How to use PsExec

In this article, you will learn how to use PsExec, a great command line utility from Microsoft's Sysinternals PsTools suite, which allows system admins to run programs on one or more remote computers while redirecting the program's output to the local computer.

Why use PsExec

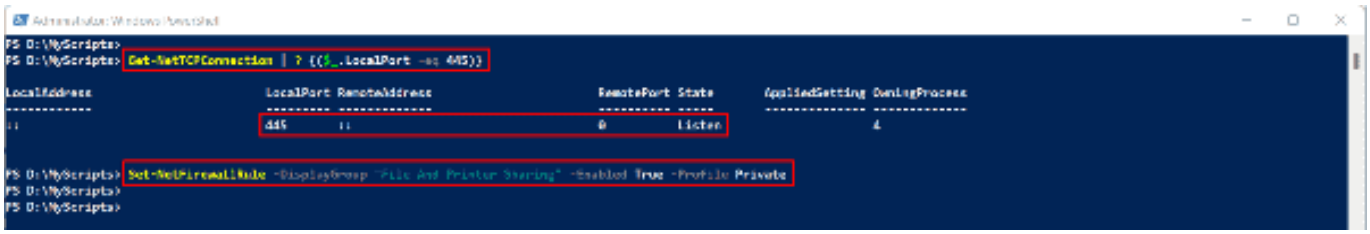
Although PsExec's ability to run processes on remote computers might not be a big deal for new admins (since PowerShell remoting is already available), it is still popular among long-time admins. So instead of considering PsExec to be an old, outdated tool, it is very powerful if you use it along with PowerShell. In my next post, I will tell you how you can use PsExec to complement PowerShell and achieve great results that are hard to achieve with PowerShell alone.

Prerequisites for PsExec

Your system and the remote systems in your network must meet the following prerequisites before you can start using PsExec:

File and printer sharing must be enabled on both local and remote computers (TCP port 445 must be open on remote computers). It could be a potential security risk, so make sure you enable this port for the Private profile in the Windows firewall.

```
Set-NetFirewallRule -DisplayGroup “File And Printer Sharing” -Enabled True -Profile Private
```



Enable File and Printer Sharing firewall rule using PowerShell

You could also use the *Invoke-Command* cmdlet in PowerShell to set this firewall rule in multiple remote computers at once.

The default admin$ share must be enabled.

View default admin shares using PowerShell

Security considerations

If a remote computer is in the domain, the members of the domain admins group will be able to use PsExec without a problem.

An earlier version of PsExec wasn't that safe since it was used to transmit passwords and commands in clear text over the network. However, starting with version 2.1, it encrypts the alternate credentials and commands in transit. Please make sure you are using the latest version of PSExec (2.34 is the latest version at the time of writing).

PsExec is kind of like a double-edged sword. In the right hands, it can be a great tool, but in the wrong hands, it can be a disaster. Due to its abilities, it is often used by hackers in combination with other tools, such as the Metasploit framework and Mimikatz, to carry out malicious attacks. This is the reason that major antivirus products have started flagging PsExec as malware. In a nutshell, PsExec is a safe sysadmin tool in itself, but in the end, it totally depends on how the person is using it.

Since PsExec uses the file and printer sharing service, which is vulnerable to various viruses and ransomware, please make sure you properly secure and restrict this service and its associated ports to the local network only; never expose it to a public network.

Downloading and installing PsExec

As mentioned above, PsExec is part of the PsTools suite. To use PsExec utility, we need to download the PsTools suite from the Sysinternals website.

There is no installer in the PsTools zip file. All you need to do is extract the files from the zip archive and start using them. The installation of PsTools depends on how and where you want to extract the files. Some people prefer directly extracting them inside the **%systemroot%\System32** directory so that the executables can run directly without having to specify the full path.

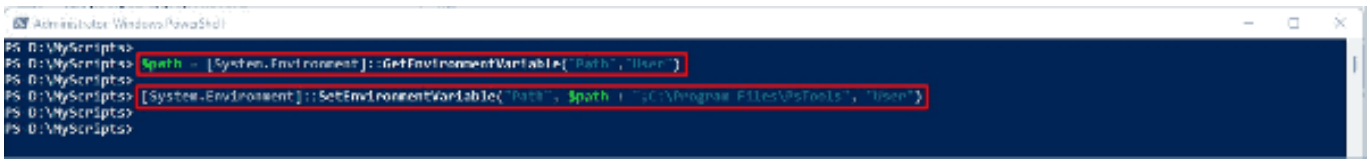
I like extracting these files into a separate directory inside **%systemdrive%\Program Files** alongside other programs and then setting up the **Path** environment variable. You could choose either way; it's just a matter of your own preference. Let's extract the zip archive using the following PowerShell command:

```
Expand-Archive -Path "$env:USERPROFILE\downloads\PsTools.zip" -DestinationPath "C:\Program Files\PsTools" -Force
```

The PsTools are now in the right place. You just need to modify the **Path** environment variable and specify the full path of your destination directory. Again, there are two **path** environment variables—**System** and **User**. It depends on how you want to use the PsExec tool. If you want PsExec (and other tools) to be available for all users in the current system, you can add it to the **System Path** variable. If you want PsExec to be available only to your current user profile, you could add it to the **User Path** variable. To do this, you can use one of the following commands:

For the User variable:

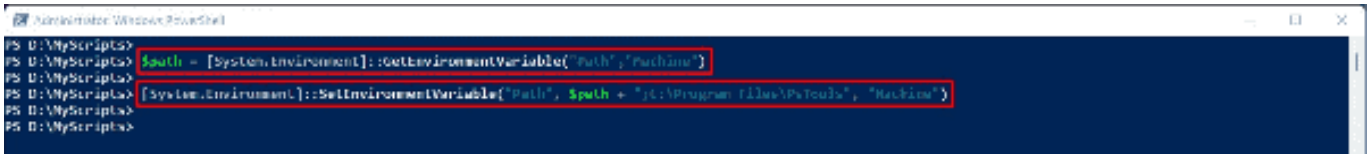
```
$path = [System.Environment]::GetEnvironmentVariable("Path","User")
[System.Environment]::SetEnvironmentVariable("Path", $path + ";C:\Program Files\PsTools", "User")
```



Add PsTools to the User Path environment variable

For the System variable

```
$path = [System.Environment]::GetEnvironmentVariable("Path","Machine")
[System.Environment]::SetEnvironmentVariable("Path", $path + ";C:\Program Files\PsTools", "Machine")
```



Add PsTools to the System Path environment variable

No matter which environment variable you set, be sure to specify the full path of the directory to which you extracted the **PsTools** executables. Also note that the semicolon (;) before the directory path is intentionally added to separate the new value from the old values.

Similarly, you can also view the **System Path** environment variable using the following command:

[System.Environment]::GetEnvironmentVariable("Path","Machine") -Split ";"

At this point, PsExec (and other utilities in PsTools) are ready for use in your system.

Use PsExec on a single remote computer

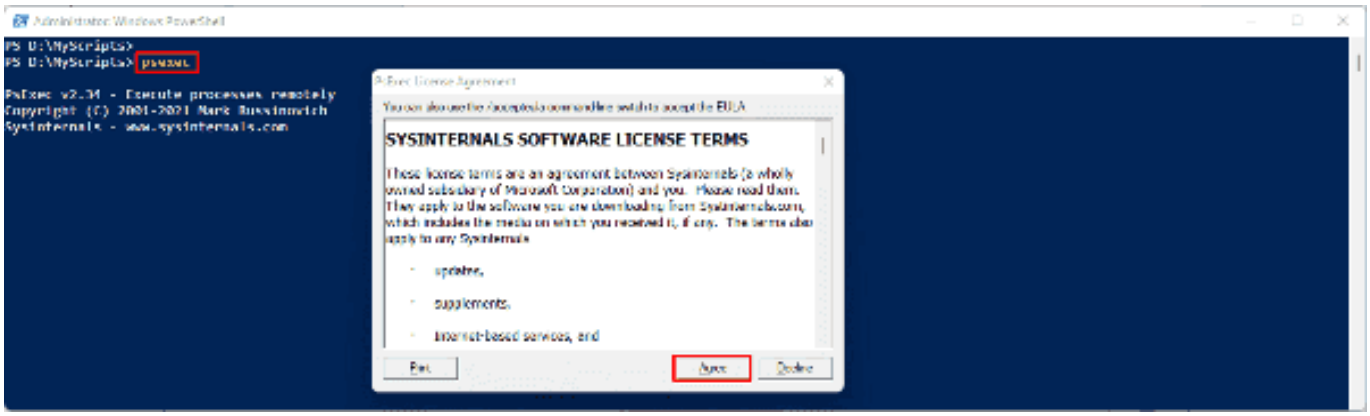
To start using PsExec, just close the existing PowerShell console and launch a new one. If you want to use it in a command prompt, you can launch a command prompt. Whichever you choose, just make sure you launch an elevated session since PsExec requires administrator privileges to run programs on remote computers. Non-admins will also be able to use PsExec locally, but there is not much you can do without admin privilege.

PsExec syntax

The syntax of PsExec is fairly simple, as shown below:

psexec [\\computer[,computer2[,...] | @file]][-u user [-p password][-n s][-r servicename][-h][-l][-s|-e][-x][-i [session]][-c executable [-f|-v]][-w directory][-d][-<priority>][-a n,n,...] cmd [arguments]

If the syntax sounds complicated to you, trust me, it will become clear as we use it. When you run PsExec for the first time, you will see an end-user license agreement (EULA), as shown in the following screenshot:



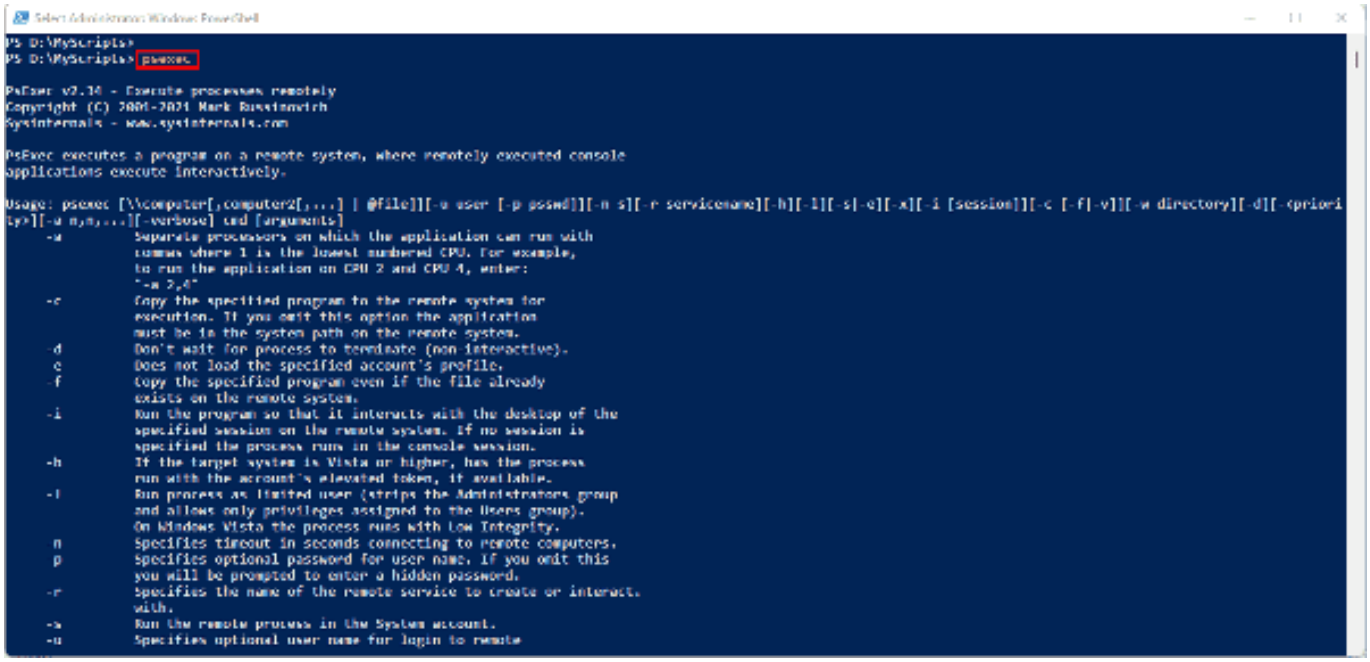
Displaying the PsExec EULA

Click the **Agree** button to accept the EULA. To suppress this EULA, you could use the */accepteula* switch as shown below:

psexec /accepteula

Getting help

Help documentation is the most important thing for getting started with any tool. If you run the **psexec** command without any switch, you will see detailed help, including its usage syntax and all the supported switches. The following screenshot shows what the help section looks like.



Displaying the PsExec help

I will not discuss every switch in detail, but will try to cover the most important ones. While researching and experimenting with PsExec, you can always refer to the built-in help.

PsExec command error

If you do not see this help page but instead see an error that says *'psexec' is not recognized as an internal or external command, operable program or batch file*, it means you did something wrong while setting up the **Path** environment variable.

'psexec' is not recognized as an internal or external command operable program or batch file

To fix this error, make sure you set the **Path** environment variable correctly.

Exit and error codes

When you use PsExec to run a program (or a process), it might display an exit code for that program and returns an error code. The error code returned is not generated by PsExec; it is generated by the original program that you run using PsExec. So, if you get any error while running a program, you should consult the program's documentation to determine what that error code means. An **exit code of 0** typically indicates successful execution.

Impersonation and alternate credentials

User impersonation occurs when a program runs in the security context of a user other than the currently logged-in user. When you run a program (or process) with PsExec, the program is executed on the remote computer impersonating the user account that you're using on your local computer to run PsExec. In other words, the process that you start on a remote computer runs by impersonating your user account on a local system. Impersonation has certain limitations in Windows. The main limitation is that the remote process cannot access any resources on the network. To get around this problem, PsExec allows you to provide alternate credentials using the -u and -p switches.

To run a process on a remote computer using alternate credentials, you could use PsExec, as shown below:

psexec \\remote_computer -u domain\admin -p password -i process_name

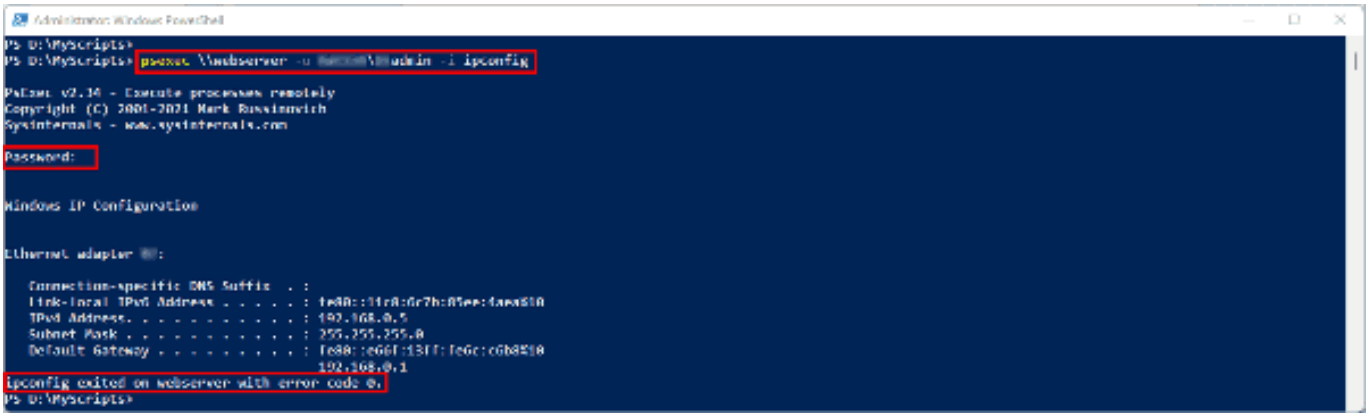
where:

- The \\remote_computer should be replaced with the name or IP address of the remote computer. You could specify multiple computers separated by a comma (,).
- The -u switch specifies the alternate user name. The -u should be followed by the user name, which can be either a local admin user on the remote computer or a domain admin.
- The -p switch is used to specify the password of the alternate user account you specified in the -u switch. Note that -p requires you to type the password in clear text, so even though PsExec will encrypt these credentials in transit, someone could still see the password as you type it in clear text. Therefore, it is a good idea to skip the -p switch and use the -u switch alone. PsExec will automatically prompt you to type the password when you press Enter, which is a safer way.
- The -i switch specifies the interactive mode. This switch was not needed until PsExec version 2.20. You should use this switch with version 2.30 and above; otherwise, you will get an error.
- The process_name should be replaced with the command or process you want to run on the remote computer.

Example

psexec \\webserver -u domain\admin -i ipconfig

This command runs the **ipconfig** command on a remote computer named **webserver** using the alternate credentials of the domain admin.

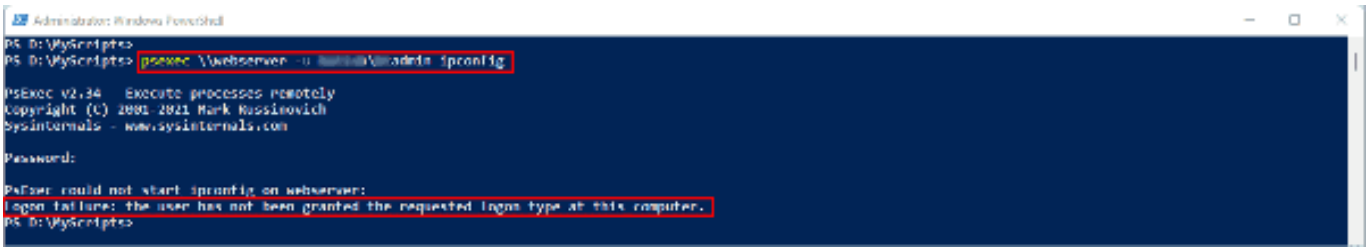


Running a command on a remote computer using alternate credentials with PsExec

Running a command on a remote computer using alternate credentials with PsExec

The screenshot shows that PsExec prompted for the password on the fly, since I did not use the -p switch. **Error code 0** indicates that the **ipconfig** command exited with success.

If you get an error that says "Logon failure: the user has not been granted the requested logon type at this computer," as shown in the following screenshot, make sure to use the -i switch with the command. It is needed with PsExec version 2.30 and above.



Logon failure the user has not been granted the requested logon type at this computer

Use PsExec on multiple remote computers

One cool feature of PsExec is that you can specify multiple computers on which to run a program (or process) at the same time. There are several ways to specify multiple remote computers.

Using comma-separated computer names

PsExec allows you to specify a comma-separated list of computers in a domain or workgroup.

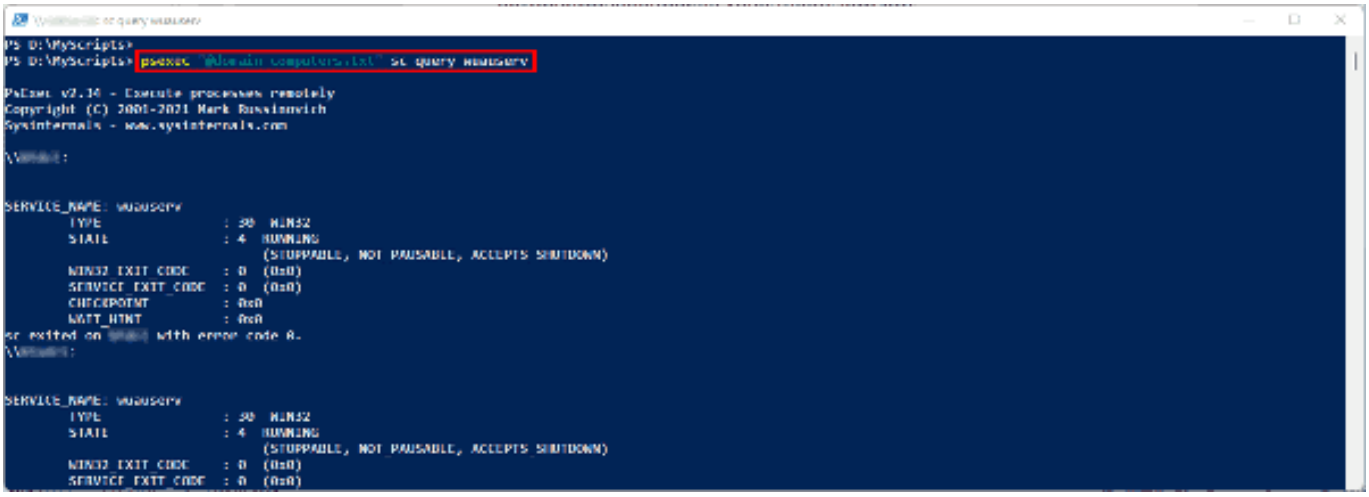
psexec \\webserver01,webserver02,fileserver01,fileserver02 sc start wuauserv

The above command will run **sc start wuauserv** to start the **Windows update** service on multiple remote computers specified in a comma-separated string. Remember not to use a space between computer names and commas.

Using a file to specify multiple computer names

You can also use a text file containing multiple computer names (one per line) using the @file format to specify the file name.

psexec @domain_computers.txt sc query wuauserv



Running a command on multiple remote computers using files with PsExec

The above command runs the **sc query wuauserv** command to view the status of the **Windows update** service on all the computers mentioned in the specified file (@domain_computers.txt in our example). Make sure there is one computer name per line in the file; otherwise, you will get an error.

Note that I am using PowerShell to run PsExec, so I had to enclose the file name in single or double quotes (for example, "@domain_computers.txt"). This is necessary, since PowerShell, by default, interprets the @ symbol as an array subexpression operator. If you're running PsExec at the normal command prompt (cmd.exe), there is no need to enclose the @filename in quotes.

Using a wildcard to run the process on all computers

PsExec also allows you to use the wildcard character (*) to specify all the computers in the current domain or workgroup.

psexec * getmac

This will run the **getmac** command on all the remote computers that are in the same workgroup or domain.

Using * causes PsExec to enumerate all the computers in the current domain or workgroup by running the **net view /all** command under the hood. Since the **net view** command depends on NetBIOS, if you run this command in your domain or workgroup environment, it will most likely fail with an error stating: "A system error has occurred: 6118".

PsExec: A system error has occurred 6118

This error essentially means that the firewall (either the Windows firewall or the antivirus firewall) is blocking the SMB connections, or the network discovery is turned off on the remote computer(s).

This is an outdated and inefficient way of enumerating network computers. Later in this guide, I will tell you how you can utilize PowerShell to build a file containing multiple (or all) domain computers so that you can easily run processes on those computers with the help of PsExec, and you won't have to use * to enumerate computers.

Run a process as the LocalSystem account

The LocalSystem account is a predefined account with extensive privileges on the local computer. It is created by the operating system during installation. The security token of the LocalSystem account includes the security identifiers "NT AUTHORITY\System" and "BUILTIN\Administrators". These accounts have unrestricted access to most of the system objects. The advantage of running a process or program under the LocalSystem account is that the process will have unrestricted access to all system resources.

One of the most powerful features of PsExec is its ability to launch a process (or program) with LocalSystem account privileges. This feature is most often exploited by hackers and other malicious users. You can imagine what malicious users can do on your system if they succeed in running a malicious process with the LocalSystem account privilege.

To launch a process as a LocalSystem account, you can use the -s switch with PsExec, as shown below:

psexec \\webserver -s cmd /c dir "c:\System Volume Information"

The -s switch causes PsExec to launch the **cmd.exe** process with the LocalSystem account on the webserver. Note that the /c switch used here belongs to **cmd.exe** itself, which is useful for running a command on the fly and then terminating it. It is particularly useful when you want to run a single command without having to launch an interactive console session of **cmd.exe** on a remote computer. Don't confuse this with the -c switch of PsExec, which is useful for copying a program to a remote computer when it doesn't exist in the default path. We will discuss the -c switch in greater detail in a separate section below.

This command displays the contents of the System Volume Information directory on the remote webserver, which is not accessible by regular user accounts.

Run a process on a remote computer under the LocalSystem account with PsExec

The following image shows how the -s switch makes a difference in the PsExec command:

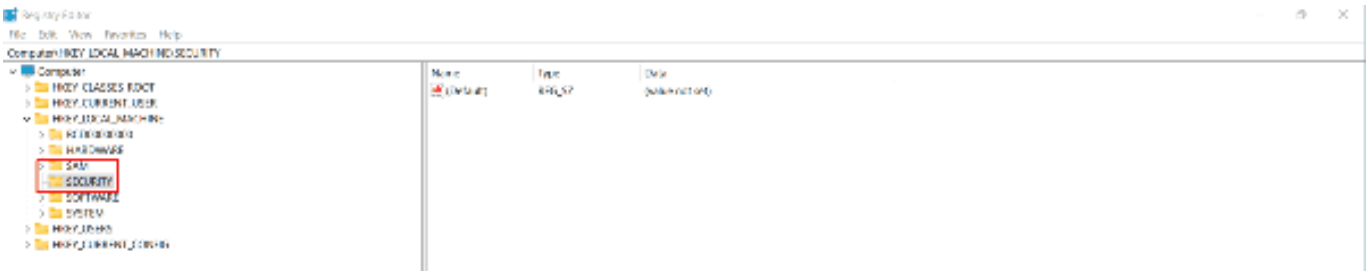
Local System Account vs. Impersonate User

This image shows that when you run a process with the -s switch, it is launched with the "NT AUTHORITY\System" privilege. When you run a process without specifying any alternate credentials, your currently logged-in user is impersonated.

Interactive mode

Another important feature of PsExec is its ability to launch a process in an interactive mode using the -i switch. It is most commonly used along with the -s switch to launch a GUI-based program (or process) on a local computer with the "NT AUTHORITY\System" privilege.

For example, when you normally launch the Windows registry editing tool (regedit.exe*)* and try to open the HKEY_LOCAL_MACHINE\SYSTEM and HKEY_LOCAL_MACHINE\SAM subkeys, you will not see anything there because these subkeys are only accessible to the SYSTEM account.



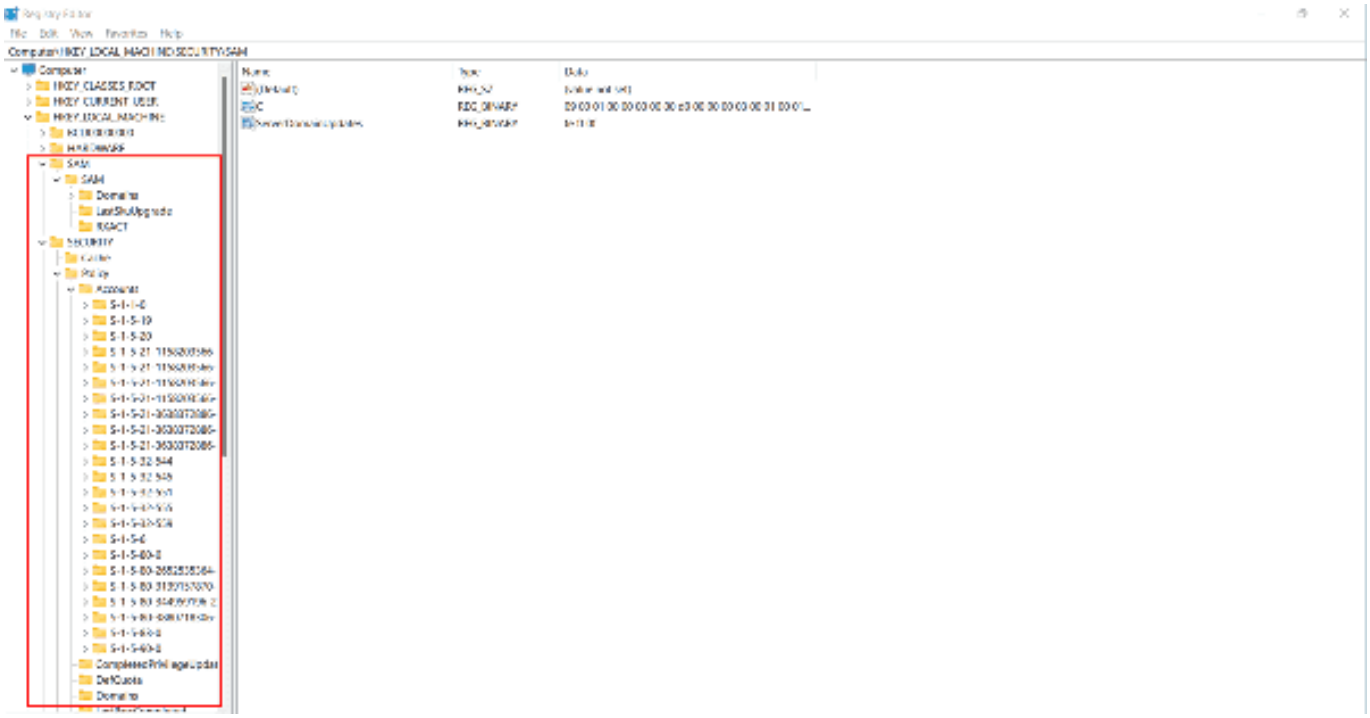
Direct access to SAM and SECURITY registry subkeys

However, if you launch regedit.exe with PsExec using the -s and -i switches, as shown in the following command, you will be able to see everything in these subkeys.

psexec -s -i regedit

Interactively launch regedit under the System account with PsExec

Take a look at the following screenshot showing all the subkeys under SYSTEM and SAM. This became possible since PsExec launched the *regedit.exe* with "*NT AUTHORITY\System*" privilege.



Accessing SAM and SECURITY registry subkeys using System Account

When you launch the process this way, PsExec waits for the program to finish running before the exit code is returned. If you do not want to wait, you can use the -d switch, as shown below:

psexec -s -i -d regedit

This command returns the exit code, and you will get the command console control back immediately.

You can also use this technique to launch any GUI-based program interactively on a remote computer, as shown in the following command:

psexec \\webserver -s -i -d powershell

This command launches PowerShell in interactive mode on a remote webserver with the SYSTEM account privilege and returns the process ID.

Launch PowerShell in interactive mode on a remote computer using PsExec

PsExec even allows you to control the priority of the process that you're running with the help of the -low, -belownormal, -abovenormal, -high, or -realtime keywords. The following command launches the PowerShell process on a remote webserver with *a* priority of high.

psexec \\webserver -i -d -high powershell

Launch a process in interactive mode and high priority using PsExec

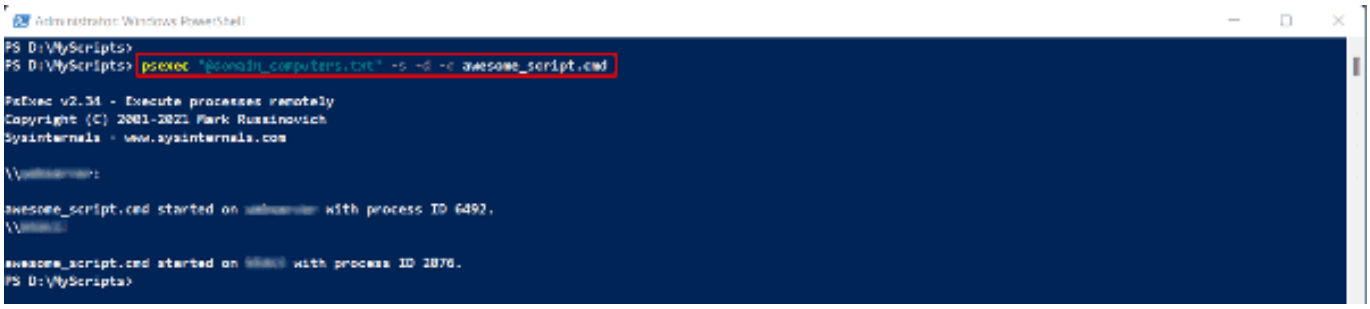
On Windows Vista, you can also use -background to run the process at low memory and I/O priority.

Copy a program to a remote computer

The most unique feature of PsExec is its ability to temporarily copy a program (or process) and then execute it on one or more remote computers. This is particularly useful when you have a program on your local computer, and you want to run it on the remote computer(s). All you need to do is use PsExec with the -c switch and specify the name of the program on the local computer. Have a look at the following command:

psexec @domain_computers.txt -s -d -c awesome_script.cmd

This command copies the awesome_script.cmd script and executes it on all the computers mentioned in the domain_computers.txt file using the SYSTEM account.



Copy a program to a remote computer using PsExec

The -c switch allows PsExec to temporarily copy the program from the local computer to the admin$ share (which is mapped to the Windows directory) on a remote computer, execute it, and then delete it as soon as the program finishes executing.

If you get an error that says, "The specified application is not on the path," as shown in the following screenshot, make sure the program or script you're trying to copy using the -c switch exists in your current working directory. In my case, the awesome_script.cmd must exist in the D:\MyScripts directory.

PsExec error: The specified application is not on the path

If the program or script already exists in the remote computer, but you still want to copy it anyway, use the -f switch. You can also use the -v switch to copy only the program if its version on the local computer is newer than on the remote computer.

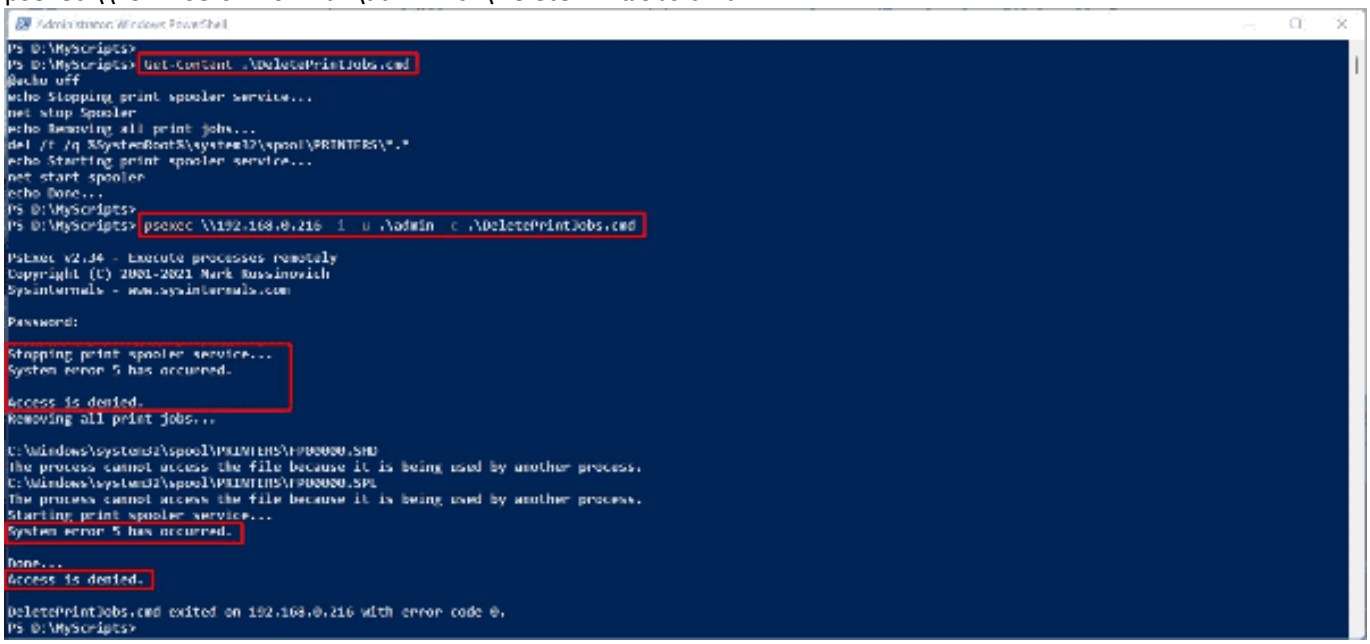
Respecting UAC and Windows security

Windows Vista and above have an access control feature known as User Account Control (UAC), which allows programs and processes to always run in the security context of a non-administrator account unless the administrator specifically authorizes that program to run with elevated access to the system. This feature prevents malware and other potentially dangerous programs from making changes to your computer.

PsExec provides the -h switch, which allows the process to run with the account's elevated token if the target system supports it. With the help of this switch, admins can launch processes without having to worry about being blocked by UAC on a remote computer. Let's understand this with an example:

I have a batch script on my workstation that stops the print spooler service, deletes all the files from the **%SystemRoot%\system32\spool\printers** directory, and then starts the print spooler service again. This is just a simple batch script; there's nothing fancy here. When I get a call from any user complaining about stuck print jobs, all I need to do is run this script remotely on their PC with the help of PsExec. The following screenshot shows that when I try to run the script without the -h switch, it fails with an error stating: "Access is denied:"

psexec \\192.168.0.216 -i -u .\admin -c .\DeletePrintJobs.cmd

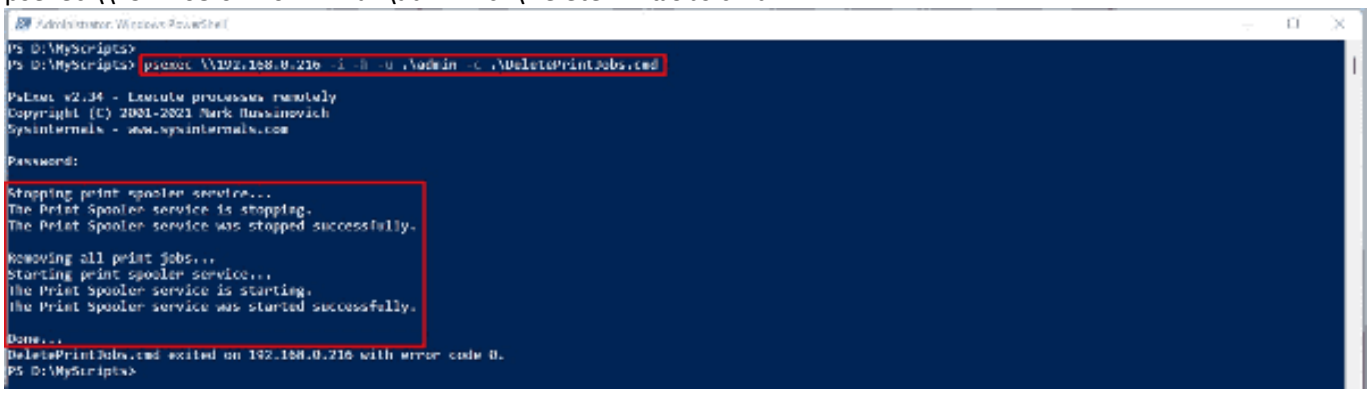


Running a script on a remote computer without an elevated security token

The script failed because UAC on the remote computer is restricting it from making changes to the computer, even though the alternate user specified is a member of the administrators group. This is the main purpose of UAC.

Now, let's see what happens when we add the -h switch to command:

psexec \\192.168.0.216 -i -h -u .\admin -c .\DeletePrintJobs.cmd



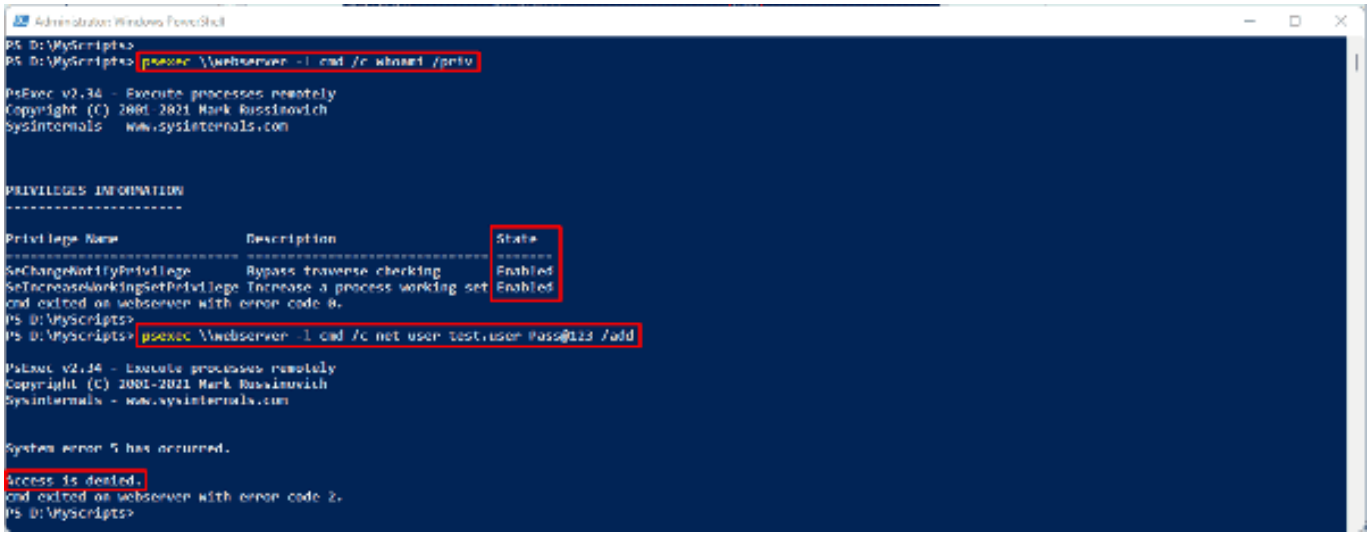
Running a script on a remote computer with an elevated security token

You can see that the script was executed successfully. The -h switch allowed the script to run with the user's elevated security token on the remote computer.

In the same way, if you think your current user account has too much privilege but you want to run the process on a remote computer with limited privileges, you can do so with the help of the -l switch. This allows PsExec to run the process as a limited user by stripping off the administrators group and allowing the privileges assigned to the Users group only. The following command shows how to run cmd.exe with the least privilege on a remote webserver:

psexec \\webserver -l cmd /c whoami /priv

This command launches **cmd** on the remote webserver with the least privilege. So, even though my currently logged-in user is a member of the domain admins AD group, it will still have restricted permissions on the remote computer within that process. Have a look at the following screenshot:



Running a process using limited privileges on the remote computer with PsExec

This screenshot shows that I had only a few privileges. When I tried creating a user, I got an Access Denied error since the **cmd** process was launched using the -l switch with PsExec. It is a great way to run business applications either on remote or local computers with the user's default security context.