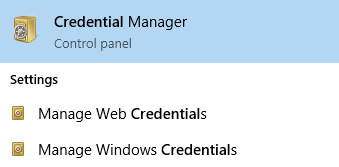
**Using Credential Manager in PowerShell**

Using PowerShell remoting can be a fantastic experience, but the number of  times I had to enter credentials to make a new pssession is getting out of hand, or to put it better a painful hand. Wouldn’t it be great if you could store the credentials somewhere safe and retrieve it when necessary? Fortunately, you can! Since a long time Windows has had the option to store credentials in a local secured database and use it when needed. This is known as the “*Credential Manager*” and can be located in the control panel.

[](https://bitsofwatercom.files.wordpress.com/2018/02/credmgr001.png)

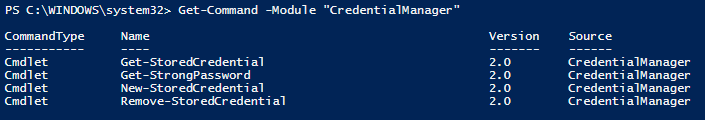
Within this tool you can store credentials for both Web Sites and network resources such as remote servers. Wouldn’t it be really cool if you could store credentials there once and retrieve them using PowerShell? Luckily, you can! Someone actually created a PowerShell Module called “*CredentialManger*” that does just what this post is about.

First, we need to install the module. It’s located in the PowerShell Gallery so you need to trust that Gallery (Use Set-PSRepository) and have NuGet installed. If you didn’t do this just yet, you’ll get additional messages, accept them to continue.

|  |  |
| --- | --- |
| 1 | Install-Module -Name "CredentialManager" |

To see what the module exposes use:

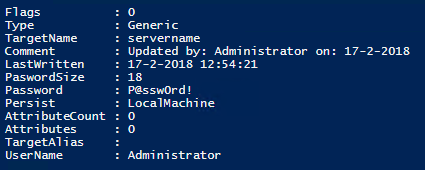
|  |  |
| --- | --- |
| 1 | Get-Command -Module "CredentialManager" |

[](https://bitsofwatercom.files.wordpress.com/2018/02/credmgr002.png)

Let’s start with creating new credentials. The CMDLET “*New-StoredCredential*” seems to do just that. I’ve constructed the following code to create a new set of credentials in the credential manager. The first part prompts for input and stores it in the $Secure variable. Next I create the new object and store it as persistant accross sessions by specifying the “*-Persist*” parameter. As it’s also a generic type of credentials I explicitly specify that also.

|  |  |
| --- | --- |
| 1  2  3  4 | $Target = "YourServerName"  $UserName = "Administrator"  $Secure = Read-host -AsSecureString  New-StoredCredential -Target $Target -UserName $UserName -SecurePassword $Secure -Persist LocalMachine -Type Generic |

The output will result in this:



Using the help system I figured out that “*Get-StoredCredential*” retrieves the objects stored in credential manager. To select the credentials we just created the “*-Target*” property needs to be specified. In this case we will be referring to the target server we just created, “servername”. Use this command to get all properties into a credentialized object:

|  |  |
| --- | --- |
| 1 | Get-StoredCredential -Target "servername" –AsCredentialObject |

Retreiving a single pair of credentials is easy, right!? Now what I am after is credentials that are stored for the use in PowerShell remoting. In my specific case to connect to a remote server I’m managing.

Eventually, I created this command-line:

|  |  |
| --- | --- |
| 1 | Enter-PSSession -ComputerName "servername" -Authentication Credssp -Credential (Get-StoredCredential -Target "servername") |

As you can see the “*Get-StoredCredential*” needs a target parameter to retrieve the credentials. That outputs a username and password (as System.Security.SecureString) that is then passed to the “*Enter-PSSession*” cmdlet.

[credmgr004](https://bitsofwatercom.files.wordpress.com/2018/02/credmgr004.png)

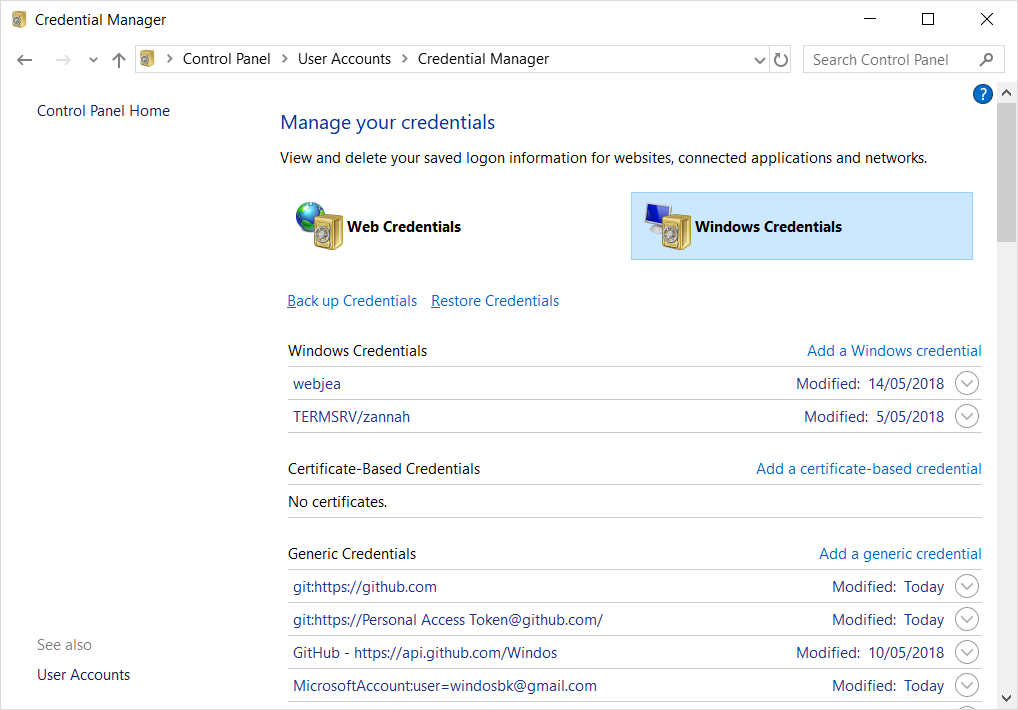
My servername in this case is “WHV” and the credentials are stored as “HyperVManager/WHV”.

Removing credentials using the same module is very straightforward. The “*Remove-StoredCredentials*” does the trick. Again pointing it to the credentials using the “-Target” parameter.

|  |  |
| --- | --- |
| 1 | Remove-StoredCredential -Target "servername" |

The Credential Manager has been a feature in Windows since Windows 7, or at least that's the first time I remember seeing it. It's used to store credentials of all sorts throughout your use of your computer. Find it by searching in the Start Menu or Control Panel.

Even if you don't save passwords, you'll probably be surprised to see how much "stuff" is in there. Check out mine:



If a user is experiencing account lockouts at work the first port of call is changing passwords on phones and other mobile devices. When that doesn't work we also go in and clear out the Credential Manager. Often there is something in there with the old password that didn't get automatically updated, and so it's trying to use the old one. A lot.

CredentialManager (minus the space) is a PowerShell module for managing credentials using this native Windows feature and it's my go to for storing and retrieving them for using in my scripts.

## **There Are Many Ways to Skin a Cat**

Very briefly, I wanted to touch on the ways to store credentials that I'm **not** using. There are a lot, and it really comes down to a conscious decision between security and usability. You can make something super secure... but then never be able to use it in a reasonable manner.

One of the most common practices I see people use when they want to keep credentials secure is to save them into a file as an encrypted string. Often this is done using Export-CliXml (and its matching Import-CliXml). This is convenient, and I'm happy with how secure it is; you can only unencrypt that file if you're the user that encrypted it on the computer on which it was encrypted.

For me, I just didn't want to deal with having files to manage.

## **Disclaimer**

Before going any further, I should note that the Credential Manager should not be considered 100% secure. Like the files saved to disk, there is nothing stopping something running as "you" seeing the passwords/tokens you have saved.

**Do not store your domain admin credentials in the Credential Manager.**

I mean, you probably shouldn't be storing them for use in your scripts using any means, but I'm not your Dad.

Personally, I've considered the risks and decided that this is "good enough" for the API keys and access tokens my scripts use.

## **Down to the Code!**

First, make sure you've installed CredentialManager, from the PowerShell Gallery.

PS C:\> Install-Module -Name 'CredentialManager'

There are three "main" functions shipped with this module, and one that's useful but I consider to be a "black sheep."

To quickly cover the black sheep, there is a function included for quickly generating random passwords.

PS C:\> Get-StrongPassword -Length 50

rvn0LOtjUsI8qy9AFCsrIzb\*#0INK:>wst@o0:&JJ3E$!zdq1.

Now, let's check out the main thrust of this module: New-StoredCredential, Get-StoredCredential, and Remove-StoredCredential.

The number one thing to be aware of when diving into this module is this: the **Target** can be considered as the unique "ID" for each credential. I originally thought of it as the "Target" of the credential, or rather "where" the credential would be used.

At one stage I tried creating two credentials for Trello, one for my API key and the other for an access token. For both, my target was set to "Trello."

This didn't work.

Targets are unique, and if you create a second credential with a duplicate target the first will be overwritten. After banging my head against my desk for a while, I ended up making my target's "Trello API Key" and "Trello Access Token". All was then right in the world.

So, enough talk. Let's create our first stored credential.

PS C:\> $Splat = @{

Target = 'King.Geek.NZ Demo'

UserName = 'Demouser'

Password = 'P@ssword'

Comment = 'This helps remind my why I created this'

Persist = 'LocalMachine'

}

New-StoredCredential @Splat

Flags : 0

Type : Generic

TargetName : King.Geek.NZ Demo

Comment : This helps remind my why I created this

LastWritten : 10/07/2018 9:27:44 PM

PaswordSize : 16

Password : P@ssword

Persist : LocalMachine

AttributeCount : 0

Attributes : 0

TargetAlias :

UserName : Demouser

The output you see is a credential type specific to this module (I believe), "PSCredentialManager.Common.Credential", and shows the Password in plain text. If you've got something logging console output... you may want to nullify that.

I already mentioned the target parameter above, and most of the other parameters are self-explanatory (we all know what a username and password is, right?)

The next big thing to be aware of is **Persist**. This defaults to Session, meaning that as soon as you logout, that credential is gone. Chances are you're using this for ongoing scheduled jobs, so make sure you set it to LocalMachine instead.

## **Getting Credentials**

Now that you've created, and stored, a credential you'll want to actually use it. To do that we'll be using Get-StoredCredential. By default, this function returns credentials in as PSCredential objects (primed and ready to be passed through to other functions and cmdlets.)

If run without any arguments, this function will try to return all stored credentials as PSCredential objects. This is result in a **lot** of warnings as there will be some in there with no passwords that cannot be converted to the desired type.

To get around this you can specify that they are returned at the custom type we saw when creating our first credential earlier.

PS C:\ > Get-StoredCredential -AsCredentialObject

This also allows some form of discoverability, as you can pipe this through and filter for specific username's or targets where you're not 100% sure of what you're looking for.

Do remember that any passwords in this format will be in plain text.  
Ultimately, however, you're going to want to specify a target (which is unique) and receive back a singular PSCredential object.

PS C:\> Get-StoredCredential -Target 'King.Geek.NZ Demo'

UserName Password

-------- --------

Demouser System.Security.SecureString

## **Closing the Loop**

The final command is a simple affair, but it deserve a mention nonetheless. If you find yourself wanting to clean up your credentials, go ahead and remove them.

PS C:\ > Remove-StoredCredential -Target 'King.Geek.NZ Demo'

## **Actual Use**

Personally, I use the credential manager anytime I need to pass-through some form of username and password to a function inside a headless script. An example of this was some work I was doing recently that involved Trello. I didn't want to leave my API key and the like in plain text in the script, so instead I stored them as passwords.

At the top of the script you can see me pulling them out and storing them in variables for use all throughout the rest of the function.

$Token = @{

Token = (Get-StoredCredential -Target TrelloToken).GetNetworkCredential().Password

AccessKey = (Get-StoredCredential -Target TrelloKey).GetNetworkCredential().Password

}

## **Wrap Up**

Passwords in general are a hotly debated topic. On top of that there are many rungs on the security ladder. Weighing up the risks, this works for me in my environment but it may be unsuitable in yours.

How do you deal with credentials in your scripts?

My ultimate wish is to be able to store them in my LastPass password manager, but I need an easy to use (and secure) PowerShell interface first.