Tips when your PowerShell script runs in a Task Scheduler

Graziella Iezzi
@lunagra80

About me

Business Application Engineer

Fascinated by PowerShell

Italian expat living in Amsterdam

… Yes you can put pineapple in your pizza if you like it!

Agenda

What problem/s I tried to solve

PowerShell Clixml

PowerShell Modules

PowerShell Transcripts

How all of this comes together

Takeaways

What problem/s I tried to solve?

Creating a user deprovision script for Trello.

- Disabling the accounts of employees when they leave the company

Not everything was about the script itself, other things I considered:

- It has to run on a scheduled task (of course)
- Authentication: how do I store my token to connect to the API?
- API pagination strategy: can I reuse any of the code I have already written?
- The Unexpected!!!

How to store your secrets without Secret Management Module?

- *`Export-Clixml`* allows to export and store credentials in the format of a secure string in a secure file
- *`Import-Clixml`* allows to retrieve the credentials, ONLY by the user that created them in the first place

What to keep in mind:

- What user will run your script in the task scheduled? (i.e. SYSTEM, service account, your account)
- The credentials you store in the file not necessarily have to be the same as the user that runs the script

Recap Export-Clixml

- Create a local script to generate the secure file with *`Export-Clixml`*
- Create a scheduled task that will run with the same user as your script
- Run it once to generate the secure file the first time or anytime you need to rotate/change the credentials

API Pagination strategy: why I cared about it?

- Every API can have a different type of pagination strategy, but if you work with the same type of API the pagination strategy is the same
- The code to loop into the results pages can be the same for the same type of APIs
- You don’t need to write this loop all the time in your script but you can generalize it in a function

- A **module** containing just that function can help you developing several scripts without worrying about having the function in your script all the times

What is a PowerShell Module?

A module is a package that contains PowerShell commands, such as cmdlets, providers, functions, workflows, variables, and aliases.

- They are just another file stored somewhere with extension .psm1 that you can import in your script
- Based on what version of PowerShell you are using you can have different folders available with a pre-installed modules but you can create your own
- Useful cmdlets and variables: `$env:PSModulePath`, `Get-Module`, `Get-Command`, `Import-Module`

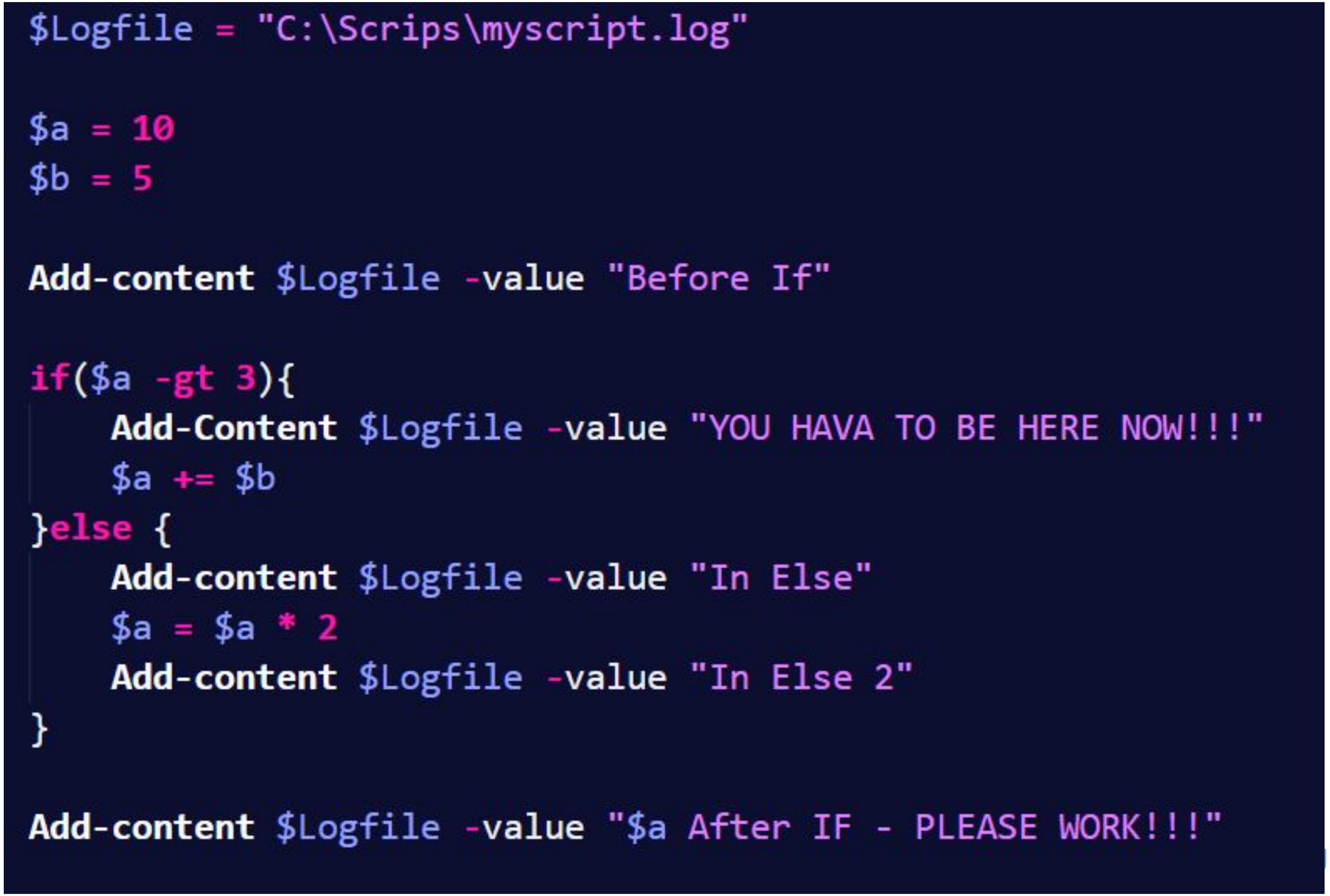
“The Unexpected” has shown its face

- The task keeps running forever
- The task is completed but it didn’t do what it was supposed to

Without a logging and error handling strategy the script just fails silently!

What can you do?

- Print some of the variables, checking if the data look correct and understanding where the script gets stuck?
- What if one of the cmdlets is failing but you are not capturing the error because you can’t see the result in the console?



```
$Logfile = "C:\Scrips\myscript.log"

$a = 10
$b = 5

Add-content $Logfile -value "Before If"

if($a -gt 3){
    Add-Content $Logfile -value "YOU HAVA TO BE HERE NOW!!!"
    $a += $b
}else {
    Add-content $Logfile -value "In Else"
    $a = $a * 2
    Add-content $Logfile -value "In Else 2"
}

Add-content $Logfile -value "$a After IF - PLEASE WORK!!!"
```

ra80

PowerShell Transcript comes to your help

`Start-Transcript` *creates a record of all or part of a PowerShell session to a text file. The transcript includes all command that the user types and all output that appears on the console.*

`Stop-Transcript` *stops a transcript that was started by the Start-Transcript cmdlet. Alternatively, you can end a session to stop a transcript.*

PowerShell Transcript comes to your help

*`Start-Transcript` **creates a record** of all or part **of a PowerShell session to a text file**. The transcript **includes** all command that the user types and **all output that appears on the console**.*

`Stop-Transcript` stops a transcript that was started by the Start-Transcript cmdlet. Alternatively, you can end a session to stop a transcript.

Takeaways

- Not all the problems you want to solve are related to the specific script you are writing
- Clixml are a secure way of store your secrets, because they can only be used in the computer it is exported in, and by the user that encrypted it
- Always refer to files with their full paths if they run in Task Scheduler
- Modules are easy to define and to use, not necessarily they have to give you a full utility of functions, you can start small
- Unexpected errors always happen (Murphy was right!) be prepared having some logs to read

References

Slide + Code: https://github.com/graiezzi/DuPSUG-2020-10

More about API and how to use them with PowerShell? Don’t be scared of calling APIs!

Why not to keep passwords in txt files? Walter Legowski - Show me all your passwords

More about Clixml: Quickly and securely storing your credentials - Jaap Brasser

More about PowerShell Modules:

- Writing a PowerShell Module
- PowerShell: Building a Module, one microstep at a time - Kevin Marquette

More about Transcript: Documenting your work with Start-Transcript - Patrick Gruenauer

Questions?