

**Win500 - Lab 5 – Credentials**

In this lab you will learn the following:

* How to create a PSCredential
* How to convert PSCredential to SecureString for scripting
* How to modify the Trusted Hosts file

**Introduction**

Up to this point you have been using transparent authentication to access remote servers. That is, you have been using an Administrator account to run your remote sessions and if you do not specify a username/password, PowerShell attempts to authenticate you with the username you logged on with. For our purposes that is OK, but it is not always practical.

There are times when we, as administrators, need to have others perform some task, requiring Admin rights. We obviously do not want to give them the Admin password or elevate their accounts to Admin levels. In addition, we may want to restrict what they can and cannot do.

a) Therefore, we need some mechanism where we can provide users with the ability to execute Admin level scripts, without changing their own status or providing them with Admin password(s).

**Note:** when we get into AD Group Policy, you will see how to set accounts that have various levels of PowerShell permissions.

1. Create a basic user account in Active Directory
2. Log into your workstation Win7-WS using this account
3. Open PowerShell (Run as Administrator)

What happened? \_\_\_\_\_\_\_**It prompted the box to ask you username and password**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. Open PowerShell
2. Run the following cmdlet

**Invoke-command –computername <ADcomputer> -scriptblock {get-process}**

What happened? \_\_\_\_\_\_\_\_\_\_\_**It showed error as you don’t have permission**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Now Run This:

**Invoke-command –computername <ADcomputer> -scriptblock {get-process} -credential <domain/username>**

What happened? \_\_\_\_\_\_\_\_ **It prompted the box to ask you username** \_\_\_\_\_\_\_\_

You can see that a user with no Administrative rights can execute commands and scripts. But, in order for the User to run the script, he/she has to enter the Admin password. That’s not good. Let’s correct this.

**1 Get-Credential**

(get-help Get-credential –full) # to read a complete description of the command.

The main purpose of this command is to pop-up a box asking for a username and password.

Type in the command: **$a =** **Get-Credential -message <fill in message>** (fill in the username and password boxes, click OK)

**Write-host** **$a**

**Question:**

What output do you see for the password? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**The object $a is now a secure PSCredential object. It is encrypted using AES. This object as 3 characteristics**

* **It is deleted from active memory after it is used**
* **Only the Windows Security Manager built into the OS can use the credential**
* **The credential is never written to disk**

**\*\*\*\*Security Alert\*\*\*\*** It would be wise to remember that just having a secure password and not revealing it is not enough. You also have to secure your scripts. If someone wants to get your password, all they have to do is modify one of your scripts or provide you with one of their own. Think about this. How often would you look at the code, within a working script that you created, to see if it has changed? What about scripts from someone you trust?

So, here is some code just to illustrate the point.

Type it to see what happens.

**Clear-Host  
$ClearPassword = Get-Credential -message “Authentication is necessary”  
 #Note: Makeup username/password do not use your Seneca account information  
$ClearPassword.GetNetworkCredential().username  
$ClearPassword.GetNetworkCredential().password**

Question:

What is on the last line of output? \_\_\_\_\_\_\_\_\_**The plain text password\_\_\_\_\_\_\_\_\_\_\_\_**

The problem here is the GetNetworkCredential() method converts the PSCredential object to a System.Net.NetworkCredential object which converts the hash value to a clear text string. Hash values must be kept secure from someone who has access to your credential or file where it is stored. This is critically important.

**2 Encrypting Passwords**

**1. Creating and Testing a SecureString**

Type the following. When you need to get a password from the user you can use the –AsSecureString parameter of Read-Host.

**$secure = read-host “Enter a password” -assecurestring**

**$secure**

What is the output? \_\_\_\_\_\_\_\_\_ **System.Security.SecureString** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**$encrypted = convertfrom-securestring -securestring $secure**

**$encrypted**

What is the output? \_\_\_\_\_\_\_\_\_\_\_**Hash value\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Now type the following code to convert the secure string back to a Pscredential object

**$secure2 = convertto-securestring -string $encrypted**

**$secure3 = new-object -typename System.Management.Automation.PSCredential `  
 -argumentlist $secure2**

**$secure3.getype()**

What is the output? \_\_\_\_\_\_**PSCredential SystemObject**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

While keeping a password secure is an important security feature, you may sometimes want to store a password (or other sensitive information) on disk so that other accounts have access to it anyway. This is often the case with scripts run by service accounts or scripts designed to be transferred between computers. The **ConvertFrom-SecureString** and **ConvertTo-SecureString** cmdlets support this by letting you to specify an encryption key.

**NOTE:** When used with a hard-coded encryption key, this technique no longer acts as a security measure. If a user has access to the content of your automated script, they have access to the encryption key. If the user has access to the encryption key, they have access to the data you were trying to protect.

**3 Securely Store Credentials on Disk**

**Problem**

Your script performs an operation that requires credentials, but you don't want to type in the password each time you need to run the script; or, you need to allow another person to execute the script without providing them your password.

**Solution**

To securely store the credential's password to disk, so that your script can load it automatically, use the ***ConvertFrom-SecureString*** and ***ConvertTo-SecureString*** cmdlets.

**Step 1: Save the credential's password to disk – Do this on your SRV1-AD**

The first step for storing a password on disk is usually a manual one. Given a credential that you've stored in the **$credential** variable, you can safely export its password to My**Credential\_L5*.ps1.credential*** using the following command. For example, if the script that requires the credential is **MyCredential\_L5.ps1**, then the credential file would be called **MyCredential\_L5.ps1.credential** . Note: The path is where PowerShell believes the file should be located. This may be different on your system but **C :\Users\Administrator\Documents\WindowsPowerShell\** should be the default.

**$credential = Get-Credential – message “Authenticate with your domain admin account”**

**$credPath = Join-Path (Split-Path $profile) MyCredential\_L5*.ps1.credential***

**$credential.Password | ConvertFrom-SecureString | Set-Content $credPath**

In the WindowsPowerShell directory open the MyCredential\_L5.ps1.credential file in Notepad.

What is the files output? \_\_\_\_\_\_\_\_\_\_\_\_\_**Hash value**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Exercises**

**Exercise 1:**

**Step 2: Recreate the Credential from the Password Stored on Disk**

Open your SRV1-AD PowerShell ISE

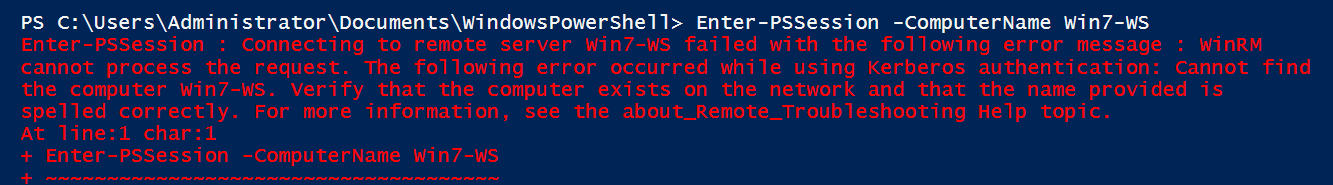
1. Now we will use our credential in Step 1 in a script that will remove Win7-WS from the domain.
2. In the script that you want to run automatically, add the following commands (beginning of Script). Name your script **RemoveComputer\_Lab5\_Ex1.ps1**

**$credPath = Join-Path (Split-Path $profile) MyCredential\_L5.ps1.credential**

**$password = Get-Content $credPath | ConvertTo-SecureString**

**$credential = New-Object System.Management.Automation.PsCredential “droy.loc\administrator”,$password**

1. Use **get-help Remove-Computer -parameter \*** to research and add the correct parameters.
2. Use the invoke-command to connect to Win7-WS and execute the command, removing Win7-WS from the domain, and rebooting the system. Don’t forget to pass your domain admin credentials to the script.
3. After the script has executed, try and connect to Win7-WS using Enter-PSSession. You should e receive an error message like the following:



Creating a PSSession is not possible if a computer is not part of the domain because Windows can’t use AD to resolve the names. (Note: AD will still show Win7-WS as part of the domain until the database is synchronized. **We are not going to reboot the DC to synchronize the change.**

**There are 2 ways to fix this problem. Install CREDSSP on each machine or use the trusted hosts file. We will do the latter.**

1. Open the TrustedHosts file on SRV1-AD and add Win7-WS as the value. Type:

**Set-item wsman:\localhost\client\TrustedHosts -value WIN7-WS**

1. Open the trusted hosts file on Win7-WS and add **SRV1-AD** as the value
2. Now on the Win7-WS VM open a session to SRV1-AD. Type:

**Get-item wsman:\localhost\client\trustedhosts  
Enter-PSsession -computername SRV1-AD**

You should be successful in connecting to SRV1-AD because you have bypassed authentication by using the trusted hosts file. Take a screen shot of your output. Name the file **LearnName\_TrustedHosts\_Lab5\_Ex1.jpeg**

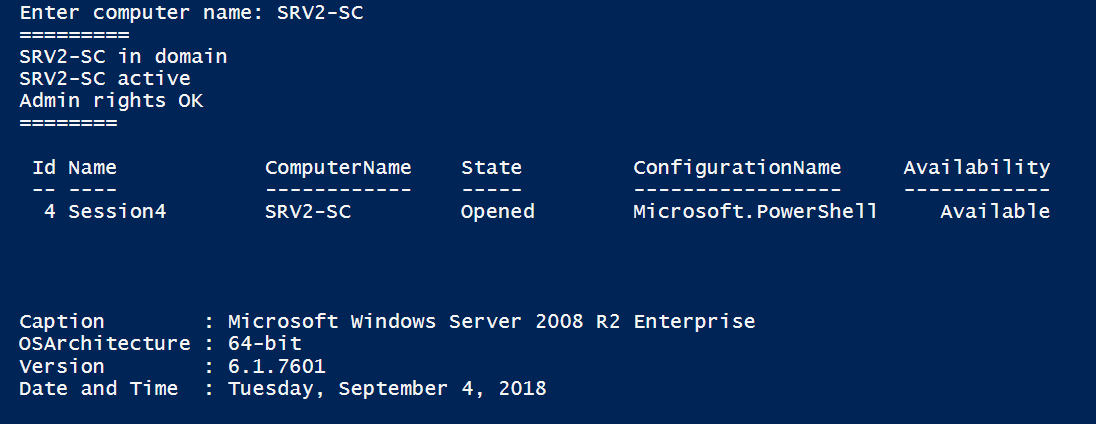
**Using the trusted hosts file is a security risk. If you don’t have strong physical and network security, you have no way of knowing if Win7-WS has been compromised. Suppose, the system was infected with a malicious virus, or someone installed a “keylogger” program, you just logged in with your domain admin account to your domain controller. Do you see the problem?**

1. On Win7-WS type:  
   **Add-computer -domainname <your domain> -restart**Make sure that Win7-WS is again part of your domain.

This exercise is difficult; to help you I used an If..Elseif..Else clause for the conditional statement which provides all the logic. Here are the cmdlets I used inside my function, in order of use:  
  
Read-Host  
Get-ADComputer  
Get-Content  
Write-Warning  
Test-Connection  
Start-Sleep -s 2  
New-PSSession  
Get-Credential  
Get-CIMInstance  
Select-Object  
Format-List  
Get-Date  
  
Take your time and write comments as you go to organize your thinking

**Exercise 2:**

This will be your first attempt at writing a production level script. Write **a function** which will do the following. (Save the script as **Learnname\_Lab5\_ex2.ps1**

1. Ask the user to input a computername  
   2. Get a listing of all computers in AD,
2. Test to see that the user’s computer name is in the list. If the computer is NOT named in the domain listing display a warning message and exit
3. Test to see that the user’s computer name is up and running. If is not active, display a warning message and exit.
4. Test to see that the user has Admin rights. If the user is not logged in as an administrator, display a warning message and exit. (to test your if statement you will need a user or guest account. The Guest account is disabled by default)
5. The function should use 5 parameters (variables):
   1. The name of the computer supplied by the user,
   2. One for the CIM class (you can use either Win32\_operatingsystem, Win32\_Bios, Win32\_Product, or Win32\_UserAccount),
   3. And any 3 properties from one of the above classes.
6. Create a Credential object and pass it to invoke-command
7. Use your variables in the cmdlets and pass them as arguments to the cmdlets parameters (any 3 of your choosing)
8. The function should display the properties in List format and add a custom property which displays the date the script ran in long format.
9. Call your function using invoke-command with the argumentlist parameter. Also pass the $credential object you created. Your output should similar to the screenshot. Name your file **LearnName\_Lab5\_ex2.jpeg**

Grading

* **LearnName\_TrustedHosts\_Lab5\_Ex1.jpeg**
* **RemoveComputer\_Lab5\_Ex1.ps1**
* **Learnname\_Lab5\_ex2.ps1**
* **LearnName\_Lab5\_ex2.jpeg**
* **Lab5\_Credentials.docx**

Zip the files together and submit using the link under Graded Work on My Seneca