Icinga2 - Windows Monitoring

Using Powershell

Powershell is a very nice and native way execute Checks on Windows. It works on any Windows since Windows Server 2008.

Adding Powershell Checks to Icinga

Here are some examples, how to add Powershell checks to icinga2. There are some things you must be aware of:

- „**-command**“ must be the first argument So use order command to change the order.
- „to get always the correct exitcode, you need to add the “;exit„ at the end. $LASTEXITCODE is a Powershell variable. Credits for that enhancement goes to the monitoring community !
- If you have a 32-bit icinga Agent it will use the 32-bit Powershell. Use „C:\\Windows\\sysnative\\WindowsPowerShell\\v1.0\\powershell.exe“ to execute x64 Powershell

```
object CheckCommand "powershell" {
    import "plugin-check-command"
    command = [
"C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\powershell.exe" ]
    arguments = {
        "-command" = {
            value = "$ps_command$"
            order = -1
        }
        "-args" = {
             value = "$ps_args$"
             order = 98
        }
        ";exit" = {
             value = "$$LASTEXITCODE"
             order = 99
        }
    }
}
object CheckCommand "powershell-x64" {
    import "plugin-check-command"
    command = [
"C:\\Windows\\sysnative\\WindowsPowerShell\\v1.0\\powershell.exe" ]
    arguments = {
        "-command" = {
            value = "$ps_command$"
            order = -1
```

```
        }
        "-args" = {
             value = "$ps_args$"
             order = 98
        }
        ";exit" = {
             value = "$$LASTEXITCODE"
             order = 99
        }
    }
}


object CheckCommand "files-windows-director" {
    import "plugin-check-command"
    command = [
        "C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\powershell.exe"
    ]
    arguments += {
        "-args" = {
            order = 98
            repeat_key = false
            required = true
            value = "$files_windows_args$"
            //Content of $files_windows_args$ can be something like '-path
"C:\\" -unit "days" -o10'
        }
        "-command" = {
            order = -1
            repeat_key = false
            required = false
            value = "& 'C:\\Program Files\\ICINGA2\\sbin\\check_files.ps1'"
        }
        ";exit" = {
            order = 99
            required = true
            value = "$$LASTEXITCODE"
        }
    }
}

object CheckCommand "vmware_snapshot_size" {
    import "plugin-check-command"
    command = [
        "C:\\Windows\\system32\\WindowsPowerShell\\v1.0\\powershell.exe"
    ]
    arguments += {
        "-command" = {
            order = -1
            value = "& 'C:\\Program
Files\\ICINGA2\\sbin\\check_vmware_snapshot_size.ps1'"
```

```
        }
        "-CredFile" = "$vmware_powercli_credfile$"
        "-GuestExclude" = "$vmware_snapshot_size_guestexclude$"
        "-HostExclude" = {
            required = false
            value = "$vmware_snapshot_size_hostexclude$"
        }
        "-allcrit" = {
            required = false
            value = "$vmware_snapshot_size_allcrit$"
        }
        "-crit" = {
            required = false
            value = "$vmware_snapshot_size_crit$"
        }
        "-hostname" = {
            order = 1
            required = true
            value = "$service_address$"
        }
        "-warn" = {
            required = false
            value = "$vmware_snapshot_size_warn$"
        }
        ";exit" = {
            order = 99
            required = true
            value = "$$LASTEXITCODE"
        }
    }
    vars["Vmware Powercli Credfile"] =
"C:\\\\programdata\\\\var\\\\log\\\\$address$_vmvare.credfile"
    vars.vmware_snapshot_size_allcrit = "20"
    vars.vmware_snapshot_size_crit = "8"
    vars.vmware_snapshot_size_warn = "6"
}
```

Using Another User with Encrypted Logins/Passwords

If you want to execute commands with another User than the Icinga2 service account, its a good idea to encrypt the passwords for it. Powershell supports that natively. All Passwords that are encrypted with this function will only be decrypted successfully on the same host with the same user.

Script example to encrypt/decrypt Logins

```
#Credential auth file example
```

```
V1.0


$authfile = "C:\Program Files\icinga2\myexample_credentials.xml"

function get_authfile ($authfile) {
  if (test-path $authfile) {
   # This output will send to Icinga, to notify the admin, that he needs to
execute the check once:
   write-host "Authfile not exists. You need to run  --> `"C:\Program
Files\ICINGA2\sbin\psexec`" -i -s Powershell.exe -command `"& 'C:\Program
Files\ICINGA2\sbin\check_myCheck_example'`" -args `"-myarg $arg1 -authfile
$authfile`" <--- (including all kind of quotes!) once to create it!"
    $credentials=IMPORT-CLIXML $authfile
  } else {
    $credentials=GET-CREDENTIAL –Credential “icinga-read-user” | EXPORT-
CLIXML $authfile
  }
  return $credentials
}


#Get Credentials
$login = get_authfile $authfile

#Now this $login can be used for all "-Credential" based logins. If you have
a Script that don't support that, you need to convert the encrypted password
to plain text at runtime:

#Convert to Plaintext
$user = $login.UserName #Username is always plain text

#Convert password to plain text
$BSTR =
[System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($login.Password
)
$Password = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)

### USAGE EXAMPLES ###

#Using the credentials directly
Connect-VIServer -Server $server -Protocol https -Credential $login -
WarningAction 0 | out-null

#Using Username and Password as plain text
Connect-VIServer -Server $server -Protocol https -User icinga-read -Password
$password -WarningAction 0 | out-null
```

Creating the authfile

As per default Icinga2 service is running with the **„network service“** account, you need to use psexec to execute the script once and enter the username/password to store it encrypted in a xml file.

If the service account is „local system“:

```
`"C:\Program Files\ICINGA2\sbin\psexec`" -i -s Powershell.exe -command `"&
'C:\Program Files\ICINGA2\sbin\check_example_with_authfile.psi'`" -args `"-
allargs "that the check needs to run succesfully`"
```

If the Service account is „network service“

```
`"C:\Program Files\ICINGA2\sbin\psexec`" -i -u "nt authority\network
service" Powershell.exe -command `"& 'C:\Program
Files\ICINGA2\sbin\check_example_with_authfile.psi'`" -args `"-allargs "that
the check needs to run succesfully`"
```

Check Plugins

Here are some Powershell checks fr Icinga / Nagios. I prefer to use Powershell checks instead of NRPE/NSClient, so I'am trying fill the gap when NSClient is not used.

> These Check-Plugins are still in development and not every aspect is tested !

Check_Files - Used to check File(s)/Folder age, date, size and count
Check_filecontent - Used to check for a string in a file or all files in a folder. It also can checks the age of a file
Check_Eventwatch - Used to Check all Eventlogs for Errors. There is a Blacklist to ignore Errors if the are „by Design“.
Check_Event - Used to look for a special eventid or message in the Windows Eventlog. It is possible to check only a specific channel/application.
check_vmware_snapshot_age - need to get permission from former developer to release it
check_vmware_snapshot_size - need permission from former developer to release it