

If you're monitoring Windows via *Nagios/NagiosXI* then chances are you are using *NSCP*(NSClient++) to do it. While I use NSClient++ (NSCP) as an example, the information presented should cross over without much major difference.

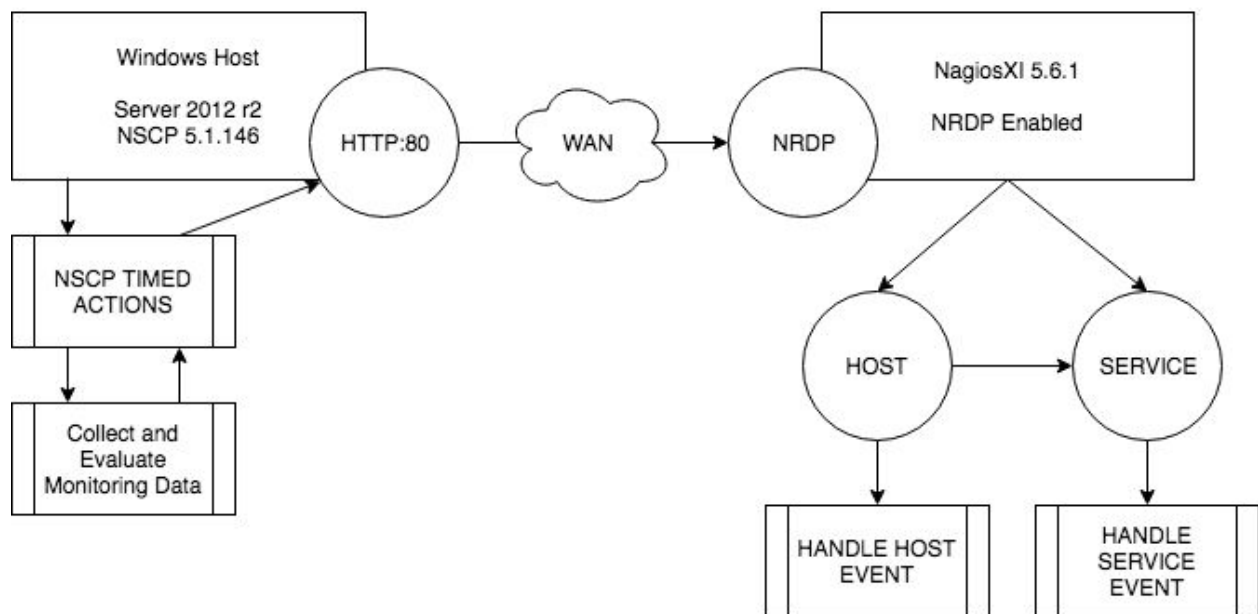
Quite true to form for a multi-tool, NSCP is very capable of pulling and submitting data in a plethora of ways. For me, the single drawback in using NSCP is in the arena of self-health and performance monitoring. In a Passive implementation with break/fix situation alerting for a production environment, how do you try and mitigate this?

While the "*check_nscp*" module gives you the basics of the agent; installed version, application crash count, logged error count and last error message and using the "*check_pdh*" module will allow for you to query any available *Windows Performance Data Counter* and return that value. We are still dependent on NSCP fully functional and neither method provides for the evaluation the nitty-gritty performance data.

Here again is the limitation with evaluation of the collected Performance Data via NSCP. That and you're still dependent upon NSCP being fully functional to ship any data back to Nagios/NagiosXI. How exactly do you monitor the monitoring agent?

Take a look at the example passive monitoring implementation.

NSCP Passive Windows Monitoring



In this example I've described a typical passive monitoring solution based on NSCP. Since I'm depending only on NSCP to communicate with Nagios the BIG thing that I need to know is the "condition" of NSCP on the Host! If NSCP is stopped or otherwise broken, there's no data going to Nagios and I'm not monitoring. That's a huge problem!

So what's the solution? Why not just write a script?

Now that it's decided, what does our script need to do?

1. The heavy lifting. Query, collect, parse and evaluate the data for both the service and performance counters.
2. Report the results of our evaluations into a properly formatted JSON hash and send to HTTP-POST to NagiosXI via NRDP.

I'm only working on Windows which makes PowerShell the obvious choice. I've got decent chops with PowerShell and honestly, I couldn't think of a reason that I couldn't accomplish both items with a little scripting now that the tasks are worked out. Lets get on to the code.

You have to start somewhere and everything in our passive setups hinges on the state of the NSCP service. This seems a logical place for us to get started.

Getting NSCP Service (Name, ExitCode, ProcessID, StartMode, State and Status)

```
$nscpinfo = @()
$nscpinfo += @(gwmi win32_service -filter "name like nscp")
echo $nscpinfo
```

That was easy. Now what to do with it?

Evaluate the NSCP Service State. Translate the Service State to Nagios Host State.

Isolating and Evaluating the Service State for NSCP and Service State to Host State translation is handled via an "if->else-if->else" code block.

Formatting the output to be included within the host_check to match that already configured for NSCP is done after our Service State evaluation is complete.

```
$nscpState = $nscpinfo.State

if($nscpState -ne "Running")
{
    # If NSCP is not running, I'm not monitoring
    # Any state other than running is a problem!

    # Mimic the formatting and content for the Host_check in NSCP
    $msg = "NSCP=(Not Running)"

    # Our "Displayed State"
    $dState = "CRITICAL"
```

```

# Our State to Return to Nagios
$sState = 1
}

elseif($nscpState -eq "Running")
{

# OK, NSCP Running! I'm monitoring things.
# Mimic the formatting and content for the Host_check in NSCP
$msg = "NSCP=(Running)"

# Our "Displayed State"
$dState = "OK"

# Our State to Return to Nagios
$sState = 0
}

else
{
# NSCP is...
# If the system fails to gather the information for NSCP
# this should catch the failure and let us send an UNKNOWN
# value with the error in the message.

# Mimic the formatting and content for the Host_check in NSCP
$msg = "NSCP=(No Data)"

# Our "Displayed State"
$dState = "UNKNOWN"

# Our State to Return to Nagios
$sState = 2
}

# Take the evaluation data and format the output for our host_check

# ServiceOutput[DISPLAYEDSTATE: Message] |
PerformanceData[nscp=(0=UP,1=DOWN,2=UNREACHABLE);]
$output = "$($dState): $($msg) | nscp=$($sState);"

echo $output

```

Well, that too was easy... What's next?

Send the Evaluation/Alert to Nagios via NRDP

Since the operational state of NSCP is the lynch pin of my monitoring architecture and I've already matched the output messages of the two sources, I'm comfortable with integrating external service state evaluation into the "host_check" data already being sent to NagiosXI via NSCP. The existing "host_check" in NSCP is set to "check_always_ok" and will only send data when running. We will sync the output of our script and NSCP to help avoid confusion,

We are using a passive monitoring design meaning that NRDP has already been configured inside my Nagios/NagiosXI Server and a valid NRDP URL and Token are already configured and working.

(With the some options enabled in the Nagios.conf file, we are able to suspend alerting/notifications for other service checks dependent upon NSCP. When the NSCP service is down we will skip processing other checks helping to greatly reduce any noise being injected into our incident management numbers)

Last item, converting the evaluation/alert into a JSON string and then forwarding it to NRDP with the *Invoke-WebRequest cmd-let*.

```
# NSCP_PA.PS1
# Author: snapier
# Date: 03-MAY-2019
# Monitor the service state for NSCP and notify nagios via
# NRDP (Hostcheck) for service state change processing.

$ncscinfo = @()
$ncscinfo += @(gwmi win32_service -filter "name like nscp")
$ncscState = $ncscinfo.State

if($ncscState -ne "Running")
{
    # If NSCP is not running, I'm not monitoring
    # Any state other than running is a problem!

    # Mimic the formatting and content for the Host_check in NSCP
    $msg = "NSCP=(Not Running)"

    # Our "Displayed State"
    $dState = "CRITICAL"

    # Our State to Return to Nagios
    $sState = 1
}

elseif($ncscState -eq "Running")
{
    # OK, NSCP Running! I'm monitoring things.
    # Mimic the formatting and content for the Host_check in NSCP
    $msg = "NSCP=(Running)"
}
```

```

# Our "Displayed State"
$dState = "OK"

# Our State to Return to Nagios
$sState = 0
}

else
{
# NSCP is...
# If the system fails to gather the information for NSCP
# this should catch the failure and let us send an UNKNOWN
# value with the error in the message.

# Mimic the formatting and content for the Host_check in NSCP
$msg = "NSCP=(No Data)"

# Our "Displayed State"
$dState = "UNKNOWN"

# Our State to Return to Nagios
$sState = 2
}

# -----
# NRDP SETTINGS
# -----

# Hostname in lowercase
$myHost = $env:COMPUTERNAME.ToLower()

# NRDP Token (Change this to your configured token)
$token = "token"

# NRDP URL (Change this to your IP)
$nrdpurl = "http://192.168.1.2/nrdp/"

# -----
# HOSTCHECK AND JSONDATA
# -----

$jsondata_open = 'JSONDATA={"checkresults":["'

```

```

$jsondata_close = '}]'
$hostcheck = "check_result":{"type":"host","check_type":"1"},"hostname":"+$($myhost)+",
"state":"+$($sState)+", "output":"+$($dState)+' : '+$($msg)+'" | nscp="+$($sState)+"}'

#json post data
$json = "$($jsondata_open) $($hostcheck) $($jsondata_close)"

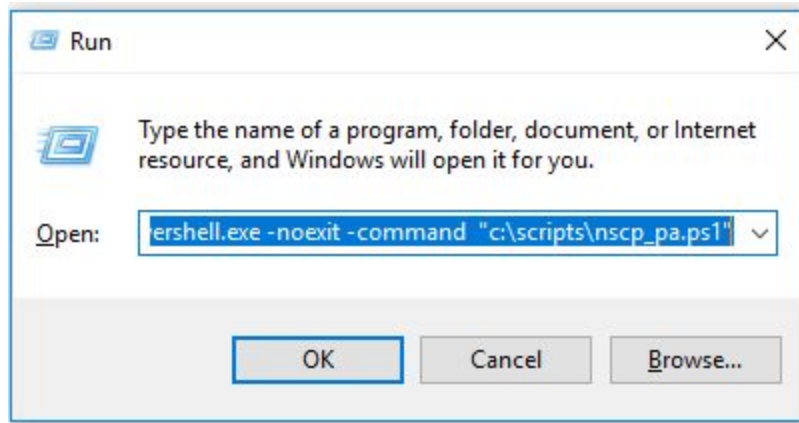
# Formatting Sanity Check
echo "JSONDATA = $($json) `n"

#HTTP POST
$post = @()
$post += @(Invoke-WebRequest -UseBasicParsing
"$($nrdpurl)?$($token)&cmd=submitcheck&$($json)" -ContentType "application/json" -Method POST)

#POST Status
$result = $post.Content
echo "NRDP = $($result) `n"

```

At this point, we should have a fully functioning power shell script. Lets see if it works.
(I want to test the script by executing it as close to the same way the scheduled event trigger will dot it so
I use the windows-run method with the -noexit option)



HAZA!

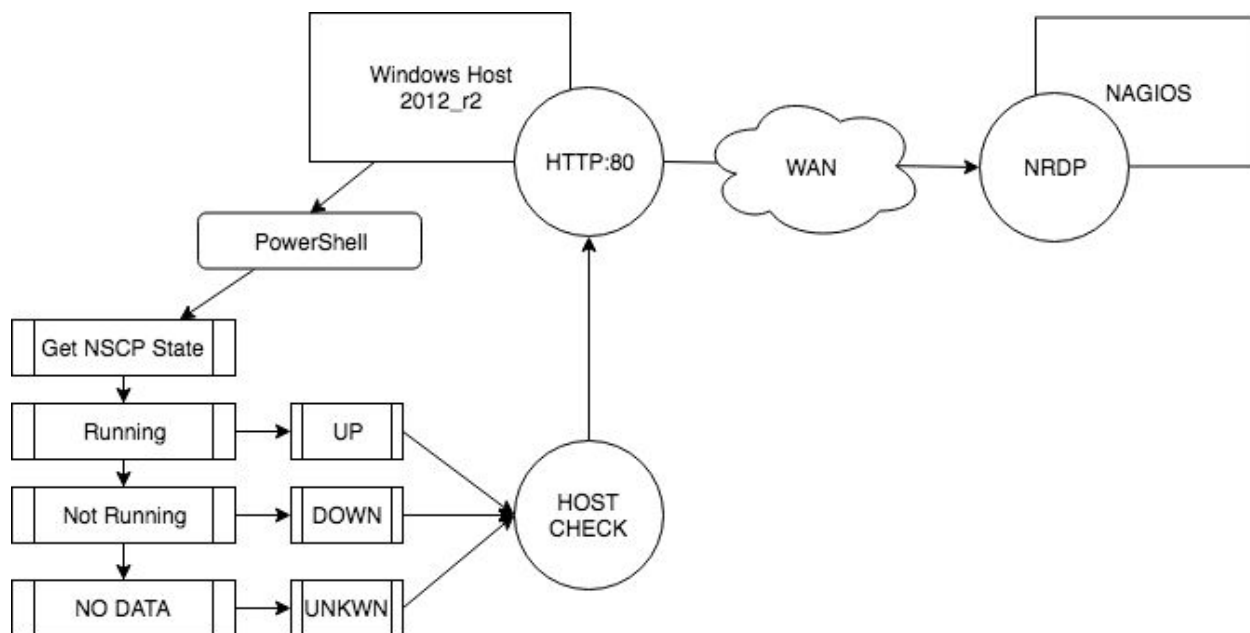
We do indeed have a functioning script. In this one line we get the conformation our NSCP Service State was sent to and processed by NRDP in the form of a Host Check. NRDP received the cmd call to SubmitCheck, processed the host_check data in the JSONDATA string and passed it into the event queue for Nagios.

```

Windows PowerShell
JSONDATA = JSONDATA={"checkresults":[{"checkresult":{"type":"host","checktype":"1","hostname":"somehost","state":
"0","output":"OK: NSCP=(Running) | nscp=0;"}]}
NRDP = { "result" : { "status" : "0", "message" : "OK", "output" : "1 checks processed." } }
PS C:\Windows\system32\WindowsPowerShell\v1.0>

```

Let visualize that



Conundrum... Now we have two host checks. How do we stop one from stepping on the other? Or, do we want to?

Since I'm already sending a host_check with NSCP (when it's running) to Nagios there's half of the data there for our Up/Down story already. With the addition of a windows scheduled task to run our PowerShell script 5 minutes our NSCP Up/Down story is finished.

When NSCP is running, we get the UP state, event message "OK: NSCP=(Running)" and the performance data counter "nscp=100;" from both our input sources.

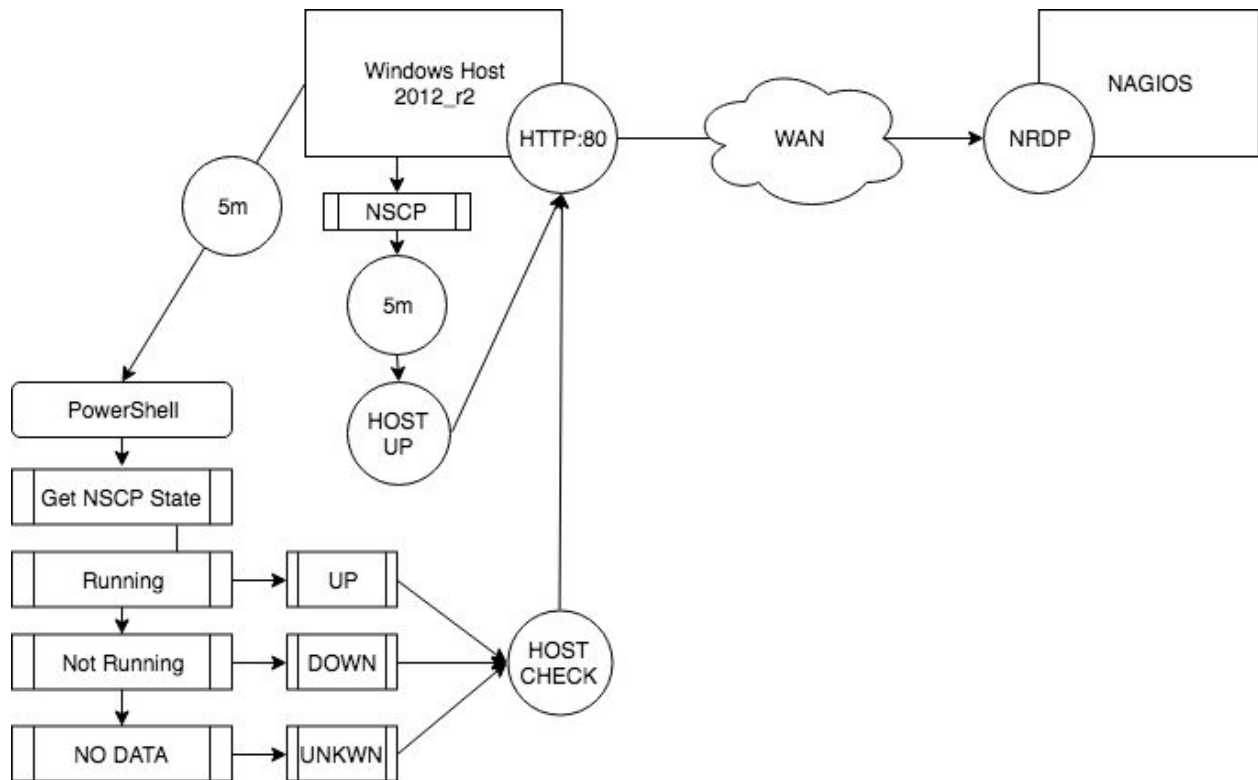
Only when NSCP is not running will we get the DOWN state, event message "CRITICAL: NSCP=(Not Running)" and performance data counter "nscp=-1;" from a single source.

If no data is available for the NSCP Service we will get the UNREACHABLE state, event message "UNKNOWN: NSCP=(Unknown)" and performance data counter "nscp=0" from a single source.

When NSCP recovers and once again enters the Running state the UP state, event message "OK: NSCP=(Running)" and the performance data counter "nscp=100;" from both our input sources will resume.

The performance data counter "nscp=state(X,98,95)" allows us to graph the NSCP service availability over time within Nagios.

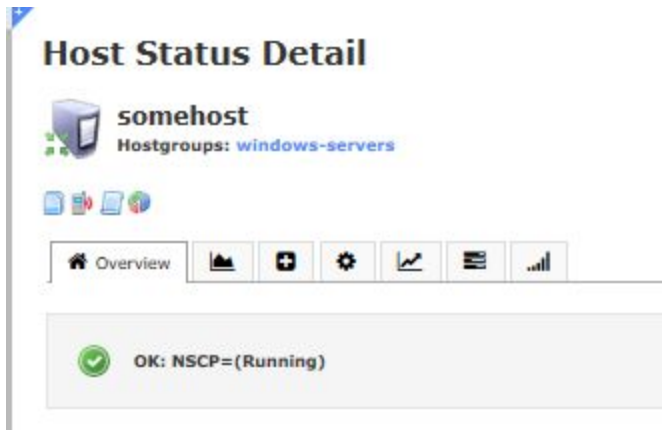
Our passive monitoring solution can now be visualized showing both "host_check" inputs.



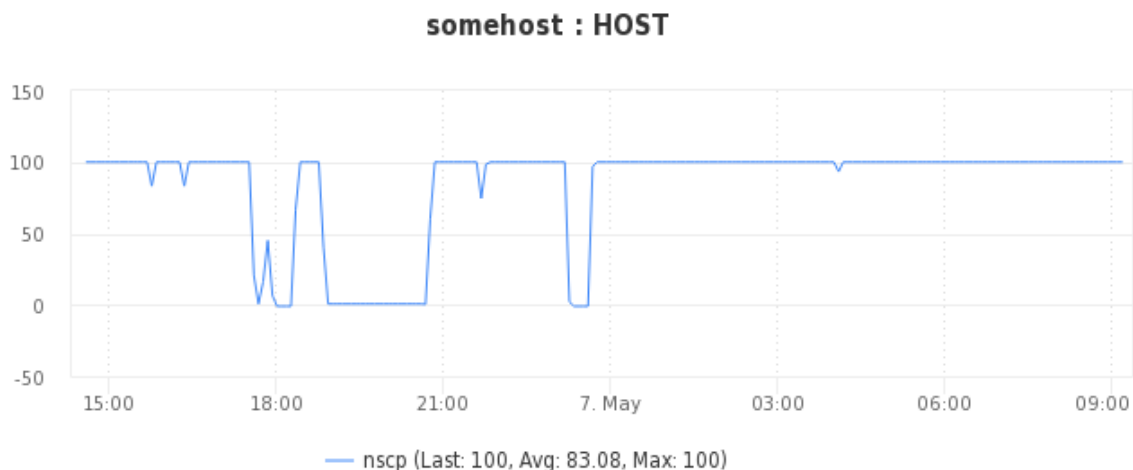
- We can verify the method with the following scenario.
 1. With the scheduled event to run the PowerShell script enabled, stop the NSCP service on the Windows Host. When the script or event is executed, the Host State will change to Down in Nagios.



2. With the scheduled event to run the PowerShell script enabled, start the NSCP service on the Windows Host. When the script or event is executed, the Host State will change to UP in Nagios.



3. We will track the percentile range for NSCP service availability via the performance counter included with the host check.



If the state of NSCP was our only goal then we could stop here but, does the script really tell us anything about the health and performance at this point? If NSCP is in a running state, what else could I use to tell me that it is indeed working as expected or not?

Here's the short list;

Log File Activity - In my NSCP configuration I have the "Log" destination set as an output for all of my timer managed service check. This means that NSCP writes to the log for services and the application. If the NSCP is not in a "Stopped" state then the log file will/should be updated every two (2) minutes when the scheduler runs a check.

Service Startup Type - I expect NSCP to be running when the target OS boots. To accomplish this the "Startup Type" for NSCP needs to be either Automatic or Automatic(Delayed Start) any other setting could disable monitoring if there is a target restart or unexpected reboot.

Both of these evaluations are easy enough to accomplish in PowerShell and the NRDP portion of the script is already sending the "host_check". In reading about NRDP I know that I can send multiple "check_result" types with just a little modification, back to the script.

NSCP Log Activity/Last Log Write Time Check

- If NSCP is "Not Stopped" then there should be an entry for our services when executed on the set Timer interval of (2) minutes.
- If NSCP enters the "Starting/Running/Stopping/Restarting" states, there is a log entry written.
- If the Delta of Time Last Written and Now is greater than 1.5times the interval and less than 2times the interval send a warning.
- If the Delta of Time Last Written and Now is greater than 2times the interval then send a critical

From reading through the NRDP documentation I know that the only differences in the "host_check" I have working and this one are; setting "check_type" to "service" and including a "service_name".

I do the time calculations and the build the check result.

```
#Get NSClient.Log file data
```

```
$nscplog = @()
```

```
$nscplog = @(Get-Item "C:\Program Files\NSClient++\nscclient.log")
```

```
# Get the last time the file was written too
```

```
$ludt = Get-Date($nscplog.LastWriteTime)
```

```
# Get the duration of time since the log file has been modified to now.
```

```
$ludt_delta = (New-TimeSpan -Start ($ludt) -End (Get-Date))
```

```
#convert the time delta to minutes so that we can evaluate the result
```

```
$ludt_delta_min = $ludt_delta.Minutes
```

```
#I have a very robust schedule for my NSClient timer, 2-minutes. If my log file has not been written to within 1-2 times
```

```
#the duration of the timer then I want a warning. If the delta is double or greater my timer then I want a critical.
```

```

if(($ludt_delta_min -ge 3) -and ($ludt_delta_min -lt 5)){
    $luState = "1"
    $ludStatus = "WARNING"
    $lumsg = "$($ludStatus): NSClient.log file has not updated in $($ludt_delta_min) minute/s.
LASTUPDATE=$(($ludt)) | log-update-delta=$(($ludt_delta_min);3;5;"
}elseif($ludt_delta_min -ge 5){
    $luState = "2"
    $ludStatus = "CRITICAL"
    $lumsg = "$($ludStatus): NSClient.log file has not updated in $($ludt_delta_min) minute/s.
LASTUPDATE=$(($ludt)) | log-update-delta=$(($ludt_delta_min);3;5;"
}else{
    $luState = "0"
    $ludStatus = "OK"
    $lumsg = "$($ludStatus): NSClient.log updated($(($ludt_delta_min)) minute/s ago.
LASTUPDATE=$(($ludt)) | log-update-delta=$(($ludt_delta_min);3;5;"
}

```

```

# Build an array with the results for check_lastupdate.
$loglastupdate = @($ludt,$ludt_delta_min,$ludStatus,$luState,$lumsg,$lu_check_valid)

```

```

#the final variable is a the boolean for IS-VALID. If the NSCP service is not equal to stopped, the check
#will be run as normal. If the value is NO, then the Unknown value will be returned.
if($loglastupdate[5] -eq "YES"){
    # If the state is not stopped, then we want to run our comparative check.
    $lastupdate_check = '{"checkresult": {"type": "service", "checktype": "1"}, "hostname": "' + $($myhost) + "',
    "servicename": "win--system--nscp--last-nsclient-log-updated-time", "state": "' + $($loglastupdate[3]) + "',
    "output": "' + $($loglastupdate[4]) + "'"
}else{
    # If the agent is stopped then there is no reason to run the comparative check. To avoid noise in our
    incident
    $lastupdate_check = '{"checkresult": {"type": "service", "checktype": "1"}, "hostname": "' + $($myhost) + "',
    "servicename": "win--system--nscp--last-nsclient-log-updated-time", "state": "3", "output": "UNKNOWN: No
    Log Activity Expected NSCP=(Stopped). | last-update-delta=-1;3;5"}'
}

```

Now that I have the check evaluation working one last code tweak to enable the service check.

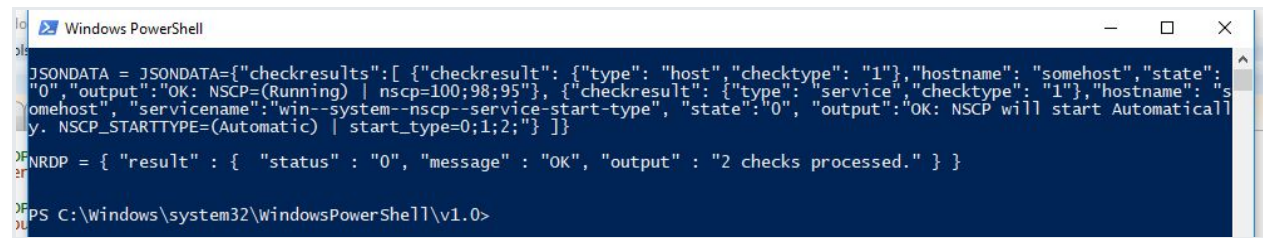
```

$jsondata_open = 'JSONDATA={"checkresults":['
$jsondata_close = ']'

```

#json post data. each service type check should be seperated by a ","

```
$json = "$($jsondata_open) $($hostcheck), $($lastupdate_check) $($jsondata_close)"
```



```
JSONDATA = JSONDATA={"checkresults":[{"checkresult":{"type": "host", "checktype": "1", "hostname": "somehost", "state": "0", "output": "OK: NSCP=(Running) | nscp=100;98;95"}, {"checkresult": {"type": "service", "checktype": "1", "hostname": "somehost", "servicename": "win--system--nscp--service-start-type", "state": "0", "output": "OK: NSCP will start Automatically. NSCP_STARTTYPE=(Automatic) | start_type=0;1;2;"} ]}
NRDP = { "result" : { "status" : "0", "message" : "OK", "output" : "2 checks processed." } }
PS C:\windows\system32\WindowsPowerShell\v1.0>
```

We can verify our new check is showing up by viewing the unconfigured objects list in NagiosXI. Once verified, fast forward through adding the new check and now we are getting somewhere! Adding the NSCP StartType check is next in line on the list.

NSCP Service Startup Type

NSCP STARTUP TYPE EVALUATION

```
if($nscpStartType -eq "Disabled"){
    $nscpStartTypeState = 2
    $nscpStartTypeDisplayState = "CRITICAL"
    $nscpStartTypeMsg = "CRITICAL: NSCP IS DISBALED!"
    NSCP_STARTTYPE=($($nscpService.StartType)) | start_type=2;1;2;"
}elseif($nscpStartType -eq "Manual"){
    $nscpStartTypeState = 1
    $nscpStartTypeDisplayState = "WARNING"
    $nscpStartTypeMsg = "WARNING: NSCP NOT SET TO AUTO START!"
    NSCP_STARTTYPE=($($nscpService.StartType)) | start_type=1;1;2;"
}elseif($nscpStartType -eq "Automatic"){
    $nscpStartTypeState = 0
    $nscpStartTypeDisplayState = "OK"
    $nscpStartTypeMsg = "OK: NSCP will start Automatically."
    NSCP_STARTTYPE=($($nscpService.StartType)) | start_type=0;1;2;"
}else{
    $nscpStartTypeState = 3
    $nscpStartTypeDisplayState = "UNKNOWN"
    $nscpStartTypeMsg = "UNKNOWN: NO DATA FOR NSCP_STARTTYPE=($($nscpService.StartType))
| start_type=3;1;2;"
}
```

#NSCP_StartupType_Check result for NRDP

```
$nscpStartType_check = '{"checkresult": {"type": "service", "checktype": "1", "hostname":
"+$($myhost)+"", "servicename": "win--system--nscp--service-start-type",
"state": "+$($nscpStartTypeState)+"", "output": "+$($nscpStartTypeMsg)+""}'
```

After a little copy-paste action, few tweaks in the logic to better help with when to run the checks, we run the test... BAM!

```
Windows PowerShell

JSONDATA = JSONDATA={"checkresults":[{"checkresult":{"type": "host", "checktype": "1", "hostname": "somehost", "state": "0", "output": "OK: NSCP=(Running) | nscp=100;98;95"}, {"checkresult":{"type": "service", "checktype": "1", "hostname": "somehost", "servicename": "win--system--nscp--service-start-type", "state": "0", "output": "OK: NSCP will start Automatically. NSCP_STARTTYPE=(Automatic) | start_type=0;1;2"}, {"checkresult":{"type": "service", "checktype": "1", "hostname": "somehost", "servicename": "win--system--nscp--last-nscclient-log-updated-time", "state": "0", "output": "OK: NSCclient.log updated(1) minute/s ago. LASTUPDATE=(05/08/2019 17:12:39) | log-update-delta=1;3;5;"}]}

NRDP = { "result" : { "status" : "0", "message" : "OK", "output" : "3 checks processed." } }


PS C:\Windows\system32\WindowsPowerShell\v1.0>
```


Fast forward through the adding our second object. It's Alive!








Now that we have our NSCP availability being monitored or think we do. Trust but verify! I need to go through and verify that my checks are catching the conditions expected. The "startup type" is the easiest so I start there.


- Verify the NSCP Service Start Type Check
 1. Modify the Startup Type for the NSClient++ agent to each of the available options.
 2. Run the script manually or wait for the timed interval to elapse.
 3. Verify the event and state being reported to Nagios.


Service Status Detail

 **win--system--nscp--service-start-type**
somehost





 Overview      

 **OK: NSCP will start Automatically. NSCP_STARTTYPE=(Automatic)**

Status Details	
Service State:	 Ok
Duration:	1d 12h 54m 2s
Service Stability:	Unchanging (stable)
Last Check:	2019-05-08 06:15:04
Next Check:	Not scheduled

Quick Actions

 [Disable notifications](#)

 [Force an immediate check](#)

Acknowledgements and Comments

No comments or acknowledgements.

Service Status Detail



win--system--nscp--service-start-type
somehost



Overview



WARNING: NSCP NOT SET TO AUTO START! NSCP_STARTTYPE=(Manual)

Status Details

Service State:	Warning
Duration:	18s
Service Stability:	Unchanging (stable)
Last Check:	2019-05-08 14:06:41
Next Check:	Not scheduled

Quick Actions

- Acknowledge this problem
- Disable notifications
- Force an immediate check

Acknowledgements and Comments

No comments or acknowledgements.

Service Status Detail



win--system--nscp--service-start-type
somehost



Overview



CRITICAL: NSCP IS DISBALED! NSCP_STARTTYPE=(Disabled)

Status Details

Service State:	Critical
Duration:	19s
Service Stability:	Unchanging (stable)
Last Check:	2019-05-08 14:07:24
Next Check:	Not scheduled

Quick Actions

- Acknowledge this problem
- Disable notifications
- Force an immediate check

Acknowledgements and Comments


No comments or acknowledgements.





Startup Type is working, now, how to verify the last log update? The only way to be sure is to create a scenario that will match/mimic the issue we are trying to catch and, to do that I have a plan.







Verify the Last Log Update Time for NSCP


1. On my target machine, I create a test file "nctest.ini" configuration file. In this file I only include the key to load a couple of modules at start but, none of the other keys that are supposed to be there. In this instance NSCP will start (write to the log) and stay running without updating the log file unless the service is cycled.
2. With the "nctest.ini" file in place all I have to do is restart the NSCP service and watch in Nagios for the events as the thresholds are crossed.

Service Status Detail


 **win--system--nscp--last-nsclient-log-updated-time**
somehost




Overview      


 **OK: NSClient.log updated(0) minute/s ago. LASTUPDATE=(05/08/2019 14:13:14)**

Status Details

Service State:	 Ok
Duration:	1d 10h 7m 33s
Service Stability:	Unchanging (stable)
Last Check:	2019-05-08 14:13:28
Next Check:	Not scheduled

Quick Actions

 [Disable notifications](#)

 [Force an immediate check](#)

Acknowledgements and Comments

No comments or acknowledgements.

Service Status Detail



win--system--nscp--last-nsclient-log-updated-time
somehost



Overview



WARNING: NSClient.log file has not updated in (3) minute/s. LASTUPDATE=(05/08/2019 14:20:11)

Status Details

Service State:	Warning
Duration:	18s
Service Stability:	Unchanging (stable)
Last Check:	2019-05-08 14:23:42
Next Check:	Not scheduled

Quick Actions

- Acknowledge this problem
- Disable notifications
- Force an immediate check

Misc

No notes or misc info

Acknowledgements and Comments

No comments or acknowledgements.

Service Status Detail



win--system--nscp--last-nsclient-log-updated-time
somehost



Overview



CRITICAL: NSClient.log file has not updated in (5) minute/s. LASTUPDATE=(05/08/2019 14:20:11)

Status Details

Service State:	Critical
Duration:	18s
Service Stability:	Unchanging (stable)
Last Check:	2019-05-08 14:25:12
Next Check:	Not scheduled

Quick Actions

- Acknowledge this problem
- Disable notifications
- Force an immediate check

Misc

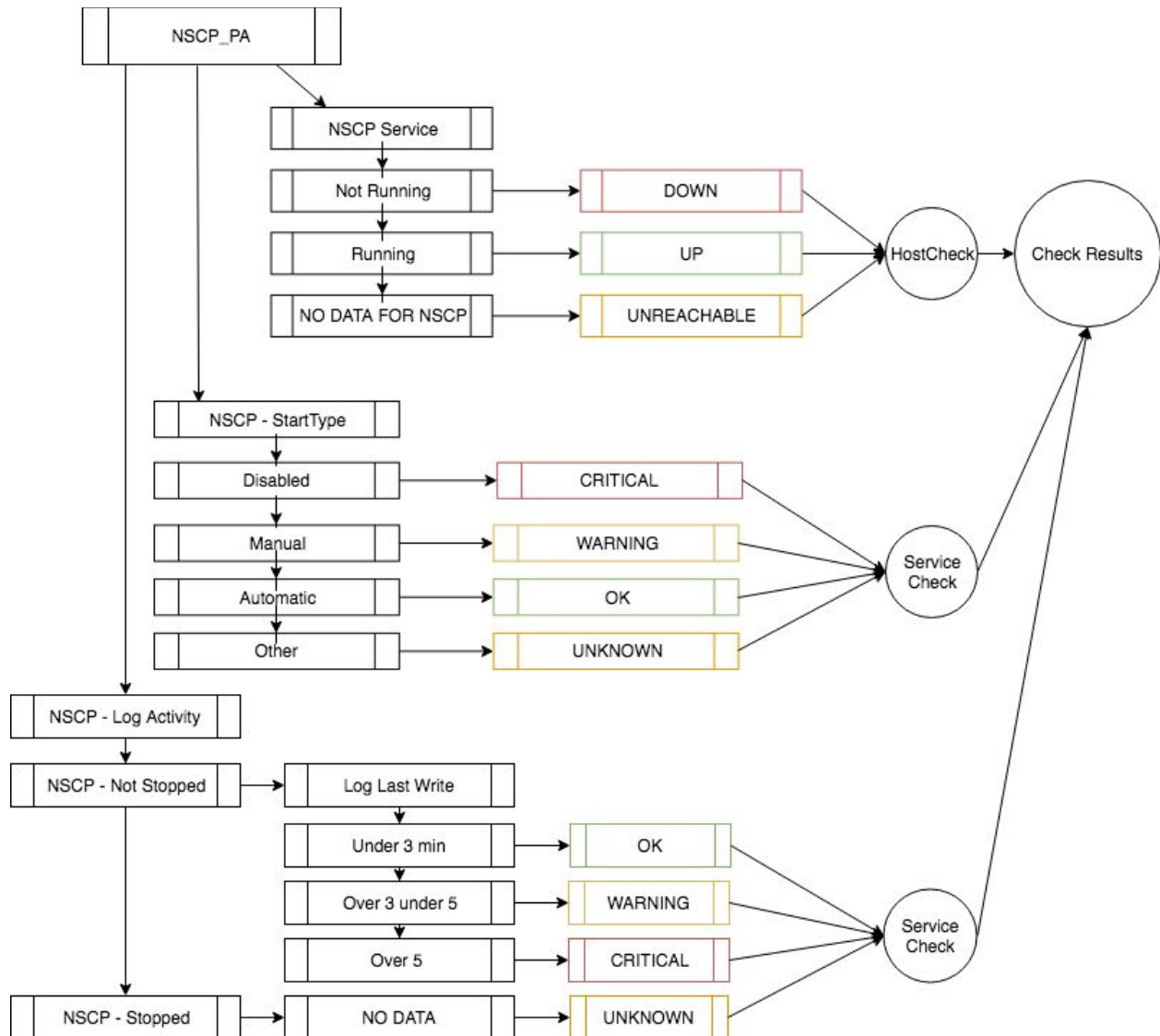
No notes or misc info

Acknowledgements and Comments

No comments or acknowledgements.

Much Excite!

With just a little bit of PowerShell, Nagios NRDP and some Google-Fu we're way ahead of the game now vs. where we started. Our new script is working great! It has Service Up/Down, StartupType and NSCP Log Activity service checks. The script Will run without any user interaction after initial configuration on an interval of 5-minutes and ALL forwarding conditions have been verified working!



There's only one thing left.

!THE PERFORMANCE DATA!

I've been adding some custom performance counter data to my checks along the way and I'm liking the results but, right now, none of them tell me anything about NSCP directly. There is a ton of performance data counters available in Windows. We've got a script that can send custom service checks. What are we waiting for?

Getting the Performance Counter Data, what's there to grab? How do I get it?

```
Caption :  
CreatingProcessID : 732  
Description :  
ElapsedTime : 1492  
Frequency_Object :  
Frequency_PerfTime :  
Frequency_Sys100NS :  
HandleCount : 547  
IDProcess : 15080  
IODataBytesPersec : 0  
IODataOperationsPersec : 0  
IOOtherBytesPersec : 5420  
IOOtherOperationsPersec : 3  
IOReadBytesPersec : 0  
IOReadOperationsPersec : 0  
IOWriteBytesPersec : 0  
IOWriteOperationsPersec : 0  
Name : nscp  
PageFaultsPersec : 0  
PageFileBytes : 14090240  
PageFileBytesPeak : 21004288  
PercentPrivilegedTime : 0  
PercentProcessorTime : 0  
PercentUserTime : 0  
PoolNonpagedBytes : 22928  
PoolPagedBytes : 233328  
PriorityBase : 8  
PrivateBytes : 14090240  
ThreadCount : 30  
Timestamp_Object :  
Timestamp_PerfTime :  
Timestamp_Sys100NS :  
VirtualBytes : 4486385664  
VirtualBytesPeak : 4507664384  
WorkingSet : 34193408  
WorkingSetPeak : 41299968  
WorkingSetPrivate : 10727424  
PSComputerName : DESKTOP-
```

Wowzers, that is a ton of numbers! Now, what to do with it? What information is relevant? Unable to make a succinct list, I decided to send it ALL.

After a little copy-paste action, few tweaks in the logic to better help with when to run the checks later we end up with something that looks like this.

```
#NSCP Process Performance Data
$perfdData = @()
$perfdData += @(Get-WmiObject -Class Win32_PerfFormattedData_PerfProc_Process -Filter
"Name='nscp'")

#We will only have perfdData if the service is running.
if(($perfdData_check_valid -eq "YES") -and ($nscpState -ne "Stopped")){
    $pDataState = 0
    $pDataStatus = "OK"
    # NSCP Is running We have Data
    $perfdDataOut = $("ThreadCount="+$perfdData.ThreadCount+";")
    $("PageFileBytes="+$perfdData.PageFileBytes+";")
    $("PageFileBytesPeak="+$perfdData.PageFileBytesPeak+";")
    $("PoolNonpagedBytes="+$perfdData.PoolNonpagedBytes+";")
    $("PoolPagedBytes="+$perfdData.PoolPagedBytes+";") $("PrivateBytes="+$perfdData.PrivateBytes+";")
    $("PriorityBase="+$perfdData.PriorityBase+";")
    $("IOReadOperationsPersec="+$perfdData.IOReadOperationsPersec+";")
    $("PercentProcessorTime="+$perfdData.PercentProcessorTime+";")
    $("PercentUserTime="+$perfdData.PercentUserTime+";")
    $("PercentPrivilegedTime="+$perfdData.PercentPrivilegedTime+";")
    $pDataMsg = "OK: NSCP=(Not Stopped) Performance Data=(TRUE) | "+$($perfdDataOut)
}else{
    # NO perfdData, so we need to zero the values
    $pDataState = 3
    $pDataStatus = "UNKNOWN"
    $perfdDataOut = $("ThreadCount=0;") $("PageFileBytes=0;") $("PageFileBytesPeak=0;")
    $("PoolNonpagedBytes=0;") $("PoolPagedBytes=0;") $("PrivateBytes=0;") $("PriorityBase=0;")
    $("IOReadOperationsPersec=0;") $("PercentProcessorTime=0;") $("PercentUserTime=0;")
    $("PercentPrivilegedTime=0;")
    #This is a non evaluated check so we will always send the OK state. The zero data points are included.
    $pDataMsg = "UNKNOWN: NSCP=(Stopped) Performance Data=(FALSE) | "+$($perfdDataOut)
}

#PERFDATA Check
```

```
$perfd_data_check = '{"checkresult": {"type": "service", "checktype": "1"}, "hostname": "' + $($myhost) + '",
"servicename": "win--system--nscp--all-process-perfd_data", "state": "' + $($pDataState) + '",
"output": "' + $($pDataMsg) + '"}'
```

We add the check to the JSON string list, give it a go and, BAM!

```
Windows PowerShell
JSONDATA = JSONDATA={"checkresults": [{"checkresult": {"type": "host", "checktype": "1"}, "hostname": "somehost", "state":
"0", "output": "OK: NSCP=(Running) | nscp=100;98;95"}, {"checkresult": {"type": "service", "checktype": "1"}, "hostname": "s
omehost", "servicename": "win--system--nscp--service-start-type", "state": "0", "output": "OK: NSCP will start Automaticall
y. NSCP_STARTTYPE=(Automatic) | start_type=0;1;2"}, {"checkresult": {"type": "service", "checktype": "1"}, "hostname": "s
omehost", "servicename": "win--system--nscp--last-nsclient-log-updated-time", "state": "0", "output": "OK: NSClient.log upd
ated(0) minute/s ago. LASTUPDATE=(05/09/2019 00:23:04) | log-update-delta=0;3;5"}, {"checkresult": {"type": "service", "
checktype": "1"}, "hostname": "somehost", "servicename": "win--system--nscp--all-process-perfd_data", "state": "0", "output":
"OK: NSCP=(Not Stopped) Performance Data=(TRUE) | ThreadCount=30; PageFileBytes=12234752; PageFileBytesPeak=20860928; Po
olNonpagedBytes=22112; PoolPagedBytes=228920; PrivateBytes=12234752; PriorityBase=8; IOReadOperationsPersec=0; PercentPr
ocessorTime=0; PercentUserTime=0; PercentPrivilegedTime=0;"} ]}

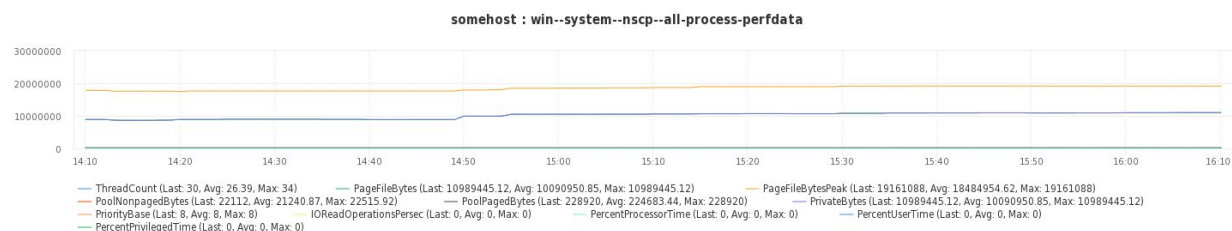
NRDP = { "result" : { "status" : "0", "message" : "OK", "output" : "4 checks processed." } }

PS C:\windows\system32\WindowsPowerShell\v1.0>
```

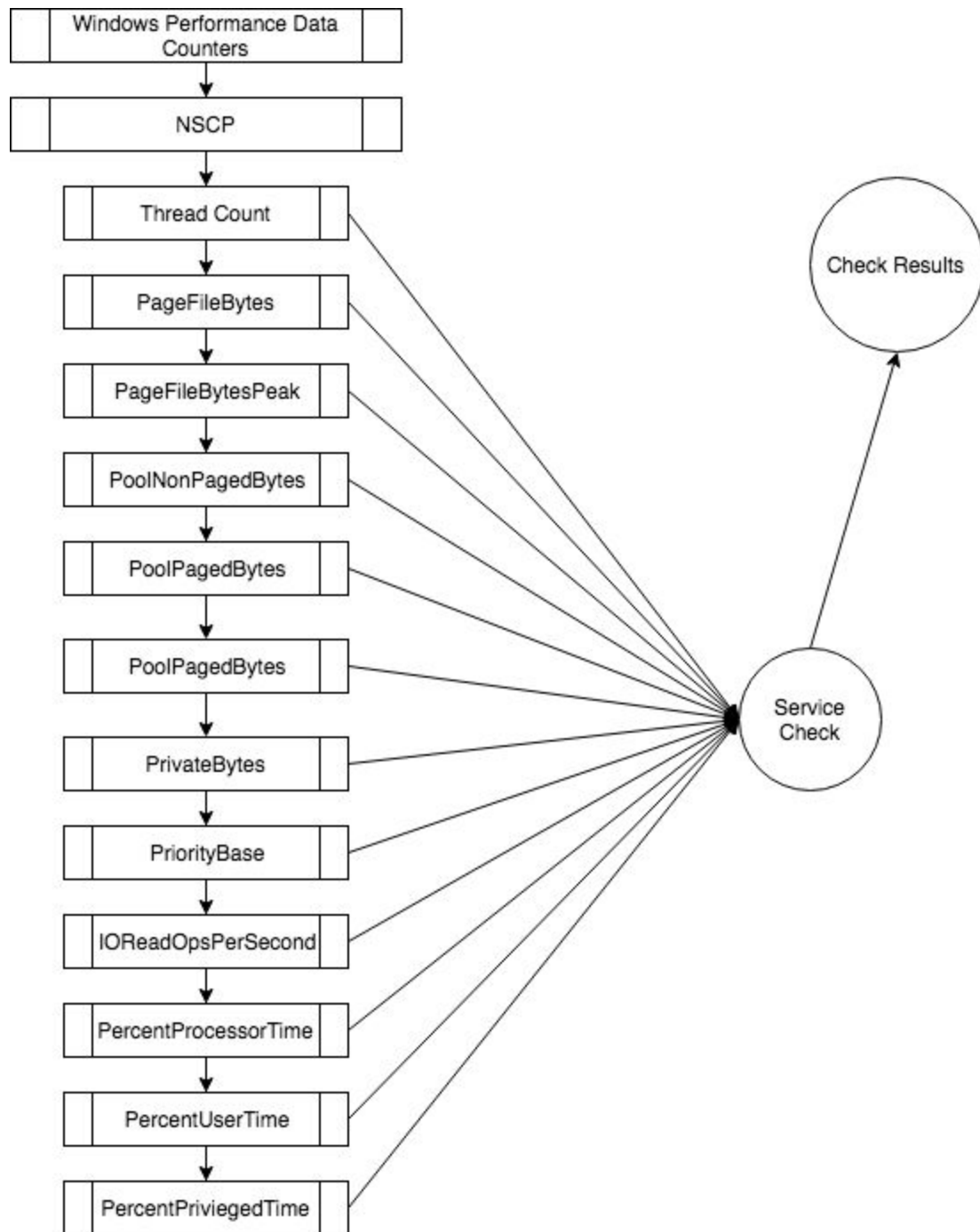
Fast forward through the adding of the new service to NagiosXI and we are on to validation.

Validating the Performance Data Collection Check

1. With the "perfd_data_check" options set and enabled we just have to wait a while and watch the graphs grow.



HAZA! Although we are not performing any evaluations the script is collecting Memory Utilization, Thread Count, IO-Read, IO-Write, CPU Usage every 5 minutes.



We now have enough performance counters for the NSClient++ service to make even the most detail oriented number cruncher happy.

Wrapping up and moving forward.

We are now monitoring the Runtime state of the NSCP service, The Start Type of the NSCP Service, The last update of the log file associated with NSCP and collecting performance data counters for NSCP with a resolution of 5 minutes.

All with a PowerShell Script, Windows Scheduled Task and NRDP.

Want more information about the Nagios, Windows, PowerShell or the methods I used in this example?

Have a look at the links below.