   0 ; Active host checks are enabled
        passive_checks_enabled  1 ; Passive host checks are enabled/accepted
}
```

Defining a service for monitoring the remote Windows server. These example service definitions will use the sample commands that are defined in the default NSC.ini file which ships with NSClient 0.3.7 or newer.

5.5 Service definitions

The following service will monitor the CPU load on the remote host. The "alias_cpu" argument which is passed to the check_nrpe command definition tells NSClient++ to run the "alias_cpu" command as defined in the alias section of the NSC.ini file.

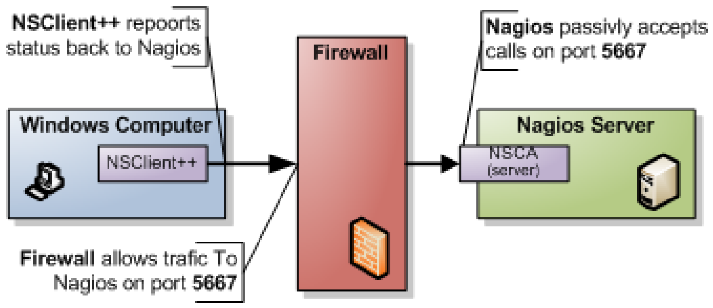
```
define service{
        use                     generic-service
        host_name               windowshost
        service_description     CPU Load
        check_command           check_nrpe!alias_cpu
        active_checks_enabled   0 ; Active service checks are enabled
        passive_checks_enabled  1 ; Passive service checks are enabled/accepted
}
```

The following service will monitor the free drive space on /dev/hda1 on the remote host.

```
define service{
        use                     generic-service
        host_name               windowshost
        service_description     Free Space
        check_command           check_nrpe!alias_disk
        active_checks_enabled   0 ; Active service checks are enabled
        passive_checks_enabled  1 ; Passive service checks are enabled/accepted
}
```

Now a better way here is to add a new template and derive the service checks for a "tpl-passive-service" instead and put the passive options there but alas I was to lazy to do so in this quick guide.

6. Where to go next


NSClient++ repoorts status back to Nagios
Windows Computer
NSClient++
Firewall
Nagios passivly accepts calls on port 5667
Nagios Server
NSCA (server)
Firewall allows trafic To Nagios on port 5667

This is of cores not the end now you need to check out what checks you want to use run on your servers. There is a lot of built-in checks but there are a lot more external scripts you can use and download from for instance monitoring exchange or the new nagios exchange.

Built in checks:

- CheckAlwaysCRITICAL (check)
- CheckAlwaysOK (check)
- CheckAlwaysWARNING (check)
- CheckCPU (check)
- CheckCRITICAL (check)
- CheckCounter (check)
- CheckEventLog/CheckEventLog (check)
- CheckFile (check)
- CheckFileSize (check)
- CheckMem (check)
- CheckMultiple (check)
- CheckOK (check)
- CheckProcState (check)
- CheckServiceState (check)
- CheckTaskSched/CheckTaskSched (check)
- CheckUpTime (check)
- CheckVersion (check)
- CheckWARNING (check)
- CheckWMI/CheckWMI (check)
- CheckWMIValue (check)