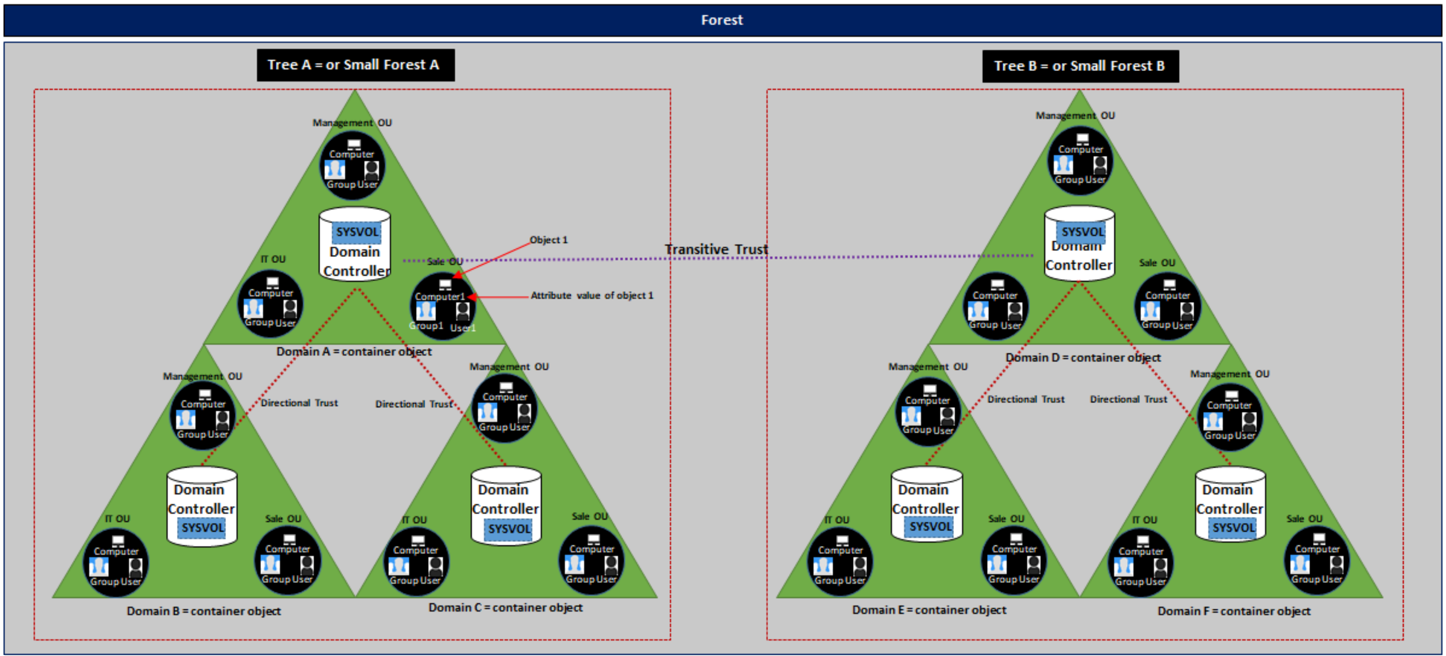

Forest
Tree A = or Small Forest A
Tree B = or Small Forest B
Management OU
Computer
Group User
SYSVOL
Domain Controller
IT OU
Sale OU
Object 1
Computer1
Group1 User1
Attribute value of object 1
Transitive Trust
Domain A = container object
Domain D = container object
Directional Trust
Directional Trust
Domain B = container object
Domain C = container object
Domain E = container object
Domain F = container object

The Ntds. dit file is **a database that stores Active Directory data**,
including information about user objects, groups, and group membership.
It includes the password hashes for all users in the domain. ... The extraction and cracking of these passwords can be performed offline, so they will be undetectable.

The term SYSVOL refers to **a set of files and folders that reside on the local hard disk of each domain controller in a domain** and that are replicated by the File Replication service (FRS).

Network clients access the contents of the SYSVOL tree by using the following shared folders: NETLOGON.

The **Security Account Manager** (SAM) is a database file in Windows XP, Windows Vista, Windows 7, 8.1 and 10 that stores users' passwords. It can be used to authenticate local and remote users. ... SAM uses cryptographic measures to prevent unauthenticated users accessing the system.

LSASS is the Local Security Authority Subsystem. It's ultimately responsible for making the access granted / access denied decision when you attempt to access resources in a Windows NT-derived operating system.

Each time you try to access any resource, a bit of code deep down in LSASS actually says "Yeah, go ahead" or "Woah! No way!"

Lsass.exe is an executable Windows file and stands for **Local Security Authority Subsystem Service**. As you can see the name of this process contains two words, “**Security Authority**,” this process controls the tasks of Windows 10 concerned with the security policy. For example, user’s verification in the server, user’s authentication during login, password changes, etc.

When you enter the wrong password during login into your account on Windows PC, it is the Lsass.exe process that displays the message “**Password does not match**.” If the lsass.exe process fails, the user immediately loses access to all his accounts on the Windows machine.

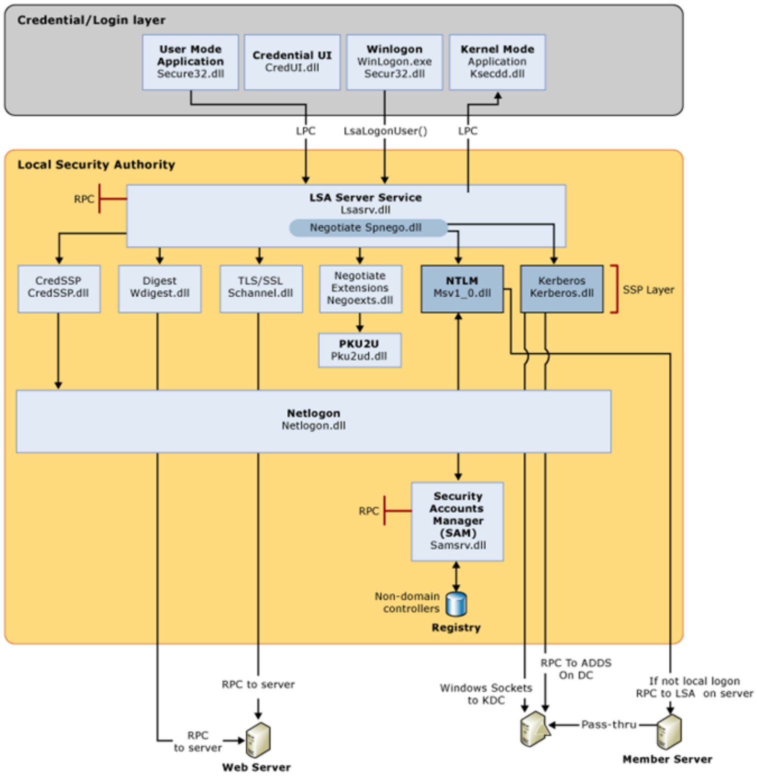
The **Local Security Authority (LSA)** is responsible for managing interactive logons to the system. ...
The SAM compares the user's credentials with the account information in the SAM database to determine whether the user is authorized to access the system.

The following diagram shows the components that are required and the paths that credentials take through the system to authenticate the user or process for a successful logon

At the backend, then, LSA must determine "who to ask" to determine if the credentials are indeed valid. That's where the "negotiate" block comes in - which will do one of two things:

•If the system is configured as a workstation, use the NTLM scheme as configured on the box and talk to the SAM database.

•If the system is configured as part of a domain, talk to the server to see what it supports, and give it what it needs.



1. Forest

# get details about the current forest.

```
Get-NetForest
```

# get details about another forest.

```
Get-NetForest -Forest dampy.com
```

# get all the domains in the current forest.

```
Get-NetForestDomain
```

# get all global catalogs for the current forest.

```
Get-NetForestCatalog
```

# determine which domain controller holds the PDC emulator FSMO role in the forest root domain

```
>> Get-ADForest | Select-Object -ExpandProperty RootDomain | Get-ADDomain | Select-Object -Property PDCEmulator
```

1.1. Domain

# Get domain information such as what forest is it in, all of the domain controllers, any child domain and the domain mode, which again tells us what kind of security is available.

```
>> Get-NetDomain
```

# get the same results for another domain, use the above command

```
Get-NetDomain -domain “Domain Name”
```

# get the domain SID (Security IDentifier is a unique ID number that a computer or domain controller uses to identify you).

```
Get-DomainSID
```

# get the policy of the current domain

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

1.1.1. Domaine Controller

# Identify administrator credentials in SYSVOL

>> We can use the PowerSploit’s get-GPPPassword

# Use this command to get information about the current domain controller (DC)

```
Get-NetDomainController
```

# identify groups and users have local administrative access on domain controllers

```
>> Get-NetDomainController | Get-NetLocalGroup –Recurse
```

1.1.2. OU (ACL)

# get all the OUs (Organization Units) in the current domain.

```
Get-NetOU
```

# To understand/identify what delegation has been configured on the OUs in the domain

```
>> Invoke-ACLScanner –ResolveGUIDs –ADSpath 'OU=X,OU=Y,DC=Z,DC=W' | Where {$_.ActiveDirectoryRights -eq 'GenericAll'}
```

# enumerate the ACLs for the users group.

```
Get-ObjectAcl -SamAccountName "users" -ResolveGUIDs
```

# see if there is any user has a modification rights to a GPO.

```
Get-NetGPO | %{Get-ObjectAcl -ResolveGUIDs -Name $_.Name}
```

# check if the user “Sarah” has the permission (Reset Password).

```
Get-ObjectAcl -SamAccountName labuser -ResolveGUIDs -RightsFilter "ResetPassword"
```

# Using PowerView we can also get the ACLs for all OUs where someone is allowed to read the LAPS password attribute, as follows.

```
>> Get-NetOU -FullData | Get-ObjectAcl -ResolveGUIDs | Where-Object {($_.ObjectType -like 'ms-Mcs-AdmPwd') -and ($_.ActiveDirectoryRights -match 'ReadProperty')} | ForEach-Object {$_ | Add-Member NoteProperty 'IdentitySID'
```

1.1.2.1. GROUP

# Use this command to get all the groups in the current domain.

```
Get-NetGroup
```

# get all the groups that contain the word “admin” in the group name.

```
Get-NetGroup *admin*
```

# Get the group membership of the user “Khalid”

```
Get-NetGroup –UserName "khalid"
```

# Identifying Computers Having Admin Rights

```
>> Get-NetGroup "*admins*" | Get-NetGroupMember –Recurse |?{$_.MemberName –Like '*$'}
```

##

# get the members of the group “Domain Admin”

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

# request the members of a particular group

```
>> Get-NetGroupMember 'Domain Admins' –Recurse
```

# identify administrator accounts indirectly

```
>> Get-NetGroupMember –GroupName "Denied RODC Password Replication Group" –Recurse
```

##

# get all the local administrators on a machine. (Note that it needs administrative rights).

```
Get-NetLocalGroup –ComputerName Client-02
```

# Retrieve more information using Get-NetLocalGroup

```
>> Get-NetLocalGroup -ComputerName computer_name
```

# Get local group membership with the NetLocalGroupGetMembers API call.

```
>> Get-NetLocalGroup -ComputerName computer_name –API
```

# The following retrieves the names of the local groups themselves.

```
>> Get-NetLocalGroup -ComputerName computer_name –ListGroups
```

# determine the actual users having RDP rights

```
>> Get-NetLocalGroup -ComputerName computer_name -GroupName "Remote Desktop Users" –Recurse
```

# Find groups in a remote domain that include users not in the target domain.

```
>> Find-ForeignGroup -Domain els.local
```

1.1.2.2. Users

# list all the users in the current domain with information about each user

```
Get-NetUser
```

# Identify potentially privileged accounts using the AdminCount property only without group enumeration

```
>> Get-NetUser –AdminCount | select name,whencreated,pwdlastset,lastlogon
```

# see the last password set of each user in the current domain.

```
Get-UserProperty –Properties pwdlastset
```

# Search for the word “pass” in the field “description” for each user in the domain.

```
Find-UserField -SearchField Description –SearchTerm “pass”
```

# Get the list of effective users who can access a target system

```
>> Get-NetLocalGroup -ComputerName computer_name –Recurse
```

# find all machines on the current domain where the current user has local admin access.

```
Find-LocalAdminAccess
```

# find all machines on the current domain where the current user has local admin access.

```
Find-LocalAdminAccess
```

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

```
Invoke-EnumerateLocalAdmin
```

# find computers where a domain has logged in

```
Invoke-UserHunter
```

# find computers where a specific user has sessions

```
Invoke-UserHunter -UserName “Aziz"
```

# find computers where a domain admin is logged in and current user has access.

```
Invoke-UserHunter –CheckAccess
```

1.1.2.3. Computers

# Use this command to list all the computers in the current domain.

```
Get-NetComputer
```

# list all the operating systems “Windows 7 Ultimate”.

```
Get-NetComputer –OperatingSystem "Windows 7 Ultimate"
```

# get all the pingable computers (live hosts) in the current domain.

```
Get-NetComputer -Ping
```

# identify machines inside the domain

```
>> get-adcomputer –filter * -Properties ipv4address | where {$_.IPV4address} | select name,ipv4addressor
>> get-adcomputer -filter {ipv4address -eq 'IP'} -Properties Lastlogondate,passwordlastset,ipv4address
```

# queries the domain for all the computer objects and then for each computer

```
>> Invoke-UserHunter -Stealth –ShowAll
```

# get actively logged users on a computer (Note that it needs administrative rights)

```
Get-NetLoggedon –ComputerName “Client-02”
```

# get the last logged user on a computer (Note that it needs administrative rights)

```
Get-LastLoggedOn –ComputerName Client-02
```

# SPN scanning:

```
  >> Get-ADComputer -filter {ServicePrincipalName -Like "*SPN*" } –Properties OperatingSystem,OperatingSystemVersion,OperatingSystemServicePack,PasswordLastSet,LastLogonDate,ServicePrincipalName,TrustedForDelegation,TrustedtoAuthForDelegation
```

# Retrieve the members of the 'Administrators' local group on a specific remote machine:

```
>> ([ADSI]'WinNT://computer_name/Administrators').psbase.Invoke('Members') | %{$_.GetType().InvokeMember('Name', 'GetProperty', $null, $_, $null)}
```

# find shares on the hosts in the current domain

```
Invoke-ShareFinder
```

1.1.2.4. POLICY (Domain, Group, User, Computers)

# discover all the group policies inside a domain

```
>>  Get-NetGPO | select displayname,name,whenchanged
```

# get a list of the GPO in the computer (Client-02).

```
Get-NetGPO -ComputerName client-02.fanzy.com
```

# find users who have local admin rights over the machine Client-02 through GPO.

```
Find-GPOComputerAdmin –Computername client-02.fanzy.com
```

# find all computers that ”Aziz” has local administrator rights in the current domain through the applied GPO.

```
Find-GPOLocation -UserName Aziz
```

# Identify all computers that the specified user has local RDP access rights to in the domain

```
>> Find-GPOLocation -UserName username -LocalGroup RDP
```

# identify which AD groups have admin rights to which computers

```
>> Get-NetGPOGroup
>> Get-NetGroupMember -GroupName "Local Admin"
```

# Request for all the members of “Domain Admins”

```
>> Get-NetGroupMember -GroupName 'Domain Admins' -FullData | %{ $a=$_.displayname.split(' ')[0..1] -join ' '; Get-NetUser -Filter "(displayname=*$a*)" } | Select-Object -Property displayname,samaccountname
```

2. Trust

# get a list of all domain trusts for the current domain to map the domain trust.

```
Get-NetDomainTrust
```

# map the trusts of a forest.

```
Get-NetForestTrust
```

# Enumerate all current domain trusts.

```
>> Get-NetUser -Domain associated_domain
```

# Find admin groups across a trust.

```
>> Get-NetGroup *admin* -Domain associated_domain
```

# Map all reachable domain trusts.

```
>> Invoke-MapDomainTrust
```

# Map all reachable domain trusts through LDAP queries, reflected through the current primary domain controller.

```
>> Invoke-MapDomainTrust -LDAP
```

# Export domain trust mappings for visualization

```
>> Invoke-MapDomainTrust | Export-Csv -NoTypeInformation trusts.csv
```

# Find users in the current domain that reside in groups across a trust.

```
>> Find-ForeignUser
```