Monitoring Baselines with Nagios

Steven Cardinal

Monitoring Baselines with Nagios

GIAC GSNA Gold Certification

Author: Steven Cardinal, steven.cardinal@gypsywagon.com

Advisor: Richard Carbone

Accepted: February 1, 2015

Abstract

System administrators have many tasks, from maintaining uptime to preparing for disaster. They must also ensure systems adhere to security guidelines such as the DoD's Security Technical Implementation Guides (STIG). Other than an audit, which most system administrators dread, how can one be sure systems remain in compliance while attentions are focused on day-to-day maintenance tasks? Monitoring – and Nagios provides that with customization options that let a user monitor changes to baselines on remote systems. This paper will focus on monitoring Windows and Linux baselines. It will look at using the Nagios Cross Platform Agent (NCPA) to perform checks between the Nagios server and the monitored hosts as well as looking at some sample, custom scripts to be used for checking configuration settings.

1. Introduction

It is 4:00 on a Friday afternoon and you, a system administrator for a large, multinational entertainment company, are putting your things away to head out for a long holiday weekend. When your desk phone rings, you get a sudden sinking feeling; this does not bode well for your immediate future. Answering it, you hear the frustrated voice of a senior manager. He needs to get a report out to a customer before the end of the day and the new system is telling him "access denied."

Troubleshooting this new reporting system could take a considerable amount of time, something neither you nor the angry manager have. You make the decision to relax the permissions in the application so that the report can be generated. You vow to put the time and effort into a better solution when you come back on Tuesday.

When Tuesday comes, however, your task list expands as all of the normal, automated jobs for Monday failed because of the holiday. The day-to-day operations continue to take precedence, and the CRM fix gets pushed further and further down the list. Eventually it is forgotten, as the CRM system is running trouble-free. The risk, however, remains. It may be discovered during the annual audit, or it may be discovered by actors of a less savory sort.

Similar scenarios take place throughout businesses everywhere. Information systems are taking on greater complexity while administration resources are stretched thin. Personnel are being asked to work more hours to support more systems. According to the Occupational Outlook Handbook (Bureau of Labor Statistics, 2014), published by the US Department of Labor's Bureau of Labor Statistics, about 25% of system administrators work more than 40 hours per week.

Perhaps the organization has not yet implemented strong Security-focused Configuration Management (SecCM) practices, but the administrators need to get a handle on changes made throughout the computing environment. The National Institute of Standards and Technology, or NIST, has stated that (Johnson, Dempsey, Ross, Gupta, & Bailey, 2011), "these changes can adversely impact the previously established security posture; therefore, effective configuration management is vital to the establishment and

Steven Cardinal, steven.cardinal@gypsywagon.com

maintenance of security of information and the information system." It is therefore critical that the organization takes steps to improve visibility of these systems.

Maintaining a secure configuration is part of The SANS Institute's Critical Security Controls. According to Control 3 (SANS Institute, 2015), one should "implement and test an automated configuration monitoring system that measures all secure configuration elements that can be measured through remote testing." This further enforces the idea that a monitoring system would be beneficial.

You consider methods to implement such a system, but prices are high, as is complexity. You might consider doing it by hand but according to the NIST Special Publication 800-137 (Dempsey, et al., 2011), "automated processes, including the use of automated support tools (e.g., vulnerability scanning tools, network scanning devices), can make the process of continuous monitoring more cost-effective, consistent, and efficient."

You go back to your search engine of choice. You find solutions in the marketplace, such as those from SolarWinds, Tenable, and Qualys, but you are budget-challenged. While your audit department may have access to those tools, you would like to know about configuration problems before they get involved.

It would be helpful if there was a way to track the baseline security of your systems. It would be convenient to receive an alert when an unexpected change to the security posture of a system occurs. It would be great to have such a monitoring system that does not require a huge budget. Fortunately, there is an open source software package, designed to run on an open source operating system that could do all this without breaking the bank. Nagios.

Yes, that free piece of software you likely already have running to let you know when a system goes down can be provisioned to handle this task. If you are not currently using Nagios, you may wonder what is Nagios.

Steven Cardinal, steven.cardinal@gypsywagon.com

2. Implementing Nagios

2.1. Nagios Implementation Overview

Nagios is an open source system monitoring application that was originally created in 1999 under the name Netsaint. It was renamed in 2003. It consists of a core server component, which initiates system checks using various plugins, and reports the results through a web interface. Optionally, it can report via numerous alerting options, such as text and email. In addition to the plugins, Nagios supports a number of different services that can provide active and passive remote checks.

Nagios comes in both commercial and free versions, the latter known as Nagios Core. This paper will cover the free version. Further information about the Nagios' commercial offerings can be found at http://www.nagios.com. Please note that, although Nagios Core v4 is now available, most major Linux distributions still ship with v3.5, and that is what this paper will focus on.

The examples discussed in this paper are based on a test environment consisting of three servers: a CentOS Nagios server, a CentOS webserver, and a Windows 2008 R2 server. All are running as virtual hosts atop VMWare. Each host will have its built-in firewall enabled and configured to permit Nagios monitoring with a minimal set of open ports.

Although there are a number of Nagios agents available for various platforms, this paper makes use of the newer Nagios Cross Platform Agent (NCPA). This agent uses a standard API across platforms, allowing for the uniform checking of most popular system statistics, such as CPU, memory, and disk utilization.

Another factor in choosing NCPA is its ability to provide SSL encryption for all network traffic between the Nagios server and the agent which requires an authentication string before checks are permitted. Monitored systems may be transmitting sensitive information about the configuration of critical servers and even internal networks are not guaranteed to be secure. An article published by the Electronic Frontier Foundation makes note of the infiltration of internal networks by government spy agencies. They state that (Galperin & Schoen, 2013), "encryption even between the devices within a data center may be an important precaution if the routers and switches connecting those

devices can be targeted with malware." Even without fear of government spying, it is prudent to recognize that they are not the only actors with these capabilities.

2.2. Configuring Windows

Performing checks on Microsoft Windows systems will require a mix of PowerShell scripts and command-line utilities. While scripts using older, "DOS Batch" commands will work, and will be used as wrappers for a number of command-line tools, there is considerable power native to the PowerShell scripting language. This will allow a system administrator to perform a large number of checks with a minimum of code.

When it comes to using PowerShell scripts, however, Windows ships with a default configuration preventing their execution. According to Microsoft (Microsoft, 2014), "the execution policy is not a security system that restricts user actions… the execution policy helps users to set basic rules and prevents them from violating them unintentionally." Considering that a layered approach to security is a best practice, the system will be configured to permit the execution of scripts in the most restrictive manner available.

Enabling the Nagios agent to execute PowerShell scripts will require changing some of the default behaviors of Windows. To permit PowerShell scripts to be executed, the system administrator will need to change the execution policy and, since it is not desirable to diminish the security posture of the system excessively, it will be required that all PowerShell scripts be signed.

Signing PowerShell scripts is not particularly difficult. One could use the Certificates MMC snap-in to generate a code-signing certificate. Moreover, it is advisable to make the private key exportable so that the certificate can be used on multiple systems. Begin by installing the Certificates snap-in for the Current User in MMC as shown in Figure 1:

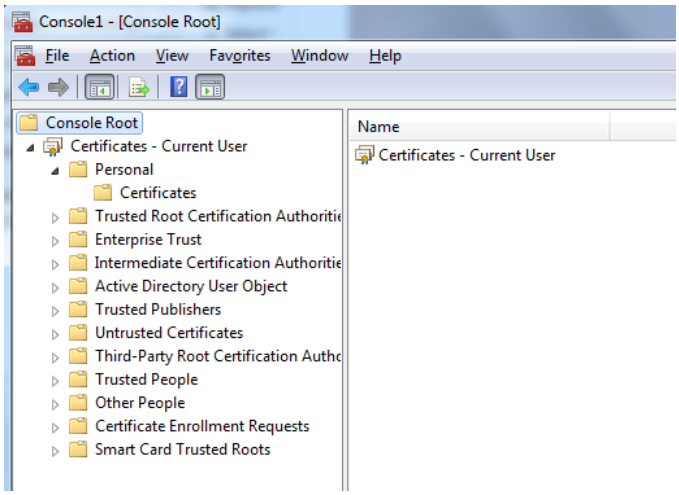

Figure 1: Certificates MMC Snap-in

Right-click the Certificates folder under Personal and choose All Tasks -> Advanced Operations -> Create Custom Request. Then click Next and Proceed without Enrollment Policy and click Next. *Note: If there is a corporate PKI infrastructure, there may be preconfigured Enrollment Policies. Follow any corporate standards for requesting a code-signing certificate.*

Leave the default Template **(No template) CNG key** and Request format **PKCS #10** and click Next. Expand the Details of the Custom request and click Properties. Fill out the following five screens, ensuring they meet any corporate or organizational standards relevant to the environment, as shown in figures 2 through 6:

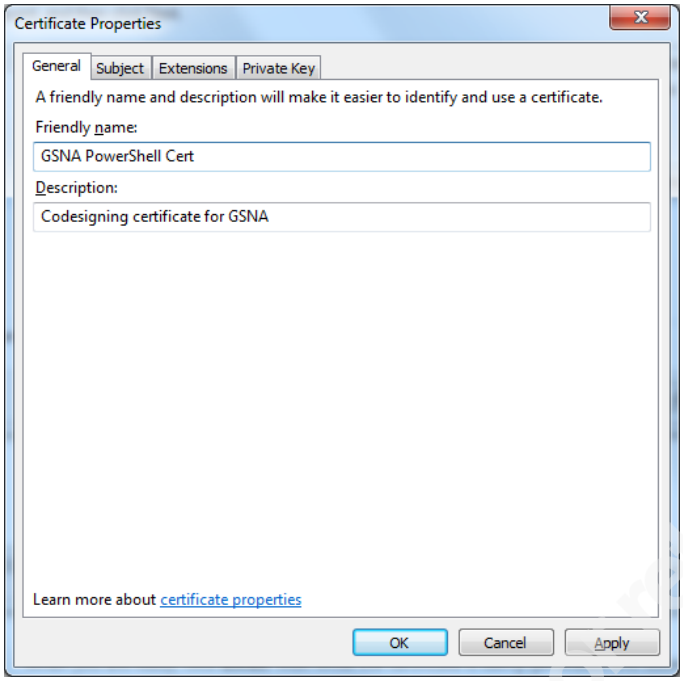

Figure 2: Certificate General Properties

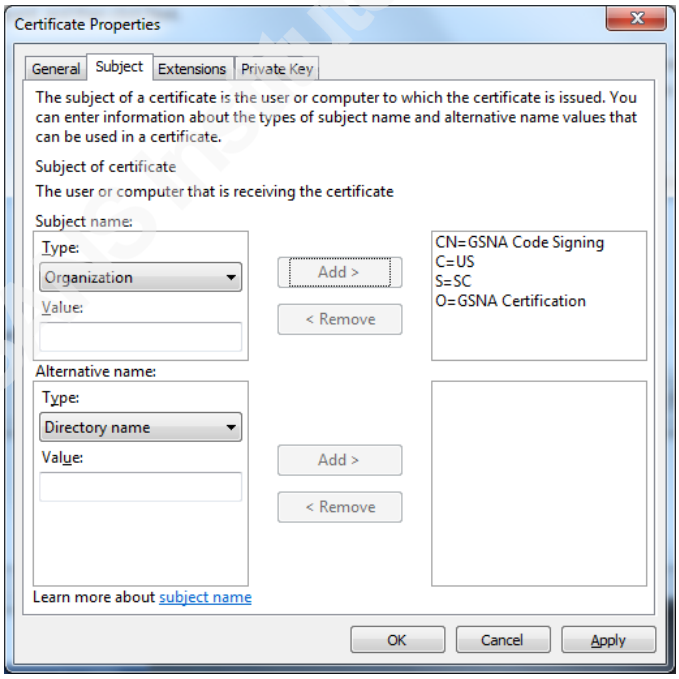

Figure 3: Certificate Subject Properties

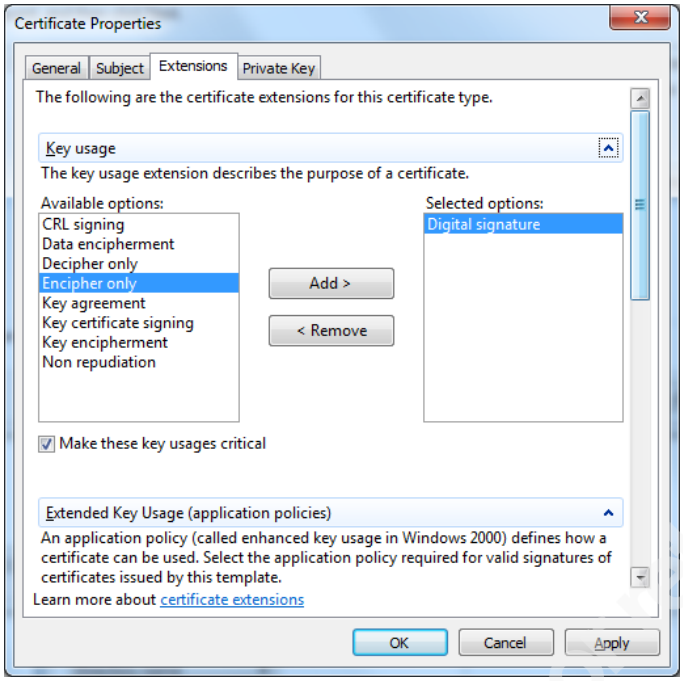

Figure 4: Certificate Key Usage Properties

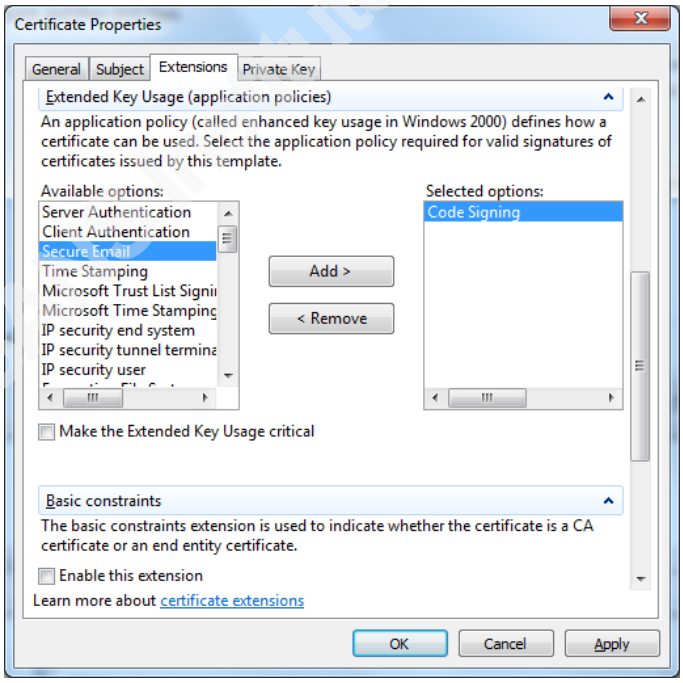

Figure 5: Certificate Extended Key Usage Properties

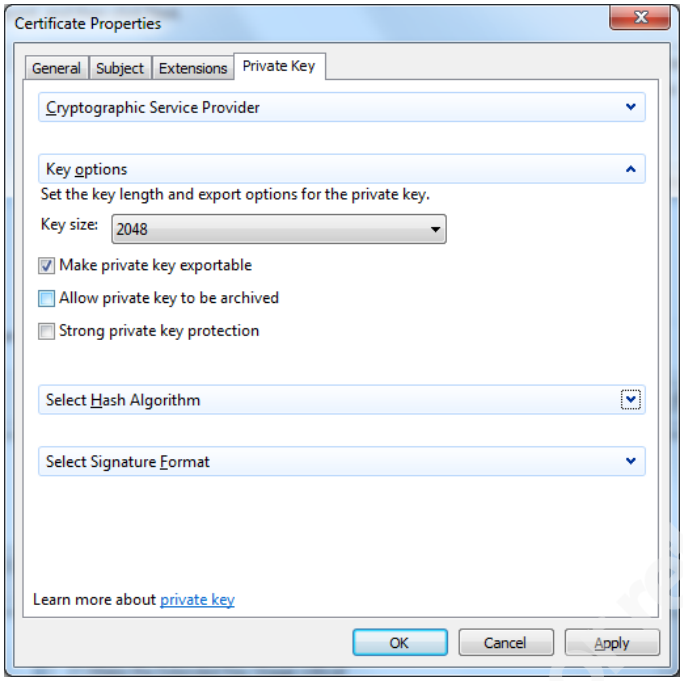

Figure 6: Certificate Private Key Properties

Click OK and save the certificate. At this point, provide the certificate request to a CA and, when the certificate is ready, install it on the servers in the Personal area of the certificate store. Navigate to the Personal\Certificates folder in the MMC Certificates snap-in, choose Import, and follow the prompts. This will allow PowerShell to find the certificate to sign the script. The system, however, will need to trust the certificate in order to execute the script.

For an enterprise with a PKI infrastructure, the code-signing certificate will likely be distributed via Group Policy Object (GPO) to the Local System\Trusted Publishers folder in the certificate store. For the purposes of this paper, manually install it there.

Within MMC, add the Certificates snap-in, this time using Local Computer as the target. Navigate to Trusted Publishers, right-click and choose All Tasks -> Import. Then navigate to the code-signing certificate imported earlier and complete the wizard.

Now that there is a trusted signing certificate, the system administrator is ready to sign scripts. Given a script called *MyScript.ps1* in the current directory and a certificate with the string GSNA in the Subject properties (the CN entry is shown in Figure 3 above); this can be done using the following syntax:

```
PS1> $codeCert = dir cert:currentuser\My | where {$_.Subject -like "*GSNA*"}
PS1> Set-AuthenticodeSignature .\MyScript.ps1 $codeCert
```

Note that using the above code will result in a script that will cease working if/when the certificate expires. There are numerous timestamp options available that should be considered for the needs of the environment, including setting the script to never expire.

Now that there is a code-signing certificate and some signed code, the system administrator can set the execution policy for the server. This needs to be set from within PowerShell running with administrative privileges. Locate the Windows PowerShell utility on the Start menu, right-click on it and choose Run as administrator. When prompted by User Account Control (UAC), click Yes. At the prompt, execute:

```
PS1> Set-ExecutionPolicy AllSigned
```

The system is now able to execute signed (and only signed) PowerShell scripts. Next, install the NCPA Nagios Agent so that the monitoring server can communicate with the Windows system and perform checks. Prior to installation, however, configure Windows to allow the agent to run correctly. If it is not already enabled, enable the disk performance counters, as the current installer (v1.7.2) will silently fail to install the services if they are not enabled.

```
C:\> diskperf -y
```

Next, configure the Windows Firewall to permit communications with the agent. It is recommended that communication to the agent be restricted to the Nagios server. Open a Command Prompt as Administrator and run the *netsh* command, substituting the IP Address for the Nagios server.

Steven Cardinal, steven.cardinal@gypsywagon.com

```
C:\> netsh advfirewall firewall add rule name="Allow NCPA" dir=in action=allow
protocol=TCP localport=5693 remoteip=<nagiosip>
```

After downloading the latest NCPA installer from the Nagios website (http://assets.nagios.com/downloads/ncpa/download.php), install it, providing it with an Active Specifications Token, which is used as a form of authorization. This example uses the following token, **b14c23a71d05e69**. Following a successful installation one should see port tcp/5693 listening when *netstat* is run. Next, configure a Linux host to run the NCPA agent.

2.3. Configuring Linux

First, install the NCPA Nagios Agent so that the monitoring server can communicate with the Linux system and perform checks. After downloading the latest NCPA installer RPM from the Nagios website, install it and then edit the configuration file in */usr/local/ncpa/etc/ncpa.cfg*. Locate the *community_string* entry in the *[api]* section and set it to use the following token, **b14c23a71d05e69**. Restart the *ncpa_listener* service and then permit access to the listener through the *iptables* firewall. As on Windows, restrict communications to only the IP address of the Nagios server:

```
$ sudo service ncpa_listener restart
$ sudo iptables -I INPUT 2 -p tcp --dport 5693 -s <nagiosip> -j ACCEPT
```

That is all that is needed on Linux to get the agent running. Now that it is running scripts will need to be created to perform checks and report them back to the Nagios server.

2.4. Writing Nagios Plugins

The Nagios server expects messages from plugins to be formatted in a particular manner. Following the guidelines published at https://nagios-plugins.org/doc/guidelines.html, one can see that any plugin written should output a short but descriptive

message that will be included in any alerts sent to monitoring personnel. Plugins must also return an error code indicative of the service status, as shown in the following table:

Table 1: Plugin Return Codes (Nagios Plugins Development Team, 2015)

Numeric Value	Service Status	Status Description
0	OK	The plugin was able to check the service and it appeared to be functioning properly
1	Warning	The plugin was able to check the service, but it appeared to be above some "warning" threshold or did not appear to be working properly
2	Critical	The plugin detected that either the service was not running or it was above some "critical" threshold
3	Unknown	Invalid command line arguments were supplied to the plugin or low-level failures internal to the plugin (such as unable to fork, or open a tcp socket) that prevent it from performing the specified operation. Higher-level errors (such as name resolution errors, socket timeouts, etc) are outside of the control of plugins and should generally NOT be reported as UNKNOWN states.

With these guidelines in mind, a system administrator can develop numerous scripts on both Windows and Linux to perform baseline checks and report status back to the Nagios server.

2.5. Windows Check Scripts

Start with an example script to query a registry value. To test a registry entry using Windows PowerShell, one can use the Get-ItemProperty of any setting. For instance, to check that the system requires Strong Keys for NetLogon, use the following:

```
PS1> Get-ItemProperty -Path hklm:system\currentcontrolset\services\netlogon\Parameters -
Name RequireStrongKey
```

The output of this command is not Nagios-friendly, however. It does not return data as a simple string nor with the appropriate return code. A better solution is to write a script that outputs a simple message and an error code and let the Nagios agent pass

parameters to it. For a Registry check, pass in a Registry Key, the value to check, and the desired setting. Have an affirmative check return success and a negative check return failure. This can be accomplished with the following bit of code:

```
#
# Validate Registry Setting
# Pass in a Registry key, value, and expected setting
# ok: return 0
# bad: return current setting
#
$regkey = $args[0]
$regval = $args[1]
$regset = $args[2]
$actual = (Get-ItemProperty -Path $regkey).$regval
if ($actual){
        if ($actual -eq $regset){
                Write-Host "OK"
                exit 0
        } else {
                Write-Host "WARNING - Value is $actual should be $regset"
                exit 1
        }
} else {
        Write-Host "CRITICAL - Value NOT FOUND"
        exit 2
}
```

Save the above code as a PowerShell script, sign it using the Authenticode certificate, and copy it to the plugins folder in the NCPA installation (by default, *C:\Program Files (x86)\Nagios\NCPA\plugins*) will make the plugin available to the NCPA agent and, therefore, to Nagios.

Not everything can be accomplished using PowerShell, however. In Windows 2008R2, checking firewall rules still requires the use of *netsh*. Create a batch (.bat) script, therefore, to execute *netsh* and return both a message and proper exit code. This example will check for the presence of a custom firewall rule that permits access to a local SQL Server installation running on the default tcp/1433. The rule is first created:

Steven Cardinal, steven.cardinal@gypsywagon.com

```
C:\> netsh advfirewall firewall add rule name="Allow SQL 1433" dir=in action=allow
protocol=TCP localport=1433
```

Check for the existence of the rule by running:

```
C:\> netsh advfirewall firewall show rule ="Allow SQL 1433"
```

That command provides multi-line output, however. While that is convenient for a human, it is not quite as useful for a script. A better script will check to ensure that the *netsh* command does not result in an error condition, such as when an invalid rule name is passed. If it does not result in an error, parse the output looking for the Enabled parameter being equal to "Yes". Finally, once the script has determined the state of the firewall rule it will return 0 for a firewall rule that is properly enabled, 1 for a rule that is disabled, and 2 for any other error. Refer to Appendix A.1 for the complete ***Error! Reference source not found.*** script.

Using the PowerShell and batch scripts as examples, one should be able to create scripts to check most settings on a Windows system. One can also check the state of the Linux systems.

2.6. Linux Check Scripts

Checking baseline settings on Linux are similarly simple and can use a variety of scripting options. While Perl and Python are commonly used scripting languages, for this paper the ubiquitous *bash* shell will be used.

As an example, this *bash* script will use the *awk* command to check a parameter within a standard configuration file, such as *sshd_config*. Configuration files are typically simple text files, many of which use a simple:

parameter value

format. For configuration files such as these, use a script that accepts the file name, the parameter, and the expected value as inputs. Again, ensure that the output of the script and return codes meet that expected by Nagios. The entire script can be found in Appendix A.2 in the ***Error! Reference source not found.*** script.

In this case, the *CheckCfg.sh* script will use *awk* to parse a standard configuration file looking for the setting to be verified. The final script should make use of as much error checking as appropriate for the environment. The purely functional bit of code looks like this:

```
FILENAME=$1
PARAM=$2
VALUE=$3

awk '{ if ( $0 !~ /^($|#)/ && $1=="'$PARAM'" && $2=="'$VALUE'" ) {exit 9} }' $FILENAME
```

2.7. Configure Nagios

The hosts can now be monitored: the security posture of each system has adjustments, scripts written, and the Nagios agent configured. Without a system to provide monitoring and alerting, though, the goal of configuration monitoring has not been achieved. While the actual installation of Nagios Core varies from platform to platform, the post-installation configuration is the same.

By default, Nagios is configured to require authentication to access the monitoring console. The default user account is nagiosadmin, and this is set in file */etc/nagios/cgi.cfg*. In association with that, the Nagios website is configured in */etc/httpd/conf.d/nagios.conf* to require Basic Authentication with a user account and password found in */etc/nagios/passwd*. For simplicity, create the account using:

```
$ sudo htpasswd -c /etc/nagios/passwd nagiosadmin
```

As always, follow any corporate standards for user authentication. If Basic Authentication is used, one should also enable Apache to provide SSL protection for the Nagios website. That, however, is beyond the scope of this paper.

Start Apache and Nagios and make sure they are enabled upon boot. On CentOS, the commands are:

```
$ sudo service httpd start
$ sudo service nagios start
$ sudo chkconfig httpd on
$ sudo chkconfig nagios on
```

If accessing the Nagios console remotely, open a port in the firewall to permit access. If using IPTables, the following command on CentOS should suffice. It will insert a new rule at the beginning of the INPUT chain. Remember to add it to the configuration to make the change permanent using the **service iptables save** command.

```
$ sudo iptables -I INPUT 1 -p tcp --dport 80 -j ACCEPT
$ sudo service iptables save
```

At this point, it should be possible to direct a browser to the monitoring server using a URL similar to http://myserver/nagios. If everything is working as expected, there will be a prompt for the nagiosadmin account and password created earlier. After authentication, the server should display the Nagios Core home screen.

Nagios, being both complex and flexible, provides for a myriad of configuration options. This paper uses a simple configuration for demonstration purposes. Begin by defining two hosts to be monitored, creating a check for each host to test the scripts created earlier, and specifying an email address to receive notifications. This last part will be addressed first.

Using a text editor, such as *vim*, edit the */etc/nagios/contacts.cfg* file and locate the email setting for the default nagiosadmin account. Assuming said account was created according to the previous instructions, setting this to a valid email address will enable notifications. If a different account was created, update the configuration file as required for this user. Save the file when finished.

For demonstration purposes, this procedure will create a single configuration file in directory */etc/nagios/conf.d/* to define the hosts, hostgroups, and checks. It will be called *gsna.cfg*. In an enterprise environment, divide these up according to best practices. See Appendix A.3, for the complete configuration file used.

To check hosts using NCPA, the *check_ncpa.py* script must be downloaded from the Nagios Github site (https://raw.github.com/NagiosEnterprises/ncpa/master/client/check_ncpa.py), copied to */usr/lib/nagios/plugins*, and made executable (*chmod* 755). Note, if SELinux is enabled, one should ensure that the label on the plugin matches those on the other plugins. On CentOS 6, if the script is downloaded directly into the plugins directory it is likely the label will be incorrect. To remedy this, the *restorecon* command should be executed:

```
$ sudo restorecon -Rv /usr/lib/nagios
```

Once the script is installed, and any permissions adjusted, create a check command so that Nagios knows how to initiate tests using NCPA. In the *gsna.cfg* file, add the following check command:

```
# 'check_ncpa' command definition
define command{
        command_name    check_ncpa
        command_line    $USER1$/check_ncpa.py -H $HOSTADDRESS$ -t b14c23a71d05e69 -P 5693
-M $ARG1$ -a $ARG2$
        }
```

This tells Nagios that, for any host configured with a *check_ncpa* command, launch the *check_ncpa.py* plugin passing the parameters for the host name, the Active Specifications Token (or *community_string*) of **b14c23a71d05e69**, the port to connect to, the API path to the check script to execute, and any arguments supplied. Add a service statement to associate a particular check, such as the Windows Firewall check described earlier, with a particular host. In this case, the Windows system was used:

```
define service{
        use                             local-service
        host_name                       winserver
        service_description             SQL-FIREWALL
        check_command                   check_ncpa!agent/plugin/CheckFW.bat!'"Open SQL
Server Port 1433"'
        }
```

Of particular note in this service definition is that each parameter passed to the *check_command* is separated by the "!" symbol and that the parameters passed to the PowerShell script must be both single and double-quoted. Different scripting languages may have different quoting requirements. The Linux SSH check covered below shows evidence of that.

Once Nagios is reloaded, to read the new configuration, it should only take a few minutes before reporting the results. A successful check will appear as:

Figure 7: Nagios Successful Check

Add an additional check command to the configuration to verify the Linux system. The following service check will verify that the SSH server does not permit a root login:

```
define service{
        use                     local-service
        host_name               linserver
        service_description     SSH-PERMITROOT
        check_command           check_ncpa!agent/plugin/checkcfg.sh!'/etc/ssh/sshd_config
PermitRootLogin no'
        }
```

Note that for Linux systems, which are case-sensitive, NCPA represents any installed plugins as all lowercase. So, even if a given check script is named using mixed-case, it will need to be referenced in all lowercase in the Nagios check commands. In the check command, be sure to use lowercase as well.

At this point, there are two hosts to monitor and a check for each of the hosts to prove that NCPA is working as desired. Now comes the work of building all the checks required for the environment.

2.8. Deciding What to Check

Considering that there may be hundreds if not thousands of baseline settings that could be monitored, which are the most critical? Consider that if the site is in a U.S. governmental organization, standards enforced via a Security Technical Implementation Guide, or STIG, may be required. According to the Defense Information Systems Agency (DISA), a STIG is used to provide (Defense Information Systems Agency, 2014), "secure configuration guidance for a product to reduce the attack surface." These STIGs can be accessed at http://iase.disa.mil/stigs. Note that some STIGs require DoD authentication while others do not. For a private organization, those publicly available STIGs would be a fine starting point.

Security checks within each STIG are prioritized by a Category of 1, 2, or 3 (think High, Medium, and Low). A great place to start would be to monitor any of those settings

designated Category 1. These are the types of findings that can get a system shutdown immediately by the organization's security personnel.

What if this is not a government agency or the organization cannot (or does not want to) use the STIGs? Starting with a Risk Assessment would make sense. Consider the triad of security: Confidentiality, Integrity, and Availability. What are the greatest risks to the systems in regards to these three areas?

For confidentiality, what settings have been put in place to meet the system's needs? Checking for the use of only approved authentication mechanisms would be a start. Check that firewall rules stay in place to ensure connectivity from only trusted systems, when required. Also consider the password and account policies that have been implemented. How does one make sure they have not been relaxed?

Ideally, these settings will be managed centrally, such as through Group Policy Objects in Windows Active Directory, but what if the GPO is changed, such as in the example scenario? An appropriately configured Nagios check will not only alert the necessary personnel to the change, it will also keep that notification active until it is addressed.

Integrity checks could include the results of a file integrity checker, such as AIDE or Tripwire. In addition, checking security permissions on critical files would be a recommended check to implement, since inappropriate permissions could lead to unauthorized changes or system access.

If system availability is a major concern, will there be monitoring of system patches, successfully applied or otherwise? Are system updates configured according to an approved schedule? Are backups running as expected or did something get turned off during a troubleshooting exercise and never turned back on again?

If using a system configuration management tool, such as Group Policy Objects or a third party tool such as Puppet, begin by combing through the existing rules and determine which would introduce the greatest risk if disabled or changed. Gradually increase coverage until there is confidence that the monitoring of key security settings of the system has been achieved.

Steven Cardinal, steven.cardinal@gypsywagon.com

3. Conclusion

Best practices dictate that system security follows the Defense-in-Depth strategy. What is Defense-in-Depth? The National Security Agency (NSA) defines it as (National Security Agency, 2015), "a 'best practices' strategy in that it relies on the intelligent application of techniques and technologies that exist today… [and] recommends a balance between the protection capability and cost, performance, and operational considerations." This means multiple layers of security should be implemented such that the failure of any one control does not lead to a system breach. Such a strategy, however, increases the complexity of a computing system and therefore increases the difficulty in supporting that system. To reduce the accompanying strain on technology personnel, many organizations divide the Information Technology department into silos of technical knowledge: servers, desktops, networks, security, Windows, Linux. The list goes on and on.

With these silos comes a decrease in business communication, and when communication breaks down, the security of the information systems is at risk. By using existing or easily acquired tools, each part of the IT organization can maintain visibility of the system in a holistic manner. For the system administrator, using a free tool, such as Nagios, can not only improve their ability to support their systems, but they can also maintain them in such a way as to meet the expectations of other departments such as the Information Assurance group.

If an organization is lagging in its auditing of systems and their security posture, the system administrator may be able to help. Implementing a Nagios monitoring system, whether the free or commercially supported version, can provide value to the department and the organization. In a world in which IT can no longer be just a build-and-fix department, such an administrator will be set up for future success.

Steven Cardinal, steven.cardinal@gypsywagon.com

4. References

Bureau of Labor Statistics. (2014, December 20). *Occupational Outlook Handbook, 2014-15 Edition, Network and Computer Systems Administrators*. Retrieved from U.S. Department of Labor: http://www.bls.gov/ooh/computer-and-information-technology/network-and-computer-systems-administrators.htm

Defense Information Systems Agency. (2014, November 12). *FAQs*. Retrieved from Information Assurance Support Environment: http://iase.disa.mil/stigs/Pages/faqs.aspx#STIG

Dempsey, K., Chawla, N. S., Johnson, A., Johnston, R., Jones, A. C., Orebaugh, A., . . . Stine, K. (2011). Information Security Continuous Monitoring (ISCM) for Federal Information Systems and Organizations. Gaithersburg, MD, United States of America.

Galperin, E., & Schoen, S. (2013, October 30). *The PRISM is Not Enough: Government Spies on Google and Yahoo's Internal Networks*. Retrieved from Electronic Frontier Foundation: https://www.eff.org/deeplinks/2013/10/prism-is-not-enough

Johnson, A., Dempsey, K., Ross, R., Gupta, S., & Bailey, D. (2011, August). Guide for Security-Focused Configuration Management of Information Systems. Gaithersburg, MD, United States of America.

Microsoft. (2014, December 3). *about_Execution_Policies*. Retrieved from Microsoft TechNet: http://technet.microsoft.com/en-us/library/hh847748.aspx

Nagios Plugins Development Team. (2015, January 2). *Nagios Plugin Development Guidelines*. Retrieved from Nagios Plugins: https://nagios-plugins.org/doc/guidelines.html

National Security Agency. (2015, January 15). *None.* Retrieved from National Security Agency: https://www.nsa.gov/ia/_files/support/defenseindepth.pdf

SANS Institute. (2015, January 9). *SANS INstitute - Critical Security Control: 3*. Retrieved from SANS Institute web site: https://www.sans.org/critical-security-controls/control/3

Steven Cardinal, steven.cardinal@gypsywagon.com

Appendix A. Scripts and Configuration Files

Any scripts or configuration files that were referenced or partially covered in the document are contained herein for completeness.

Appendix A.1 CheckFW.bat

The following is for Windows batch file *CheckFW.bat*, previously mentioned in this document:

```
@ECHO OFF
REM
REM Validate Firewall Rule is enabled
REM Pass in a firewall rule name
REM enabled return 0
REM disabled return 1
REM Other error, including rule not found return 2
REM

SET VAL=
SET RETVAL=

netsh advfirewall firewall show rule %1 > nul

SET RETVAL=%ERRORLEVEL%

IF %RETVAL% NEQ 0 GOTO :CRIT

FOR /F "tokens=1,2" %%i IN ('netsh advfirewall firewall show rule %1') DO ^
IF "%%i"=="Enabled:" SET VAL=%%j

IF %VAL% NEQ Yes GOTO WARN

:SUCCESS
ECHO OK
EXIT 0
GOTO END

:WARN
ECHO WARN - Rule Not Enabled
EXIT 1
GOTO END
```

```
:CRIT
ECHO CRITICAL - Check Failed
EXIT 2
GOTO END

:END
```

Appendix A.2 CheckCfg.sh

The following is for Linux script file *CheckCfg.sh*, previously mentioned in this document:

```
#!/bin/bash

FILENAME=$1
PARAM=$2
VALUE=$3

if [ -z "$FILENAME" ] || [ -z "$PARAM" ] || [ -z "$VALUE" ]
then
        echo Invalid syntax
        echo Syntax $0 filename parameter value
        echo UNKNOWN
        exit 3
elif [ ! -f "$FILENAME" ]
then
        echo $FILENAME not a file
        echo Syntax $0 filename parameter value
        echo UNKNOWN
        exit 3
else
        awk '{ if ( $0 !~ /^($|#)/ && $1=="'$PARAM'" && $2=="'$VALUE'" ) {exit 9} }'
$FILENAME
        if [ $? -eq 9 ]
        then
                echo OK
                exit 0
        else
                echo WARN
                exit 1
```

```
        fi
fi
```

Appendix A.3 gsna.cfg

The following is for Nagios configuration file *gsna.cfg*, previously mentioned in this document:

```
define host{
        use             linux-server
        host_name       linserver
        alias           GSNA Linux Srv
        address         10.0.1.15
        }

define host{
        use             windows-server
        host_name       winserver
        alias           GSNA Windows
        address         10.0.1.2
        }

# 'check_ncpa' command definition
define command{
        command_name    check_ncpa
        command_line    $USER1$/check_ncpa.py -H $HOSTADDRESS$ -t b14c23a71d05e69 -P 5693
-M $ARG1$ -a $ARG2$
        }

define service{
        use                     local-service
        host_name               linserver
        service_description     SSH-PERMITROOT
        check_command           check_ncpa!agent/plugin/checkcfg.sh!'/etc/ssh/sshd_config
PermitRootLogin no'
        }

define service{
        use                     local-service
        host_name               linserver
        service_description     SSH-USEPAM
        check_command           check_ncpa!agent/plugin/checkcfg.sh!'/etc/ssh/sshd_config
```

```
UsePAM yes'
        }


define service{
        use                     local-service
        host_name               winserver
        service_description     STRONGKEY
        check_command
check_ncpa!agent/plugin/CheckReg.ps1!'hklm:system/\currentcontrolset/\services/\netlogon/
\parameters RequireStrongKey 1'
        }


define service{
        use                      local-service
        host_name                winserver
        service_description      SQL-FIREWALL
        check_command            check_ncpa!agent/plugin/CheckFW.bat!'"Open SQL Server
Port 1433"'
        }
```