SPECTEROPS
Creators of BloodHound

AdminSDHolder

Misconceptions & Misconfigurations

Notice I Didn't Say SDProp?

Jim Sykora

Table of Contents

Introduction

Do you recall the first time you ran into AdminSDHolder? Maybe you compromised a Domain Admin user and modified permissions to add persistence...only to have it disappear shortly thereafter?

My first go-around with AdminSDHolder was as a SysAdmin setting up BlackBerry Enterprise Server (BES). For the folks that aren't BES years old like me, this was a very popular middleware mobile email solution before the iPhone was released in 2007. Most everything worked well. Most users could click and clack away on their tiny BlackBerry keyboards to send and receive business emails, but there were a couple of users that kept having problems. They didn't have the correct SendAs permissions. They didn't even inherit the SendAs permissions at all. And when you manually added SendAs permissions, they disappeared shortly thereafter. Very frustrating!

The issue here is that those accounts having issues with SendAs permissions were members of the Print Operators group. *Why were they members of Print Operators? I'm not sure, because I never met the person who set that particular Active Directory (AD) Forest up.* The Print Operators group, by default, is one of the security principals in AD that is counted as privileged or sensitive. So AD tries to protect Print Operators and its members. *A bunch of other security principals are also protected and we'll get into that more in a bit.* But the protection mechanism that Active Directory uses to protect known-privileged principals is called AdminSDHolder.

Now that I've long since decommissioned my last BlackBerry Enterprise Server and know a little more about AD, I've discovered that there are a lot of misconceptions around how AdminSDHolder works, what it does, and (most importantly) what it doesn't do.

I'm hoping to set that straight and get things technically correct: the pedantic, yet best, type of correct.

Security Descriptor Basics

To understand what AdminSDHolder does (and doesn't) do, we first need to understand what a Security Descriptor is because that's Admin Holder's middle name.

Security Descriptor Definition:

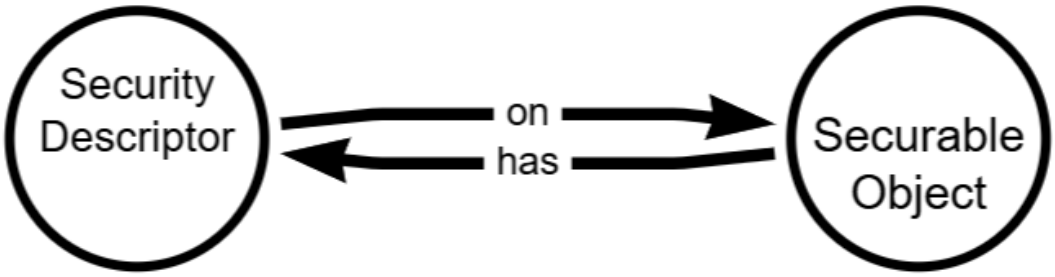
> *A structure and associated data that contains the security information for a Securable Object. A security descriptor identifies the object's owner and primary group. It can also contain a discretionary access control list (DACL) that controls access to the object, and a system access control list (SACL) that controls the logging of attempts to access the object.*

The security descriptor for an object determines which access is granted to whom. It provides the source of truth for the Windows Security Reference Monitor (SRM) to make decisions on whether to allow, deny, or audit access to an object.

Security descriptors are comprised of multiple parts:

- Owner Security Identifier (SID)
- Primary Group (generally only used for Posix compatibility)
- Security Descriptor Control is a set of bit flags that qualify the meaning of the security descriptor and its components.
- An optional DACL, which can be one of the following:
    - NULL
    - Empty
    - Comprised of one or more Access Control Entries (ACEs)
- An optional SACL which can be comprised of one or more ACEs

Overall, the security descriptor is the data that contains the security information for a Securable Object. What's a Securable Object? We're going around in circles here, but a Securable Object is any object that has a security descriptor. All named Windows objects are securable, as are some unnamed objects like process and thread objects. Almost everything in AD (or Windows, for that matter) is a Securable Object.



The Owner SID designates which security principal owns the object. The owner, by default, can read and write to the security descriptor. I have already covered a lot of information about object ownership in another paper: Owner or Pwned?

The primary group in the security descriptor is very rarely applicable or important in a homogenous Microsoft ecosystem. It is included for POSIX compatibility.

Note: The primary group in the security descriptor is a different concept from the primaryGroupID attribute of user-derived security principals that grants membership in the group with the RID value specified in the primaryGroupID.

Security Descriptor Control Flags determine behaviors of the security descriptor. The most commonly noticed flag is whether the DACL is protected from inheritance.

A DACL is an access control list that is controlled by the Owner of a Securable Object to specify the access particular users and groups can have to the object. ACEs are the individual access rules the DACL is composed of.

Security descriptors can also include an optional SACL. The purpose of a SACL is to provide a method for auditing access to the object and controlling mandatory integrity checks, when applicable.

Note: For a very detailed understanding of how security descriptors work, along with the entire Windows access model and so much more, I recommend getting a copy of James Forshaw's book Windows Security Internals - A Deep Dive into Windows Authentication, Authorization, and Auditing.

Access Checks

Any time access is requested to an object in the Windows ecosystem, an access check is performed. In the simplest of scenarios, the security context of a user, including all the groups the user is a member of

and the privileges assigned to that user exists as an Access Token in the Local Security Authority (LSA). The Access Token provides the authorization security context of that user.

The Access Token is combined with an Access Request, which is an access mask that specifies specific permissions that are desired, and sent to the Windows Security Reference Monitor (SRM). The SRM is part of the Windows Kernel and performs all decisions on granting, denying, or auditing access. When a process or thread running in the context of the user desires access to an object and has presented an Access Request to the SRM, the SRM will communicate with the Windows Kernel's Object Manager to retrieve the security descriptor for that object.

The SRM now has all the information it needs to determine if that thread can access the target object. Integrity and Ownership are checked first before proceeding to the DACL, where each ACE is compared in order against the user's Access Token and Access Request. If an applicable Deny ACE matches the Access Token and Access Request, the request is denied. If an Allow ACE matches the Access Request and any SID in the Access Token which is not a deny-only SID, access is granted. If no ACE is matched to grant access, the request is denied. ACEs in the SACL are also processed to see if Windows should trigger any audit events.

This is a relatively simplified overview of the Access Check process which leaves out a lot of caveats and edge cases. Here are a couple of edge cases:

- Previously, I mentioned NULL and empty DACLs. An empty DACL will deny access to everyone (except the object Owner, who can inherently WriteDACL if not specifically prohibited). A null DACL allows access to Everyone; however, Active Directory has rules enforced by code to make creation of a null DACL highly unlikely–or, at least, null DACLs are significantly less likely to occur in AD than for objects such as processes and threads.
- The order of ACEs in a DACL matters. The SRM iterates through the ACEs in a DACL in the order they were in when the Object Manager passed the security descriptor over to it. It is often understood that Deny ACEs come before Allow ACEs and that, if there is conflict between Deny and Allow ACEs, the Deny ACE wins. This is generally true when the DACL is in canonical order, which has been the default preferred state since Windows Server 2003 Forest Functional Level. When a DACL is in canonical order, the ACEs are placed in order such that explicit ACEs defined directly on the object are before inherited ACEs and Deny ACEs are before Allow ACEs. This blog by Raymond Chen explains why deny ACEs are before allow ACEs: Why does canonical order for ACEs put deny ACEs ahead of allow ACEs? Additionally, the MS-ADTS specifications have a section on Checking Accesses which then goes into details on the algorithms for checking simple access, checking object-specific access, checking control access right access, checking validated write access, and checking object visibility.
  With all this information stating that ACEs are evaluated in sequence, starting with the first ACE, an appropriate question might be, "Are all DACLs in canonical order?" To which I might respond with the question, "If all DACLs are in canonical order, why do security descriptor properties in .NET like CommonAcl have a 'IsCanonical' property with the possibility of that boolean value being false?"

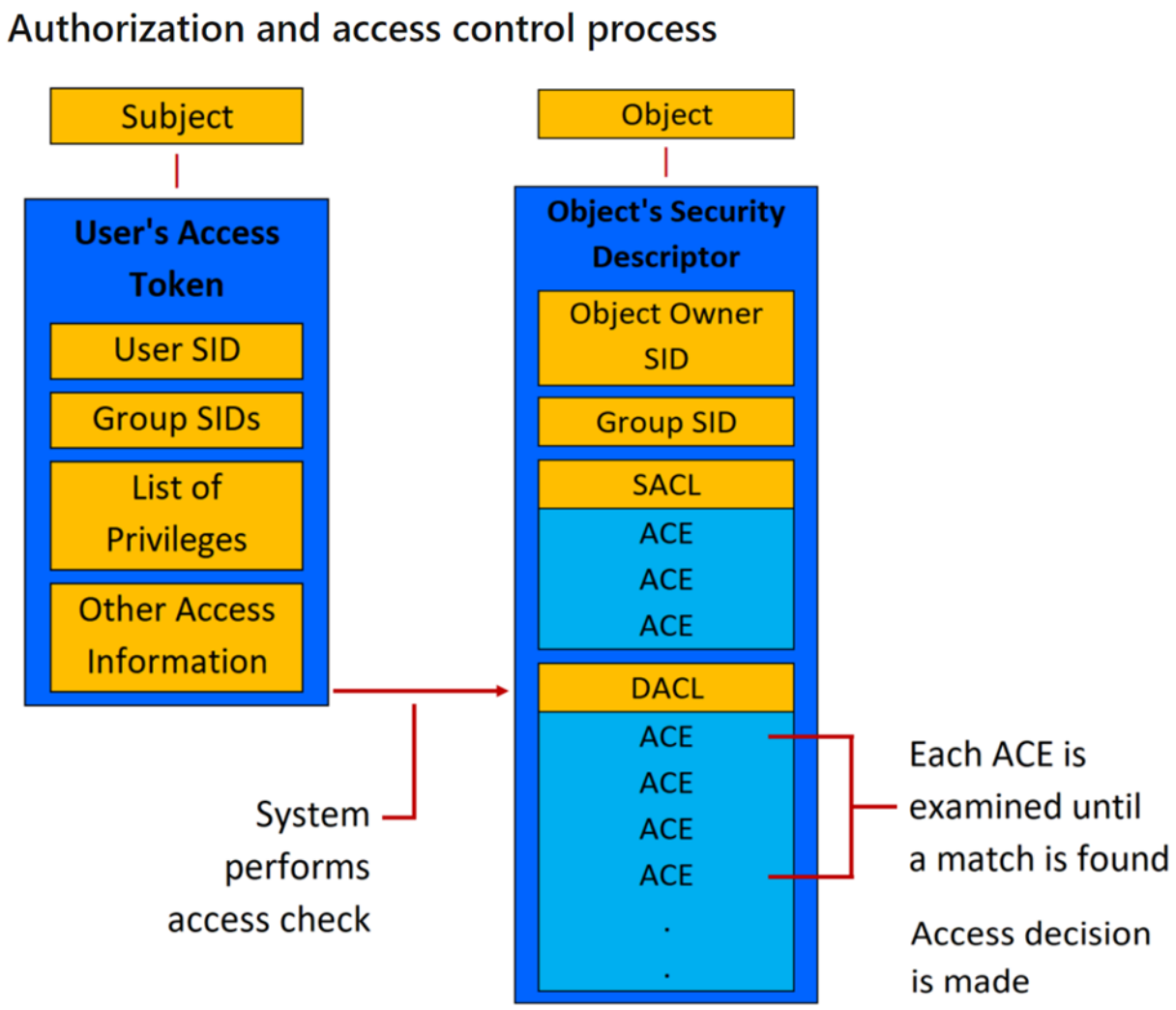


Figure 1 - Microsoft Authorization & Access Control Process

Propagation and Inheritance

Manually setting individual permissions on every individual object in Windows or AD would quickly become untenable whether you or the system attempted to manage it that way. To simplify administration and minimize how often permissions need to be individually assigned inheritance is utilized. Security inheritance refers to how child objects can inherit permissions configured on a parent object. In AD, not all permissions on a parent object are inheritable. The granular configurability of AD permissions allows for an ACE to apply permissions only directly to an object, to an object and its descendants, just to its descendants, only to descendents of a certain object type, and a few other options.

When a parent container or organizational unit (OU) has inheritable ACEs configured on it, those inheritable permissions propagate down to descendant objects within the container or OU, again depending on how the inheritable ACE was configured. By default, a child object will gladly accept any inheritable ACEs from its parent. This results in a scenario where child objects usually have the same permissions as their parent.

Sometimes it is not appropriate for child objects to inherit propagated permissions from their parent object. The DACL_Protected flag prevents inheritable permissions from an object's parent being propagated to it just as the SACL_Protected flag prevents audit rules from being propagated from parent objects. These security descriptor flags will appear in AD Users and Computers, and most other standard Windows Security settings GUI forms as "Disable inheritance" and "Enable inheritance" buttons.

When the button looks like this, the DACL_Protected flag is enabled:

When the button looks like this, the DACL_Protected flag is not set, which is the default:

Inheritable permissions flow from the root of the directory hierarchy down through each layer of containers, OUs, and objects. If more inheritable permissions are assigned on an OU or container that an object is nested within, the inheritable permissions are additive in their propagation, with each layer of inheritable permissions creating a large DACL on the child objects.

Here is an example of an AD domain which has many nested OUs. Each OU adds another inheritable ACE on the DACL as the hierarchy goes deeper, until we arrive at the OU where the six user accounts are. Of the six user accounts, three are standard users and three are admins. The three admin accounts are all members of the Domain Admin group for the *DOMAIN.ROOT* domain, which means AdminSDHolder will protect them. These admin accounts have the DACL_Protected flag on their

security descriptor because of their AdminSDHolder protection, which is represented by the purple umbrella emoji:

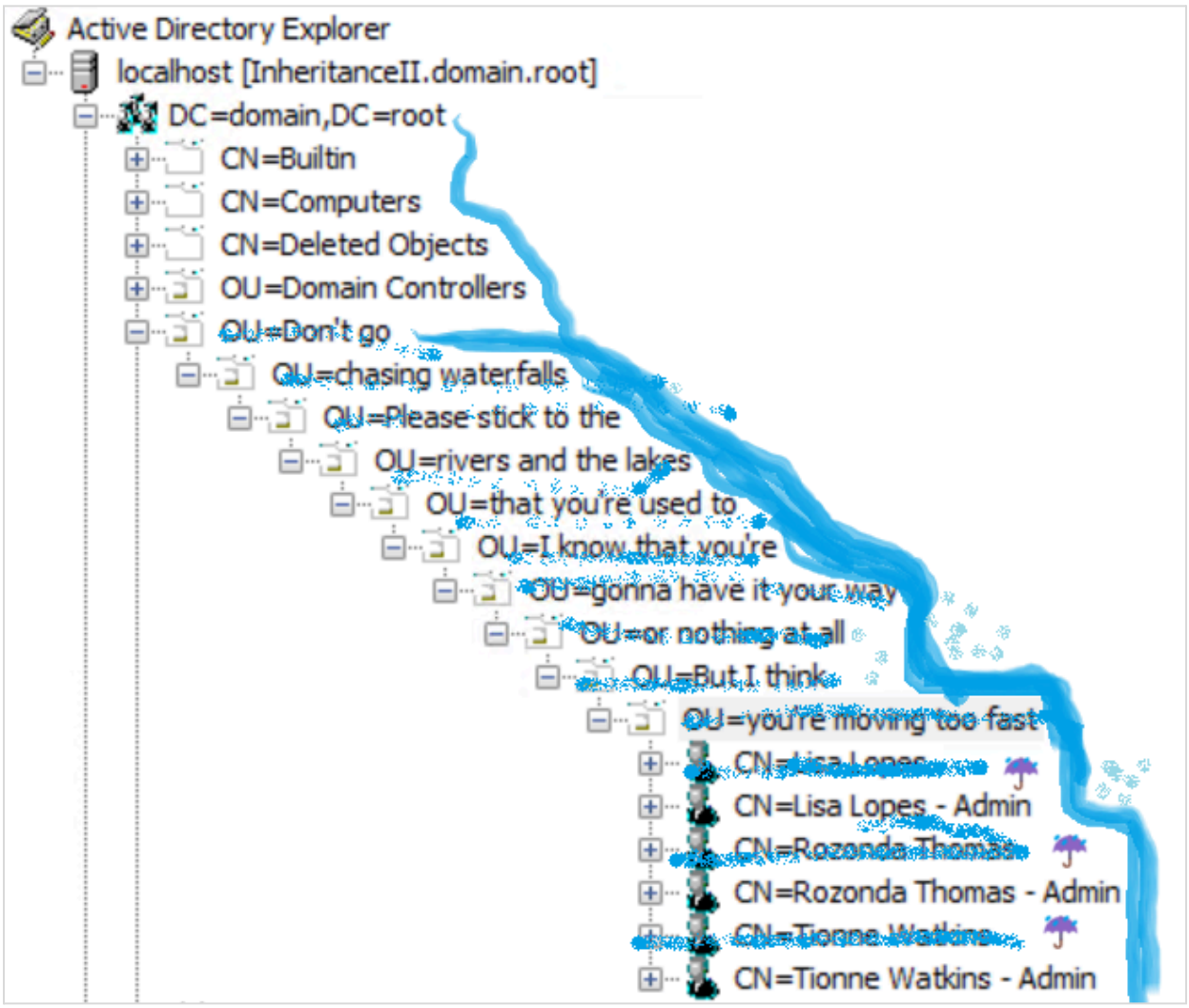
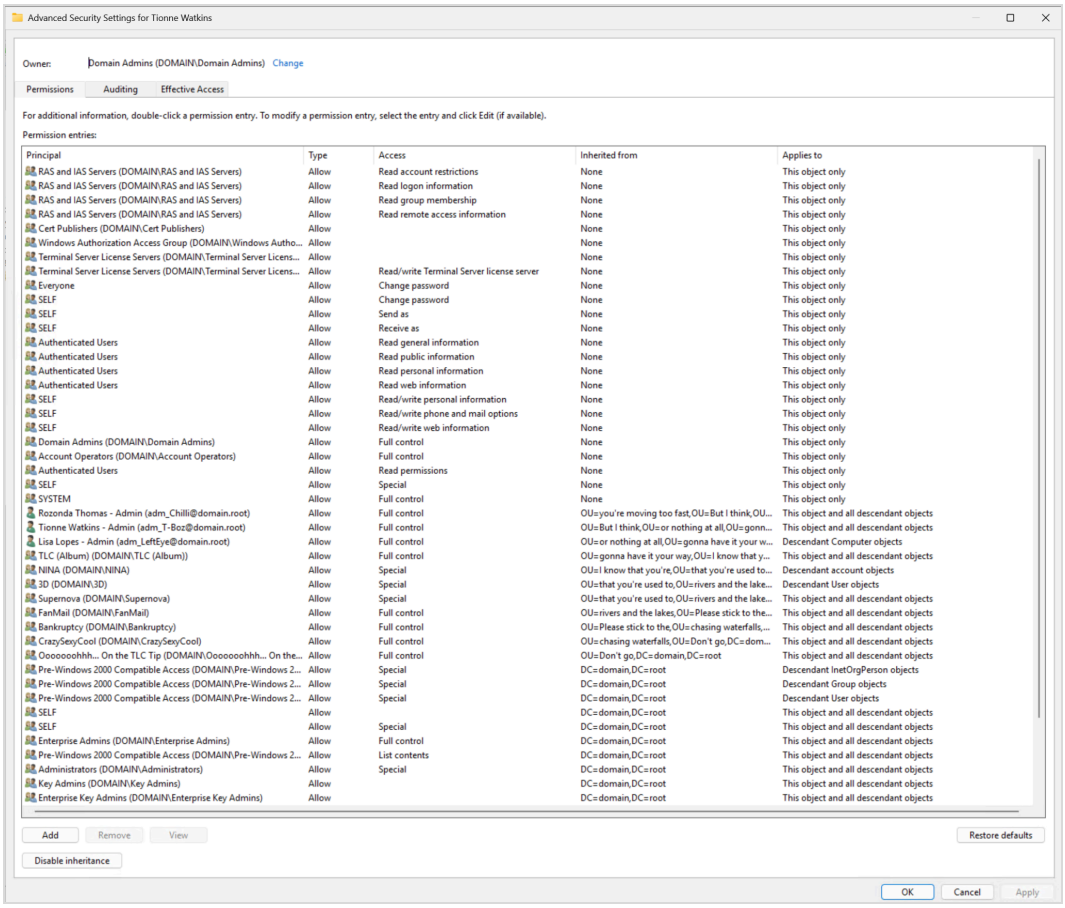


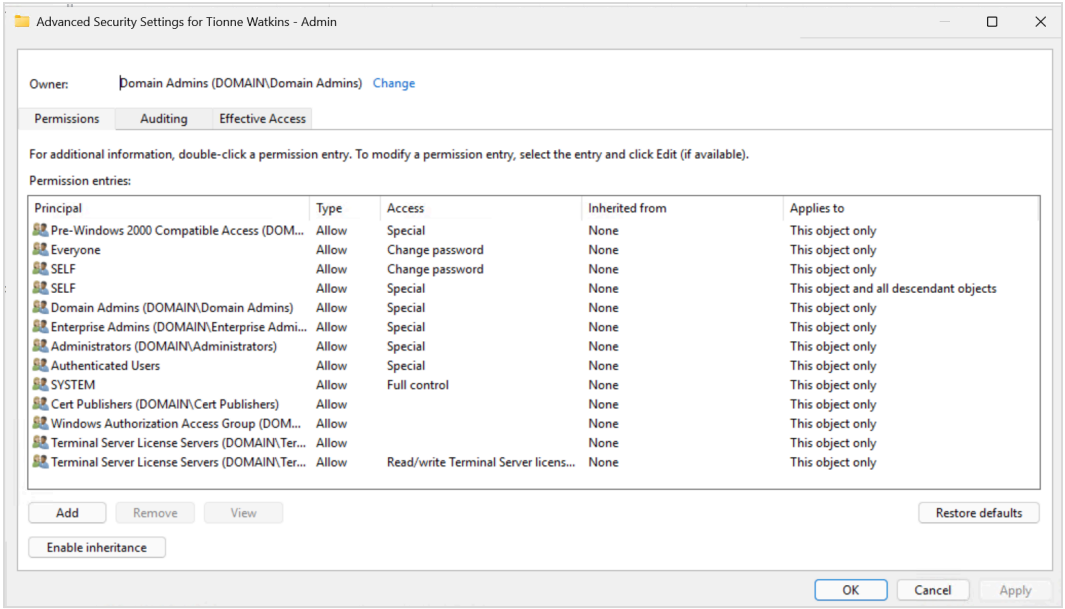
Figure 2 - An MSPaint Waterfall of Permissions

If we imagine the inheritable permissions as a river that we're used to, each container or OU is a potential tributary that can increase the flow of water (i.e., permissions) through the river. The river flows through the OUs and containers until it gets to one where user objects are located. Here the river spills all of its inheritable permissions down on the users in that container like a waterfall. And if you stand under a waterfall, you're gonna get wet; however, in this case, the Admin accounts have their trusty AdminSDHolder umbrella above them and it keeps them nice and dry. Hopefully I'm not moving too fast.

If we have a look at T-Boz's standard user account, we see that all sorts of inheritable permissions propagate down to it and that inheritance is enabled:



By comparison, T-Boz's admin account has a very simple, dry security descriptor:



I probably won't get any FanMail about using TLC to explain the propagation and inheritance of security permissions in AD, but it's worth a try.

AD enforces a set of rules for security descriptors, and here I'm repeating the first two rules:

1. The security descriptor of each object retains the set of explicit (non-inherited) ACEs stamped in its ACL (along with the required Owner SID and optional Group SID).
2. The security descriptor also includes the set of inheritable ACEs from its parent object. ACEs that are applicable to the child object are included. Also, ACEs that are not applicable to the direct child object are included as they may be applicable to grandchild objects. The following exceptions apply to the inheritance rule:
    a. The object is the root of a Naming Convention (NC).
    b. The security descriptor has the appropriate DACL or SACL protection flag set.
    c. The object is deleted.

When the DACL or SACL protection flags are set in the security descriptor, child objects will not inherit permissions that propagate down from parent objects. Permission propagation and inheritance can be an incredibly powerful tool for least-privilege delegation of permissions and without a clean and clearly defined OU structure with rigorous attention to detail on exactly where privileged objects are placed in the hierarchy, there would be multiple default attack paths in AD if the AdminSDHolder process wasn't there.

Default Security Descriptors

I often get the impression that many folks in IT Operations and Information Security feel like the primary purpose and intent of AdminSDHolder is to disable inheritance (i.e., set a protected DACL). Disabling inheritance of permissions on highly privileged objects is certainly key, but it's also important to understand that many objects, including most security principals, have default security descriptors that define many of the explicit permissions on objects when created.

The defaultSecurityDescriptor attribute is optional on object classes defined in the AD schema. When defined, it contains the DACL (and perhaps SACL) portion of a security descriptor in SDDL format. Here is an example from a Windows Server 2025 domain showing the defaultSecurityDescriptor for all computer objects in SDDL format:

```
Expanding base 'CN=Computer,CN=Schema,CN=Configuration,DC=AD2025,DC=lan'...
Getting 1 entries:
Dn: CN=Computer,CN=Schema,CN=Configuration,DC=AD2025,DC=lan
    adminDescription: Computer;
    adminDisplayName: Computer;
    auxiliaryClass: ipHost;
    cn: Computer;
    defaultHidingValue: FALSE;
    defaultObjectCategory: CN=Computer,CN=Schema,CN=Configuration,DC=AD2025,DC=lan;
    defaultSecurityDescriptor: D:(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;DA)(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;AO)(A;;RPWPCRCCDCLCLORCWOWDSDDTSW;;;SY)(A;;RPCRLCLORCSDDT;;;CO)
        (OA;;WP;4c164200-20c0-11d0-a768-00aa006e0529;;CO)(A;;RPLCLORC;;;AU)(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)(A;;CCDC;;;PS)(OA;;CCDC;bf967aa8-0de6-11d0-a285-00aa003049e2;;PO)
        (OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)(OA;;SW;f3a64788-5306-11d1-a9c5-0000f80367c1;;PS)(OA;;RPWP;77B5B886-944A-11d1-AEBD-0000F80367C1;;PS)(OA;;SW;72e39547-7b18-11d1-
        adef-00c04fd8d5cd;;PS)(OA;;SW;72e39547-7b18-11d1-adef-00c04fd8d5cd;;CO)(OA;;SW;f3a64788-5306-11d1-a9c5-0000f80367c1;;CO)(OA;;WP;3e0abfd0-126a-11d0-
        a060-00aa006c33ed;bf967a86-0de6-11d0-a285-00aa003049e2;CO)(OA;;WP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967a86-0de6-11d0-a285-00aa003049e2;CO)(OA;;WP;bf967950-0de6-11d0-
        a285-00aa003049e2;bf967a86-0de6-11d0-a285-00aa003049e2;CO)(OA;;WP;bf967953-0de6-11d0-a285-00aa003049e2;bf967a86-0de6-11d0-a285-00aa003049e2;CO)(OA;;RP;46a9b11d-60ae-405a-b7e8-
        ff8a58d456d2;;S-1-5-32-560);
```

Figure 3 - Computer Object Schema With defaultSecurityDescriptor in SDDL

Here is the same computer object schema defaultSecurityDescriptor presented in a GUI format for easier reading:

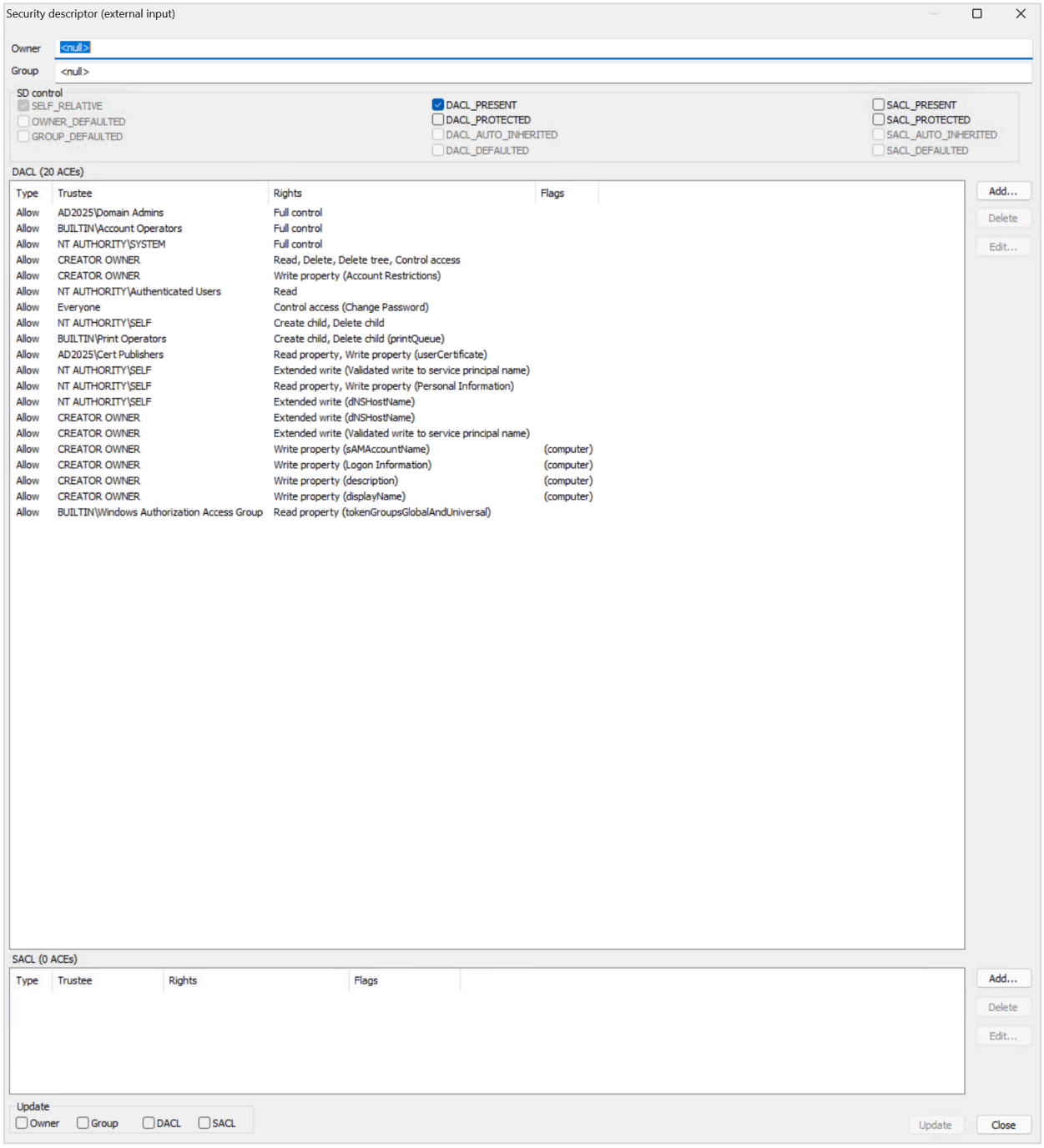


Figure 4 - Computer Object Schema defaultSecurityDescriptor Converted to Display in LDP's SD Edit Function

The allow ACE granting Account Operators explicit Full Control over the computer object stands out as the most glaring issue here, but the CREATOR OWNER and SELF ACEs can also allow an attacker to abuse the computer object. These explicit ACEs will be present, even if the object is placed in an OU where DACL inheritance is disabled. Even if a Deny ACE is created in the parent OU, the explicit ACE takes priority over an inherited ACE.

The defaultSecurityDescriptor on computer objects is perhaps the most abusable, but it's important to understand that users, groups, OUs, managed service accounts, and more all have defaultSecurityDescriptors. You can find more examples of those on my GitHub here: https://github.com/JimSycurity/AdminSDHolder/tree/main/SchemaAndDomainDefaults

The only way to protect an object that will have default explicit ACEs which could be abused that are assigned at creation is to remove those abusable ACEs. One could script this out, but AdminSDHolder does it automatically by wholly replacing the entire security descriptor, DACL included.

DACL Abuse and Attack Paths

As far as attack paths that abuse DACLs are concerned, there are a lot of great resources available that explain it without me turning this into a 150+ page document (**spoiler**: I went beyond this. Sorry!).

Microsoft Advanced Threat Analytics Team released a blog on Active Directory Access Control List – Attacks and Defense back in 2017. It covers the basics of DACL abuse well and, as a bonus, includes at least one common AdminSDHolder Misconception.

There are a lot of other great resources on DACL abuse, also known as ACL Attack Paths.
Here are a few:

- Heat-ray: Combating Identity Snowball Attacks Using Machine Learning, Combinatorial Optimization and Attack Graphs (2009)
- BloodHound 1.3 – The ACL Attack Path Update
- The Attack Path Management Manifesto
- An ACE up the Sleeve (pdf)
- The Hacker Recipes: DACL Abuse
- SecureIdeas: Watching yOUr Permissions
- BloodHound Edges

BloodHound is both a free open-source software (FOSS) and Enterprise software solution that maps out attack paths in AD, Entra ID, and Azure environments. Every important object in an environment is a Node. Traversable paths between Nodes are Edges. In the early versions of BloodHound, specifically version 1.3, ACL Attack Paths were a primary focus and the majority of the edges available. Currently, I count 20 unique Edges in BloodHound which specify ACL Attack Paths.

AdminSDHolder Purpose & Intentions

AdminSDHolder is an AD object **and associated process** in AD Domain Services (AD DS) that helps protect specific sensitive and highly privileged accounts in AD from lower-privileged accounts manipulating them. AdminSDHolder does this by applying a special security descriptor (SD) to the privileged accounts it helps protect.

The AdminSDHolder AD Object itself is located in the System container of the domain's default naming context. It has a DistinguishedName of "CN=AdminSDHolder,CN=System,DC=contoso,DC=com" if *CONTOSO.COM* were the domain's name. The AdminSDHolder object has a default security descriptor that's pretty special in AD DS.

By default, the AdminSDHolder SD:

- Has a stringent, secure DACL:
    - Write (Modify) permissions only granted to Administrators, Domain Admins, and Enterprise Admins
    - SYSTEM is the only principle granted Full control (GenericAll)
- The Owner is Domain Admins instead of Administrators
- Inheritance is disabled so that no parent permissions are inherited

Special indeed!

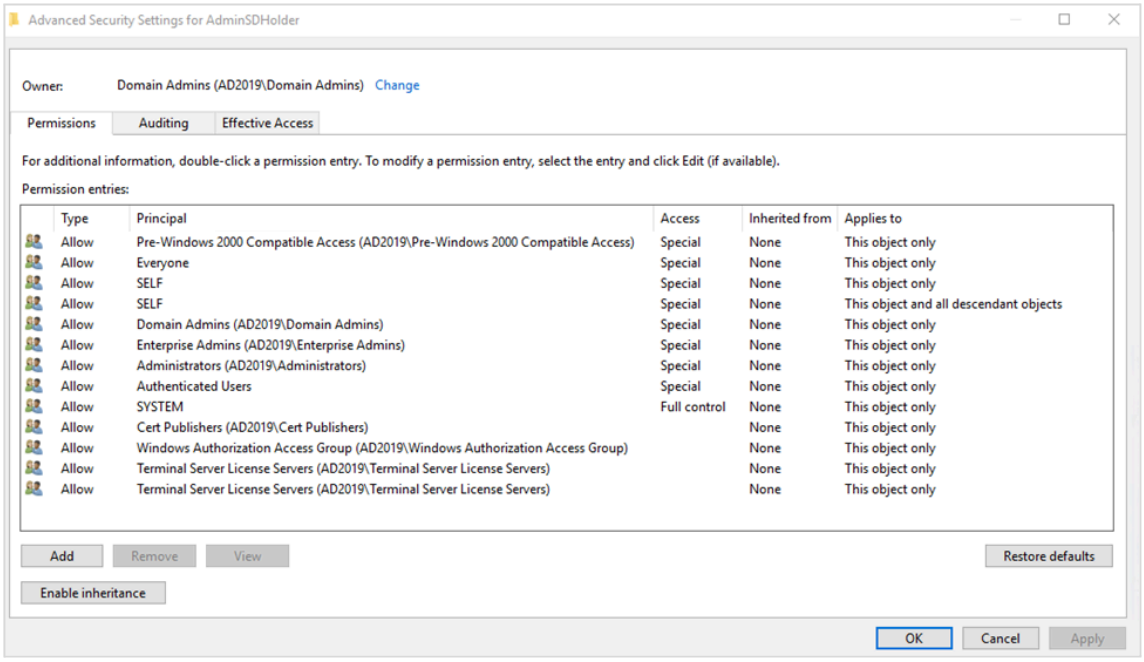


Figure 5 - Screenshot of Advanced Security Settings for AdminSDHolder in Windows Server 2019

The screenshot above is what a lot of folks are used to seeing when looking at permissions on an object. ; however, it doesn't give a full picture of what the DACL and ACEs look like. For example, all of the entries with an Access value of "Special" are ACEs that the default Windows and Active Directory Users and Computers (ADUC) Security Settings can't fully describe. That could be something as simple as Read access or a combination of multiple ACEs. To dig deeper, we'll utilize some other tools: ldp.exe, PowerShell, and dsacls.exe. Here's a screenshot of the same AdminSDHolder object from ldp.exe:

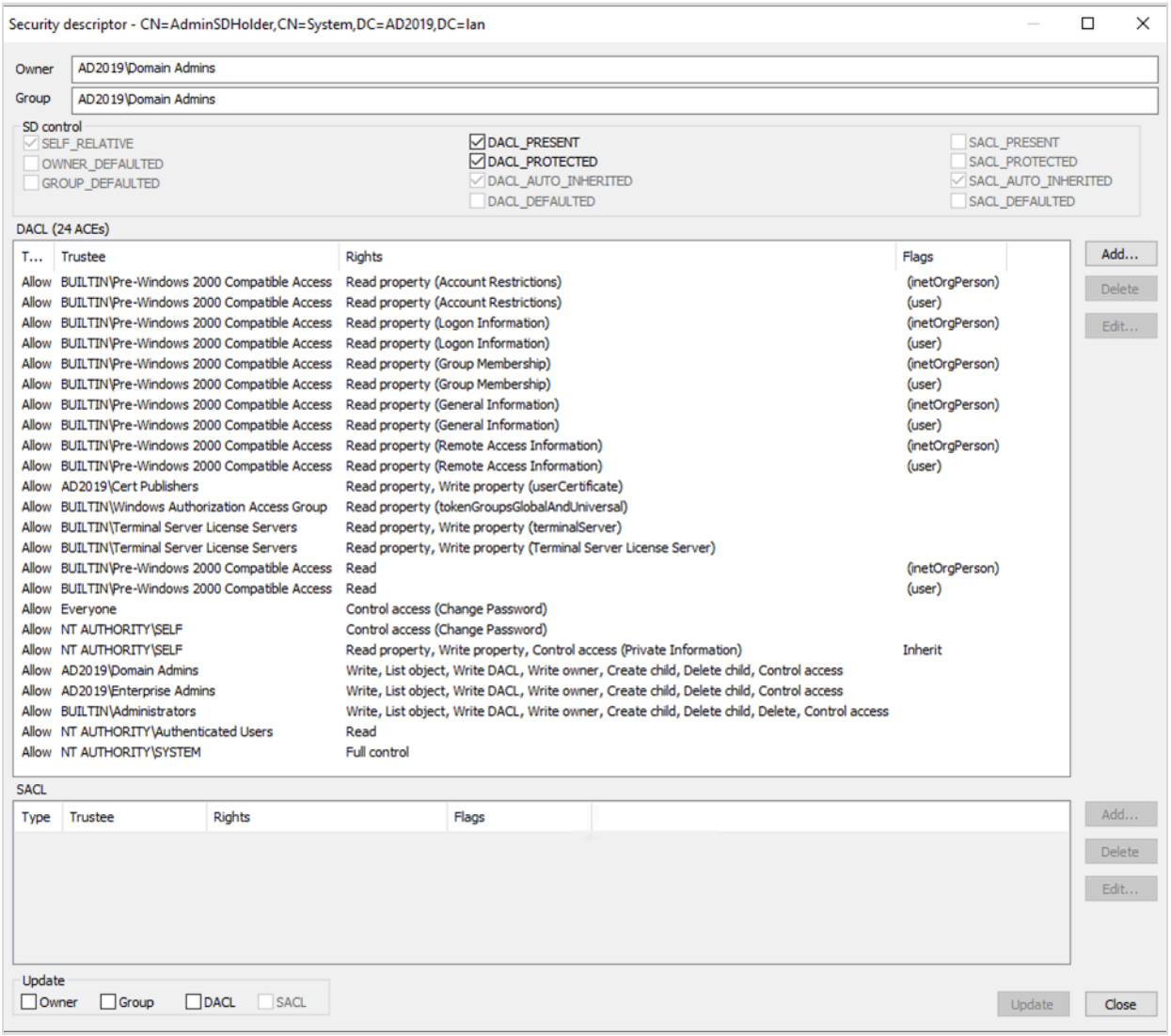


Figure 6 - Screenshot of LDP.exe Security Descriptor for AdminSDHolder in Windows Server 2019

The ldp.exe view of the security descriptor shows more details. Each individual ACE is shown, all 24 of them. We can see the full contents of each ACE in plain terms.

Note: I don't recommend doing any serious security descriptor work with ADSIEdit as it does not accurately portray ACEs.

There are a few ways to review the security descriptor with PowerShell also. One way is to get the object and then expand the nTSecurityDescriptor properties. This can be useful because it's possible to view the ACEs in SDDL format, which we'll get to shortly. Accessing security descriptors in PowerShell also allows them to be manipulated programmatically. There are also third-party tools written for PowerShell which allow reviewing security descriptors and permissions.

```
PS AD:\> $AdminSDHolder = Get-ADObject 'CN=AdminSDHolder,CN=System,DC=AD2019,DC=lan' -Properties ntSecurityDescriptor

PS AD:\> $AdminSDHolder


DistinguishedName    : CN=AdminSDHolder,CN=System,DC=AD2019,DC=lan
Name                 : AdminSDHolder
ntSecurityDescriptor : System.DirectoryServices.ActiveDirectorySecurity
ObjectClass          : container
ObjectGUID           : d77a7b04-5511-45b0-8bad-ca88e5af0182



PS AD:\> $AdminSDHolder.ntSecurityDescriptor

Path Owner                Access
---- -----                ------
     AD2019\Domain Admins NT AUTHORITY\Authenticated Users Allow  ...


PS AD:\> $AdminSDHolder.ntSecurityDescriptor.Access


ActiveDirectoryRights : GenericRead
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

ActiveDirectoryRights : CreateChild, DeleteChild, Self, WriteProperty, ExtendedRight, Delete, GenericRead, WriteDacl, WriteOwner
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : BUILTIN\Administrators
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

Figure 7 - Screenshot of PowerShell Security Descriptor for AdminSDHolder in Windows Server 2019

```
PS AD:\> $ACL = Get-ACL 'CN=AdminSDHolder,CN=System,DC=AD2019,DC=lan'

PS AD:\> $ACL

Path                                                                                                                Owner                Access
----                                                                                                                -----                ------
Microsoft.ActiveDirectory.Management.dll\ActiveDirectory::://RootDSE/CN=AdminSDHolder,CN=System,DC=AD2019,DC=lan AD2019\Domain Admins NT AUTHORITY\Authenticated Users Allow  ...


PS AD:\> $ACL.Access


ActiveDirectoryRights : GenericRead
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

ActiveDirectoryRights : CreateChild, DeleteChild, Self, WriteProperty, ExtendedRight, Delete, GenericRead, WriteDacl, WriteOwner
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : BUILTIN\Administrators
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None

ActiveDirectoryRights : CreateChild, DeleteChild, Self, WriteProperty, ExtendedRight, GenericRead, WriteDacl, WriteOwner
InheritanceType       : None
ObjectType            : 00000000-0000-0000-0000-000000000000
InheritedObjectType   : 00000000-0000-0000-0000-000000000000
ObjectFlags           : None
AccessControlType     : Allow
IdentityReference     : AD2019\Domain Admins
IsInherited           : False
InheritanceFlags      : None
PropagationFlags      : None
```

Figure 8 - Screenshot of PowerShell Get-Acl Security Descriptor for AdminSDHolder in Windows Server 2019

Operating systems (OS) prior to Windows Server 2008 didn't include PowerShell by default and PowerShell 1.0 was not widely installed on Server 2003 from my understanding. To access AD security descriptor information in text format, the DSACLS command-line utility is required on older server OSes and can still be used on modern OSes as well.

```
PS AD:\> dsacls.exe "CN=AdminSDHolder,CN=System,DC=AD2019,DC=lan" /A
Owner: AD2019\Domain Admins
Group: AD2019\Domain Admins

Audit list:
Success Everyone  SPECIAL ACCESS
                  WRITE PERMISSIONS
                  CHANGE OWNERSHIP
                  WRITE PROPERTY

Permissions inherited to subobjects are:
Inherited to organizationalUnit
Success Everyone  SPECIAL ACCESS for gPLink   <Inherited from parent>
                  WRITE PROPERTY
Success Everyone  SPECIAL ACCESS for gPOptions   <Inherited from parent>
                  WRITE PROPERTY

Access list:
{This object is protected from inheriting permissions from the parent}
Allow BUILTIN\Pre-Windows 2000 Compatible Access
                                  SPECIAL ACCESS
                                  READ PERMISSONS
                                  LIST CONTENTS
                                  READ PROPERTY
                                  LIST OBJECT
Allow BUILTIN\Pre-Windows 2000 Compatible Access
                                  SPECIAL ACCESS
                                  READ PERMISSONS
                                  LIST CONTENTS
                                  READ PROPERTY
                                  LIST OBJECT
Allow AD2019\Domain Admins        SPECIAL ACCESS
                                  READ PERMISSONS
                                  WRITE PERMISSIONS
                                  CHANGE OWNERSHIP
                                  CREATE CHILD
                                  DELETE CHILD
                                  LIST CONTENTS
                                  WRITE SELF
                                  WRITE PROPERTY
                                  READ PROPERTY
                                  LIST OBJECT
                                  CONTROL ACCESS
Allow AD2019\Enterprise Admins    SPECIAL ACCESS
                                  READ PERMISSONS
                                  WRITE PERMISSIONS
                                  CHANGE OWNERSHIP
                                  CREATE CHILD
                                  DELETE CHILD
                                  LIST CONTENTS
                                  WRITE SELF
                                  WRITE PROPERTY
                                  READ PROPERTY
                                  LIST OBJECT
                                  CONTROL ACCESS
Allow BUILTIN\Administrators      SPECIAL ACCESS
                                  DELETE
```

Figure 9 - Screenshot of DSACLS.exe Security Descriptor for AdminSDHolder in Windows Server 2019

The default security descriptor for AdminSDHolder varies by AD Schema version; however, there are only three revisions as of June 2025.

For reference, the following table maps the schema version to its corresponding OS version:

Version	Operating System
91	Windows Server 2025
88	Windows Server 2022
88	Windows Server 2019
87	Windows Server 2016
69	Windows Server 2012 R2
56	Windows Server 2012
47	Windows Server 2008 R2
44	Windows Server 2008 RTM
31	Windows Server 2003 R2
30	Windows Server 2003 RTM
13	Windows Server 2000

The defaults by schema version are specified in the MS-ADTS specifications:

Schema version 13:

```
O:DA
G:DA
D:PAI(A;;LCRPLORC;;;AU)(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)
(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)
(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)
(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
```

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)

S:AI(AU;CIIDSAFA;CCDCSWWPDTCRSDWDWO;;;WD)

Schema versions 30-31:

O:DA

G:DA

D:PAI(A;;LCRPLORC;;;AU)

(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)

(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;PS)

(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;LCRPLORC;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8a58d456d2;;S-1-5-32-560)

(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5-32-561)

S:AI(AU;SA;WPWDWO;;;WD)

(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)

(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)

Schema versions 44-91:

```
O:DA

G:DA

D:PAI(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28
;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)

(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8a58d456d2;;S-1-5-32-560)

(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5-32-561)

(OA;;RPWP;5805bc62-bdc9-4428-a5e2-856a0f4c185e;;S-1-5-32-561)

(OA;;LCRPLORC;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;PS)

(OA;CI;RPWPCR;91e647de-d96f-4b70-9557-d63ff4f3ccd8;;PS)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)

(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)

(A;;LCRPLORC;;;AU)

(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)

S:AI(AU;SA;WPWDWO;;;WD)

(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa00304
9e2;WD)

(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa00304
9e2;WD)
```

Note: For a broad sample of the default security descriptor for AdminSDHolder across various Windows Server OSes, refer to: https://github.com/JimSycurity/AdminSDHolder/blob/main/SchemaAndDomainDefaults/SchemaINIData.md

This is challenging for most folks to read because it's in the SDDL format. We could use some of the knowledge we gained in the previous section on security descriptor basics. Instead, let's use the Display SDDL in Editor utility (included in the most recent versions of LDP) to translate it.

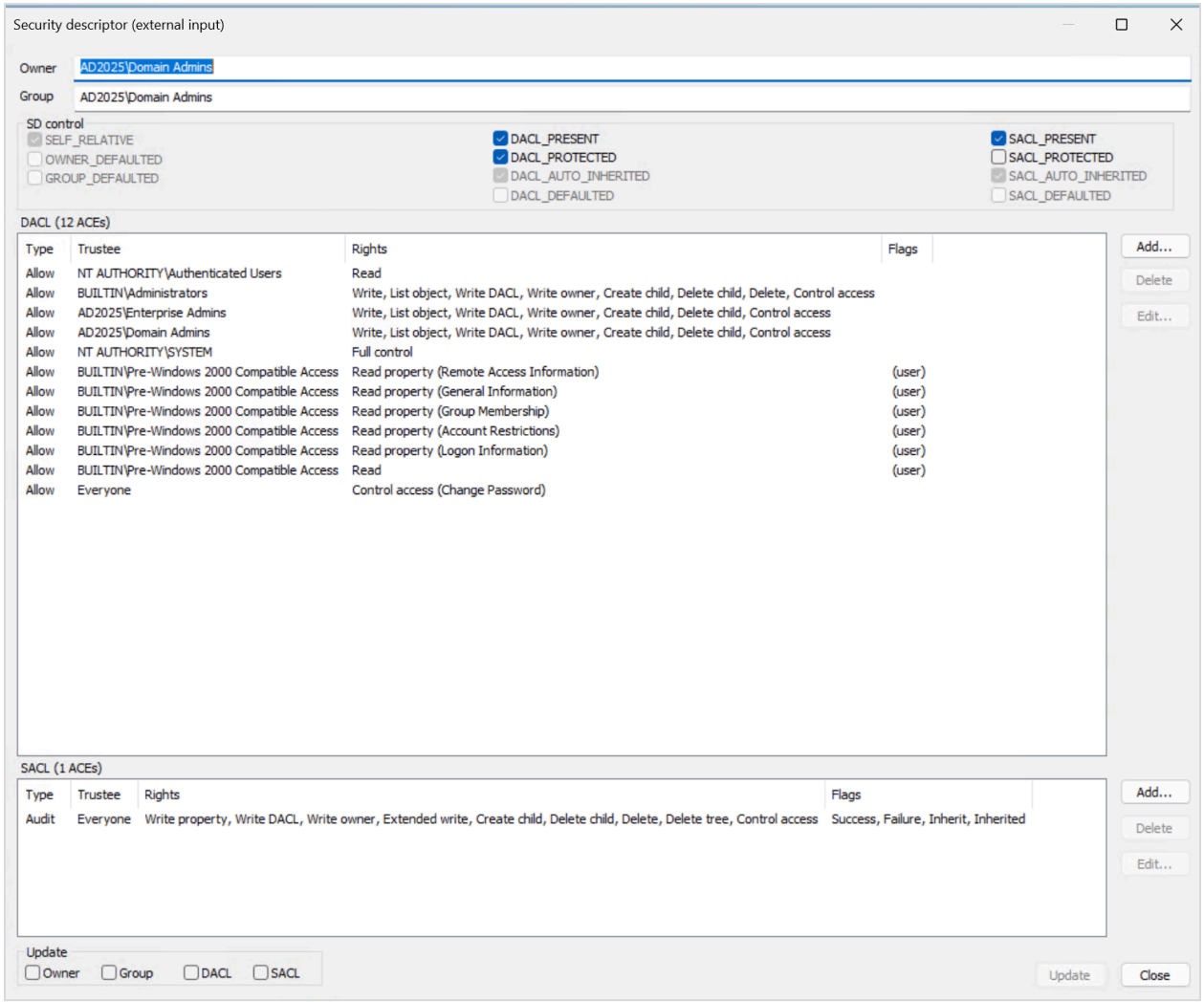


Figure 10 - AdminSDHolder Security Descriptor for AD Schema Version 13

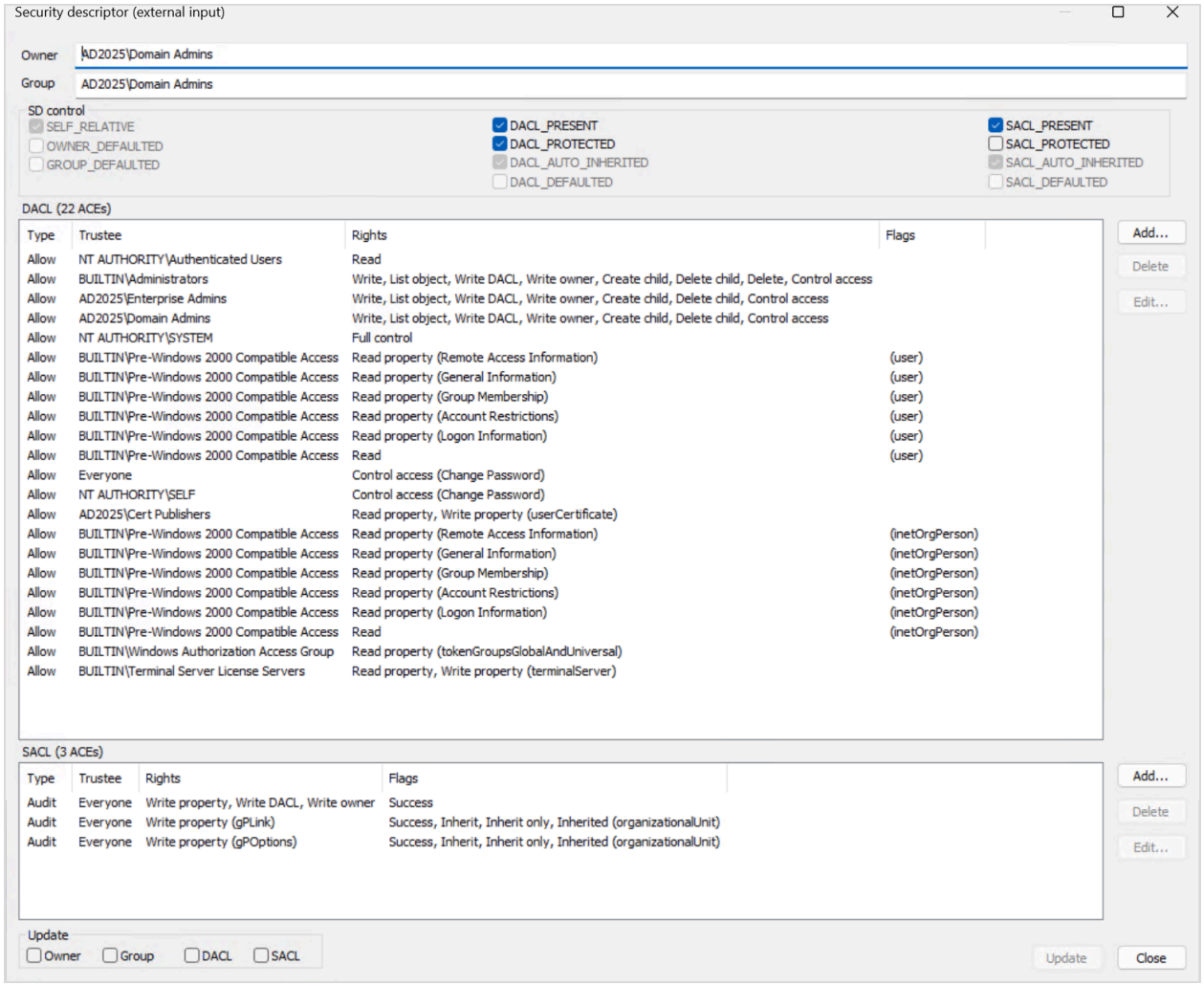


Figure 11 - AdminSDHolder Security Descriptor for AD Schema Version 30

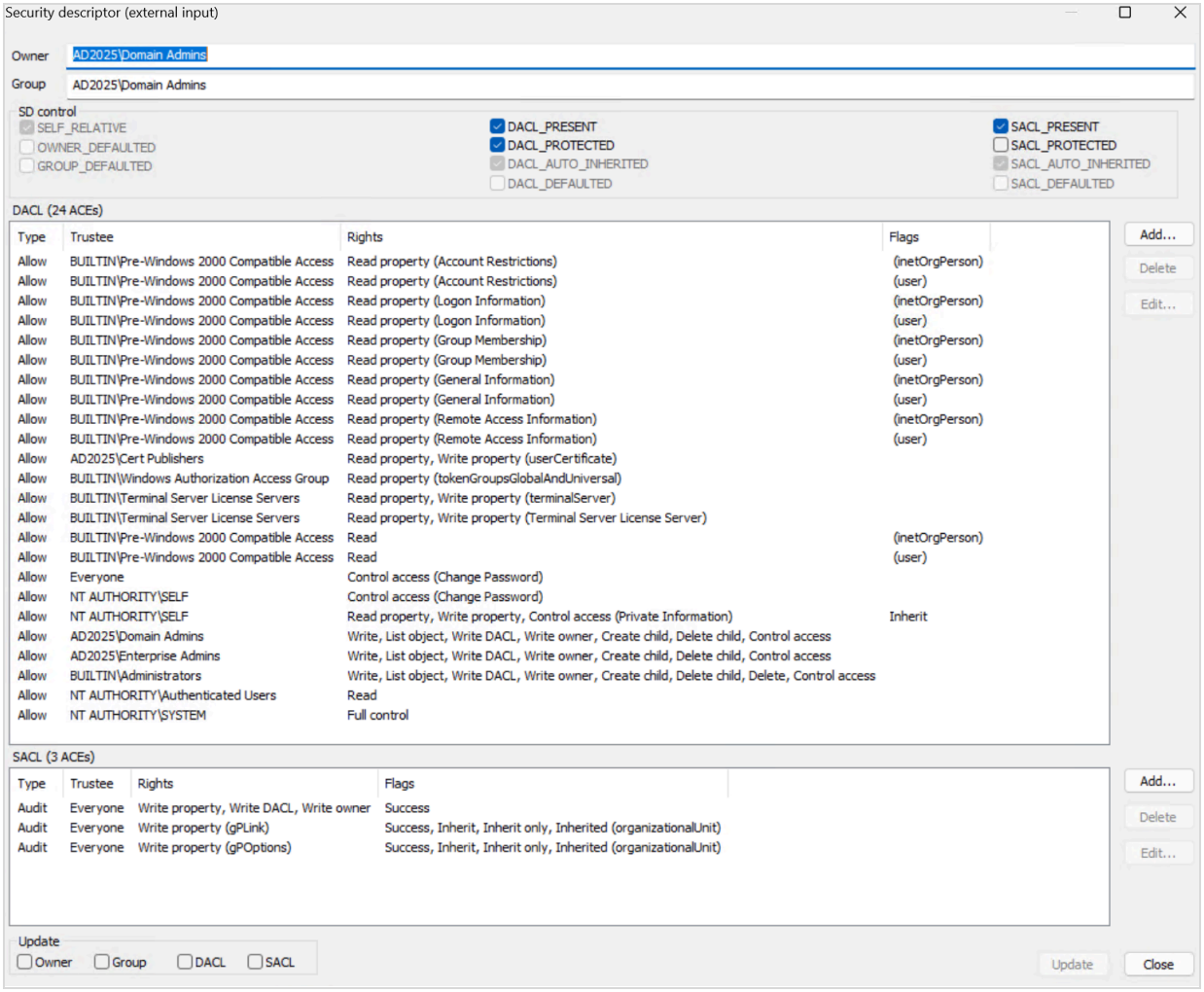


Figure 12 - AdminSDHolder Security Descriptor AD Schema Version 44

That special security descriptor is intended to protect specific sensitive and highly-privileged accounts. In early AD (Windows 2000 Server before SP4), the specific accounts to protect were those that belong to "protected groups": Domain Admins, Enterprise Admins, Schema Admins and the original Administrator account. These default "protected groups" have changed over the years, expanded from the original release to currently include as of Windows Server 2019 through Windows Server 2025: Account Operators, Administrator, Administrators, Backup Operators, Domain Admins, Domain Controllers, Enterprise Admins, Enterprise Key Admins, Key Admins, KRBTGT, Print Operators, Read-Only Domain Controllers, Replicator, Schema Admins, and Server Operators.

The associated process runs, by default, every 60 minutes on the domain controller (DC) that holds the Flexible Single Master Operations (FSMO) role of Primary DC Emulator (PDCe). The process checks the security descriptor on members of "protected groups" and compares it to the one on the AdminSDHolder object. If the security descriptor on the object to be protected is different from the security descriptor on

the AdminSDHolder object, then the security descriptor on the object to be protected is changed to that of the AdminSDHolder object and inheritance is disabled.

The AdminSDHolder object and process, hopefully, protect the most sensitive accounts in AD from [Clean Source Principal] violations via DACL abuses using these common permission sets:

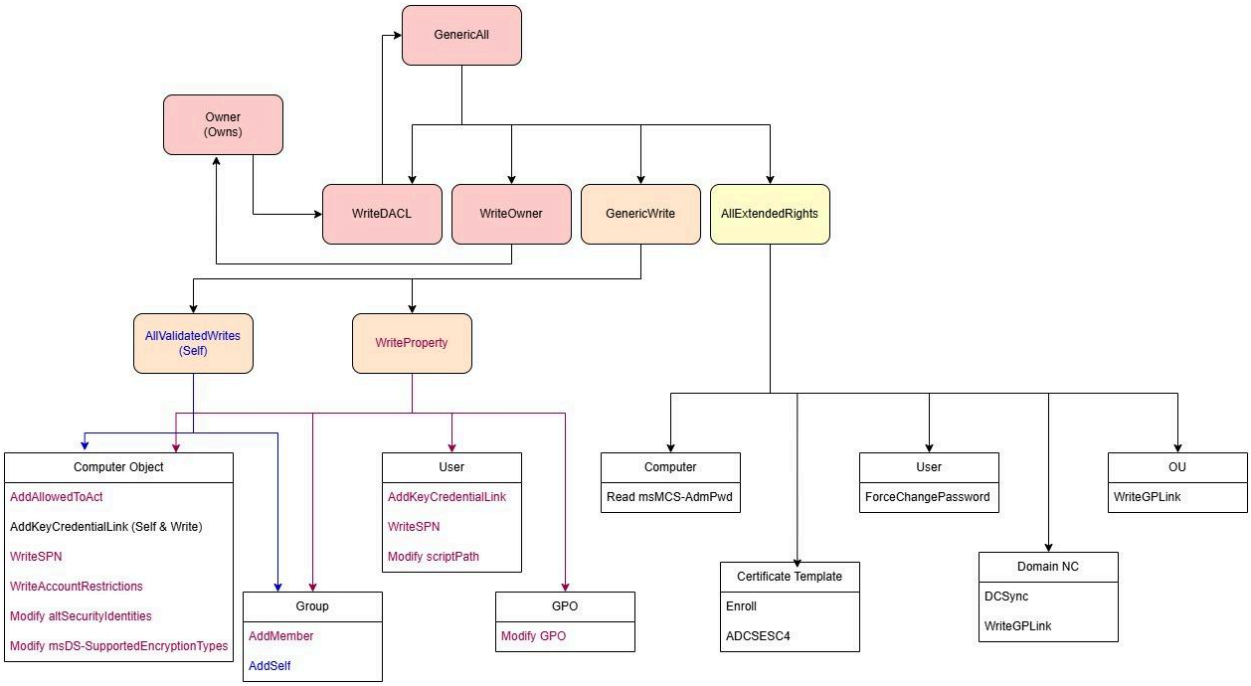


Figure 13 - "Dangerous" Permissions

TL;DR: AdminSDHolder helps prevent the Help Desk (or someone who has compromised a Help Desk user account) from becoming a Domain Admin. Maybe.

This functionality was introduced when AD was first released with Windows 2000 Server some 25+ years ago,*so it's pretty well understood now...right?*

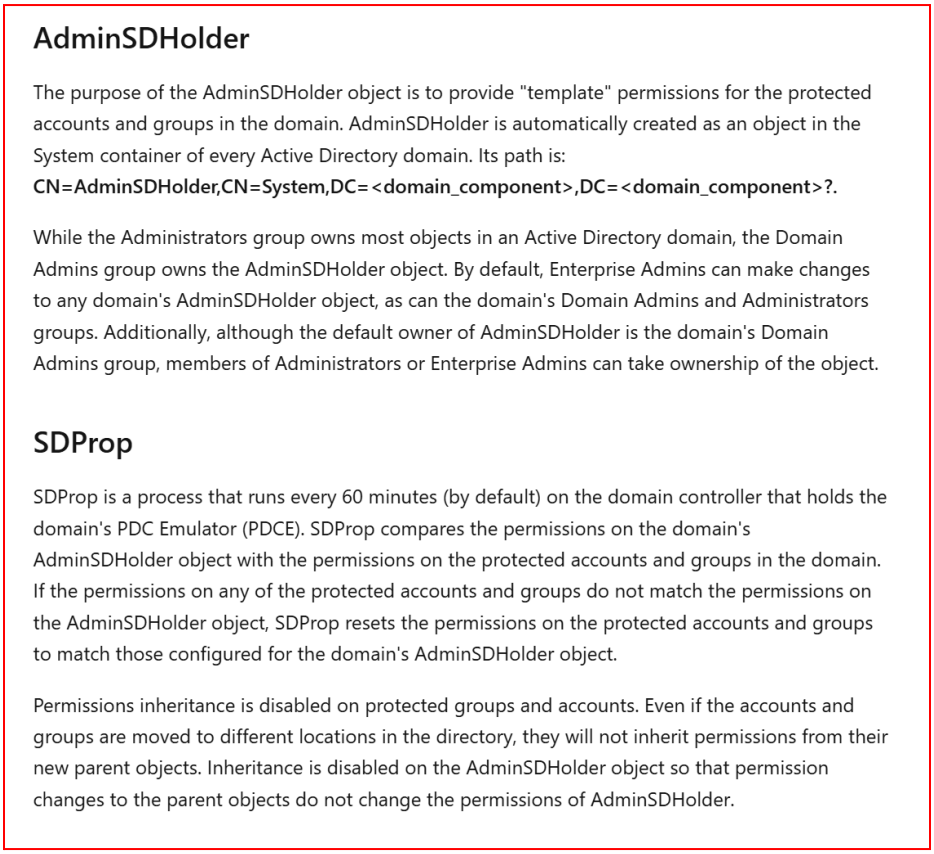
Common Misconceptions and Misconfigurations

With many of the common AdminSDHolder misconceptions, the end result is largely the same even if the technical details aren't correctly understood. Some of the other misconceptions allow for dangerous misconfigurations.

Misconception: SDProp

Did you notice that I haven't said SDProp yet? The most common and widely spread misconception around AdminSDHolder is that SDProp is somehow directly related. There are plenty of resources on the internet that have this conflated, including Microsoft's. AdminSDHolder and SDProp are not related other than that they are both Background Tasks in AD DS and both related to security descriptors.

The primary Microsoft Documentation for AD DS has an appendix section on Protected Accounts and Groups in Active Directory stating that AdminSDHolder is the object or "template" permissions for the protected security principals (which is true), but that SDProp is the process that runs and "stamps" the AdminSDHolder permissions onto the DACL of the security descriptor for the Protected Object.



AdminSDHolder

The purpose of the AdminSDHolder object is to provide "template" permissions for the protected accounts and groups in the domain. AdminSDHolder is automatically created as an object in the System container of every Active Directory domain. Its path is: **CN=AdminSDHolder,CN=System,DC=<domain_component>,DC=<domain_component>?.**

While the Administrators group owns most objects in an Active Directory domain, the Domain Admins group owns the AdminSDHolder object. By default, Enterprise Admins can make changes to any domain's AdminSDHolder object, as can the domain's Domain Admins and Administrators groups. Additionally, although the default owner of AdminSDHolder is the domain's Domain Admins group, members of Administrators or Enterprise Admins can take ownership of the object.

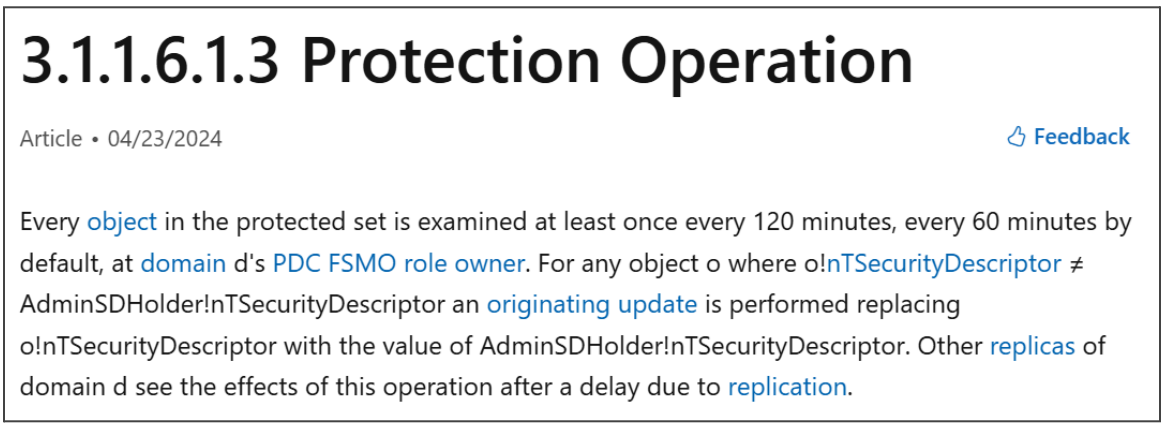
SDProp

SDProp is a process that runs every 60 minutes (by default) on the domain controller that holds the domain's PDC Emulator (PDCE). SDProp compares the permissions on the domain's AdminSDHolder object with the permissions on the protected accounts and groups in the domain. If the permissions on any of the protected accounts and groups do not match the permissions on the AdminSDHolder object, SDProp resets the permissions on the protected accounts and groups to match those configured for the domain's AdminSDHolder object.

Permissions inheritance is disabled on protected groups and accounts. Even if the accounts and groups are moved to different locations in the directory, they will not inherit permissions from their new parent objects. Inheritance is disabled on the AdminSDHolder object so that permission changes to the parent objects do not change the permissions of AdminSDHolder.

Figure 14 - Appendix C Microsoft AD-DS - March 2025 (Incorrect)

Looking back at the Microsoft Windows Protocols [MS-ADTS]: AD Technical Specifications in Section 3.1.1.6.1.3 Protection Operation, we see that the Protection Operation is in the 3.1.1.6.1 AdminSDHolder Background Task, not the SDProp Section 3.1.1.6.3 of the OpenSpecs:



3.1.1.6.1.3 Protection Operation

Article • 04/23/2024 Feedback

Every object in the protected set is examined at least once every 120 minutes, every 60 minutes by default, at domain d's PDC FSMO role owner. For any object o where o!nTSecurityDescriptor ≠ AdminSDHolder!nTSecurityDescriptor an originating update is performed replacing o!nTSecurityDescriptor with the value of AdminSDHolder!nTSecurityDescriptor. Other replicas of domain d see the effects of this operation after a delay due to replication.

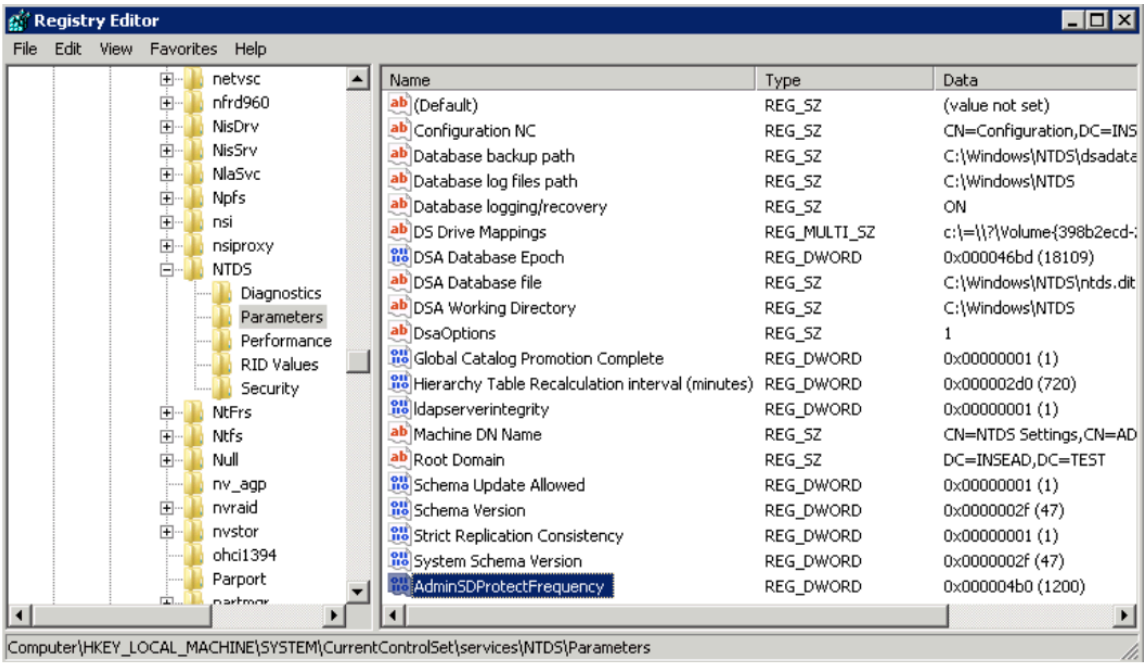
Figure 15 - MS-ADTS Protection Operation (Correct)

Here's another older article written by MVPs hosted on Microsoft TechNet that shows up when you search for AdminSDHolder. It explains some parts of AdminSDHolder really well up until it gets to the "What is Security Descriptor Propagator (SDPROP)" section. This section is sort of correct, except again it conflates the SDProp background process with the AdminSDHolder process.

What is Security Descriptor Propagator (SDPROP)?

Security Descriptor Propagator is the process used by each Active Directory domain to identify manual modifications to the owner, ACLs or inheritance applied on protected groups and replaces them with the ones defined on AdminSDHolder container. This background process runs, by default, every sixty (60) minutes on the Domain Controller holding PDC Emulator FSMO role in an Active Directory domain.

The default frequency for running Security Descriptor Propagator process could be changed by creating a **REG_DWORD** registry entry and setting the new frequency value in seconds using a Decimal value (Example: 1200 means that the frequency is twenty (20) minutes). The registry entry should be created under **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters** on the Domain Controller holding PDC Emulator FSMO role.



Remark: Lowering the default frequency value is not recommended due to potential LSASS performance issues in large Active Directory environments. However, this is not a problem for test labs.

Figure 16 - Microsoft TechNet AdminSDHolder – Screenshot March 2025 (Incorrect)

Note: Does it make sense that controlling the SDProp frequency would be done with an AdminSDProtectFrequency?

To demonstrate how widespread this misconception is, here's an incomplete list of websites that, as of June 2025, conflate AdminSDHolder and SDProp:

- Microsoft TechNet - AdminSDHolder, Protected Groups and SDPROP
- Microsoft AD-DS Appendix I - Creating Management Accounts for Protected Accounts and Groups in Active Directory
- SpecterOps - An ACE Up the Sleeve (pdf)
- ADSecurity - Sneaky Active Directory Persistence #15: Leverage AdminSDHolder & SDProp to (Re)Gain Domain Admin Rights
- DEF CON 25 - Here To Stay: Gaining Persistence by Abusing Advanced Authentication Mechanisms (pdf)
- Elastic - AdminSDHolder Backdoor
- eXploit – PowerView - A New Hope
- harmj0y - Abusing Active Directory Permissions with PowerView
- Lepide - What is an AdminSDHolder Attack and How to Defend Against it?
- Netwrix - Achieving Persistence using AdminSDHolder and SDProp
- Penetration Testing Lab - Domain Persistence – AdminSDHolder
- Petri IT Knowledgebase - Active Directory Security: Understanding the AdminSDHolder Object
- PwnDefend - Abusing AdminSDHolder to enable a Domain Backdoor
- SelfADSI - AD Permissions: The AdminSDHolder Mechanism
- Semperis - AdminSDHolder to improve Active Directory Security
- SentinelOne - Protecting Your Active Directory from AdminSDHolder Attacks
- Specops - Password Reset Understanding Privileged Accounts and the AdminSDHolder
- Tenable® - Securing Active Directory: How to Prevent the SDProp and adminSDHolder Attack - Blog
- The Hacker Recipes AdminSDHolder

Note: To anyone included on the above list, please do not take this as a slight. There's a reason this misconception is so widely spread. Great job on your SEO and I hope you might correct these documents in the future after considering what's written here.

These are all websites from trusted names in the industry, including the company that literally wrote the software, so why question it? Well, there are a few inconsistencies in the language around SDProp being run more frequently; namely the fixupInheritance part. Additionally, because I question everything, including myself.

This one conflation doesn't mean that these resources are worthless or worth less just because there's one issue on them. Quite the contrary. Everything linked so far is a great resource valuable for securing AD.

When I first dug into this issue, I didn't have a leg to stand on trying to convince folks that Microsoft itself was wrong. I had started to dig around in an AD lab with some SysInternals tools in an attempt to catch the elusive background tasks running in *LSASS.exe* on a PDCe, but then I stumbled upon Daniel Ulrichs's Secure Identity blog on AdminSDHolder Pitfalls and Misunderstandings and it all clicked together: Almost everyone got it wrong!

Note: Be sure to check out Part 2 of Daniel's blog: Where the adminCount Doesn't Count and the SD isn't What You Thought. I recommend reading all of the posts in his blog.

Daniel correctly explains that AdminSDHolder is both an object *and* a process.

All of these websites conflated some technical terms, but the end result was basically the same thing either way regardless of whether AdminSDHolder or SDProp was the background task on the PDCe, right? So who cares? *The end result, so far...*

Here are some other websites I've found beyond Daniel's that properly understand the relationship between AdminSDHolder and SDProp:

- [MS-ADTS]: Background Tasks - This one really needs to be correct as it's the actual protocol open specifications.
- KB817433 - Delegated permissions are not available and inheritance is automatically disabled - This one gets SDProp correct by not including it, but it has workarounds that are easy to misconfigure. This is also, most likely, the first document I read about AdminSDHolder back when I was still running BES.
- ITProToday: Demystifying the AdminSDHolder Object - Kudos to Tony for getting this correct back in 2007!
- Microsoft: Five common questions about AdminSdHolder and SDProp - This one is a bit of a gimme as I let Ned know about the issue and he quickly fixed it. Here's the old version of the post for posterity.

Additionally, the Designing, Deploying, and Running Active Directory book by Brian Desmond, Joe Richards, Robbie Allen, and Alistair G Lowe-Norris explains AdminSDHolder correctly and doesn't even mention SDProp once during the entire 800+ page book. Great job on the "cat book". I've got at least one copy of every edition.

I'm always interested in the history behind design choices and features. The history behind AdminSDHolder is especially interesting to me, as I've seen a lot of interesting takes on the history of this important functionality:

> The brief history of the SDProp process goes back to the early 2000s. Administrators were breaking what privileged groups could do in Active Directory when the access control list of the privileged group was changed in error. Microsoft fixed this by introducing the SDProp process, which used the adminSDHolder objects' access control list (ACL) and the adminCount attribute of both users and groups.

Figure 17 - Screenshot From tenable.ad blog 'Securing AD: How to Prevent the SDProp and adminSDHolder Attack'

The reality of why AdminSDHolder was implemented in AD, starting as early as Windows Server 2000 RC2, is explained in a comment from early Windows Server source code that explains the Theory of Operation for AdminSDHolder:

```
73      /*
74
75          Theory of Operation
76
77
78
79          NT4 and earlier releases of NT protected the users in Administrative groups by
80          changing the ACL on the member as they were added to the group. NT5 cannot adopt
81          this strategy as
82
83              1. NT5 supports nested groups ( NT4 did not have group nesting )
84              2. NT5 supports universal groups which can have members in other domain domains
85                 and could themselves be members of groups in other domains.
86
87
88          NT5 implements protection of administrative groups by a background daemon. This daemon
89          first computes the set of memberships in transitive fashion of all administrative groups.
90          It then walks the list of objects that it has and checks whether the security descriptor
91          on them is a well known protected security descriptor. If the well known protected security
92          descriptor is not set then this security descriptor is set on the object. This task is
93          executed only on the PDC FSMO holder.
94
95      --*/
```

Figure 18 - Theory of Operation - secadmin.c

Versions of Windows Server prior to Windows 2000, also known as NT5, had an entirely different security model. In Windows NT4, there was no nested group membership to account for. There were no Universal security groups, no global catalog, and little granularity in the permission model. Windows 2000 changed all of that, and so Microsoft designed a new method to protect administrative groups as a background task.

The history of why AdminSDHolder and SDProp are so commonly conflated can be traced back to an interesting discussion I had with Ned Pyle about AdminSDHolder on the social media platform formerly known as Twitter. When Ned originally wrote the Five common questions about AdminSdHolder and SDProp blog, he was the Microsoft PM for AD DS. Ned has written a lot of great blogs.

Later in the Twitter thread, I learned that this misconception exists because almost everyone, including Microsoft, followed one incorrect bit of documentation on an old SDProp KB article way back when and that incorrect bit propagated through all the Microsoft documentation and out to everyone else, becoming the authoritatively incorrect source.

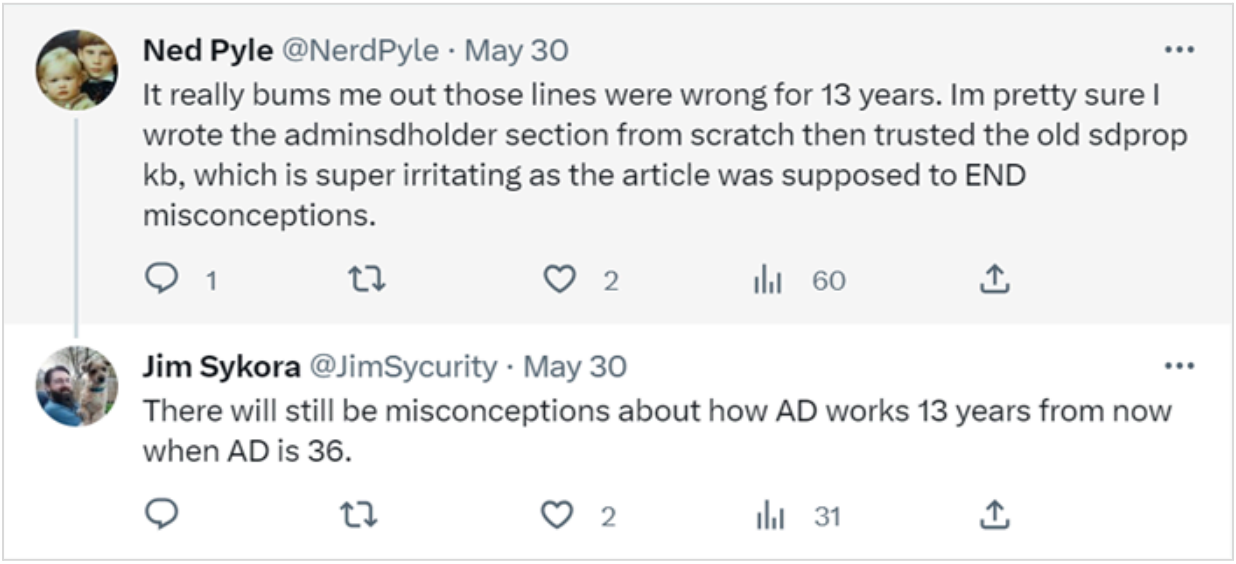
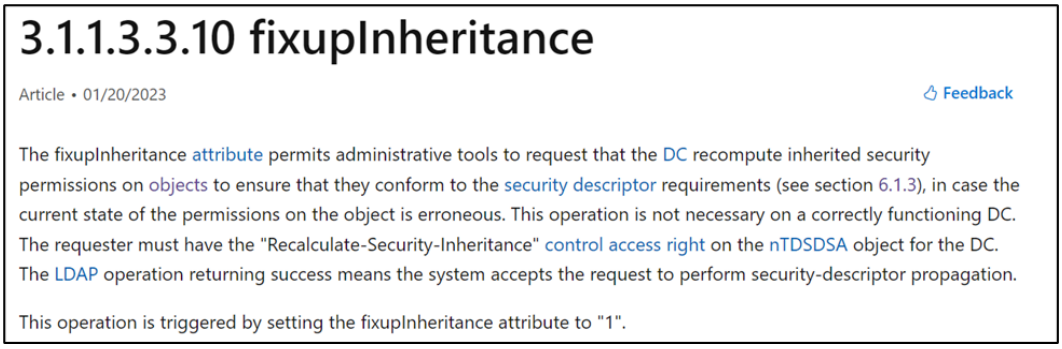


Figure19 - SDProp: The Root Cause

I probably should have noticed this all sooner reading through AD DS Appendix C where it links to the [MS-ADTS] Section 3.1.1.3.3 rootDSE Modify Operations, where I also would have noticed both fixupInheritance and runProtectAdminGroupsTask being two different operations whereby the runProtectAdminGroupsTask subsection also briefly describes how the AdminSDHolder protection operation actually works and links to [MS-ADTS] Section 3.1.1.6.1 AdminSDHolder where it's described in technical detail.



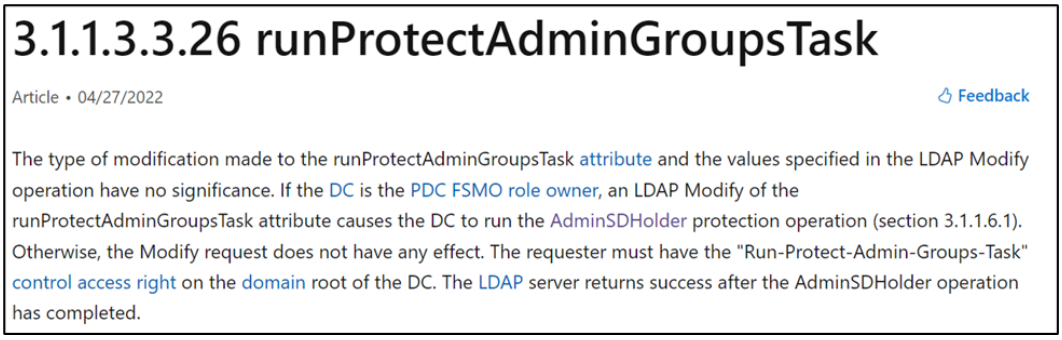
3.1.1.3.3.10 fixupInheritance

Article • 01/20/2023 Feedback

The fixupInheritance attribute permits administrative tools to request that the DC recompute inherited security permissions on objects to ensure that they conform to the security descriptor requirements (see section 6.1.3), in case the current state of the permissions on the object is erroneous. This operation is not necessary on a correctly functioning DC. The requester must have the "Recalculate-Security-Inheritance" control access right on the nTDSDSA object for the DC. The LDAP operation returning success means the system accepts the request to perform security-descriptor propagation.

This operation is triggered by setting the fixupInheritance attribute to "1".

Figure 20 - fixupInheritance: Runs on any DC



3.1.1.3.3.26 runProtectAdminGroupsTask

Article • 04/27/2022 Feedback

The type of modification made to the runProtectAdminGroupsTask attribute and the values specified in the LDAP Modify operation have no significance. If the DC is the PDC FSMO role owner, an LDAP Modify of the runProtectAdminGroupsTask attribute causes the DC to run the AdminSDHolder protection operation (section 3.1.1.6.1). Otherwise, the Modify request does not have any effect. The requester must have the "Run-Protect-Admin-Groups-Task" control access right on the domain root of the DC. The LDAP server returns success after the AdminSDHolder operation has completed.

Figure 21 - runProtectAdminGroupsTask: Only Functions on PDCe FSMO Role Holder

The 'runProtectAdminGroupsTask' is for AdminSDHolder and 'fixupInheritance' is for SDProp.

The best description I can come up with for AdminSDHolder is that AdminSDHolder is both an AD Object in the system partition of AD **and** a Background Task that runs on the PDCe every 60 minutes, by default, that stamps the stringent security descriptor of the AdminSDHolder object on security principals with elevated rights in AD to protect them from straightforward DACL-based attacks.

Recommendation: None. I'm being more pedantic than didactic about this particular misconception other than that information flows quickly and broadly from trusted sources, even when it's incorrect.

If we want to be especially pedantic, the background task for AdminSDHolder should probably be referred to as "ProtectAdminGroups" as that's the name of the task and primary AdminSDHolder function in the Windows Server source code.

If you're someone from Microsoft reading this, I recommend an update AD DS Appendix C.

Note: Before I inadvertently create a misconception, it is true that SDProp will run **if** AdminSDHolder changes a security descriptor, but SDProp doesn't do anything to protect the object in this instance. If you want to know what SDProp actually does, read the MS-ATDS links or skip ahead to **What SDProp Actually Does**.

Bonus: Any bets on how long AI will spit out incorrect information on this?

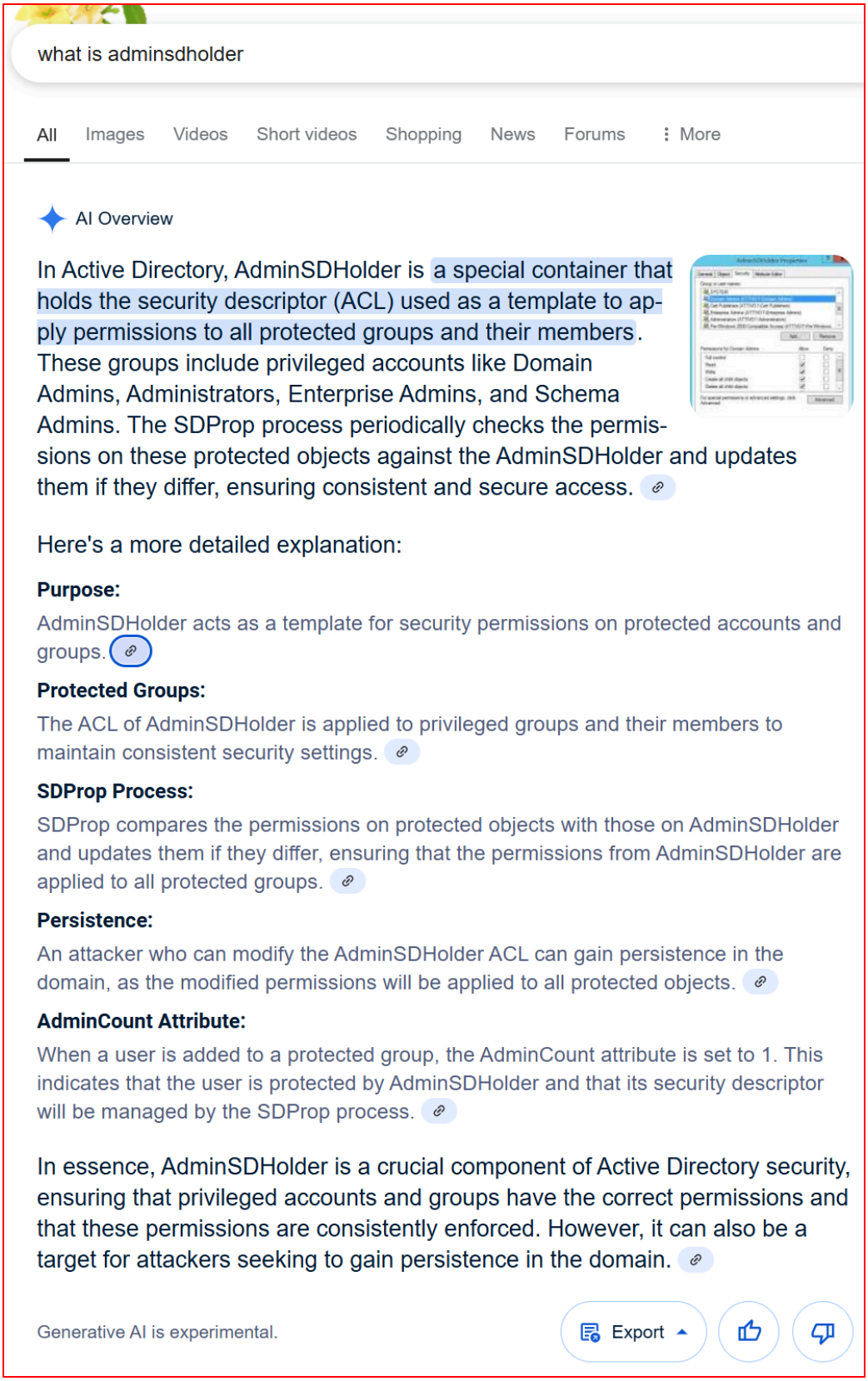


Figure 22 - Screenshot of Generative AI in Google Search Answering "What is AdminSDHolder"

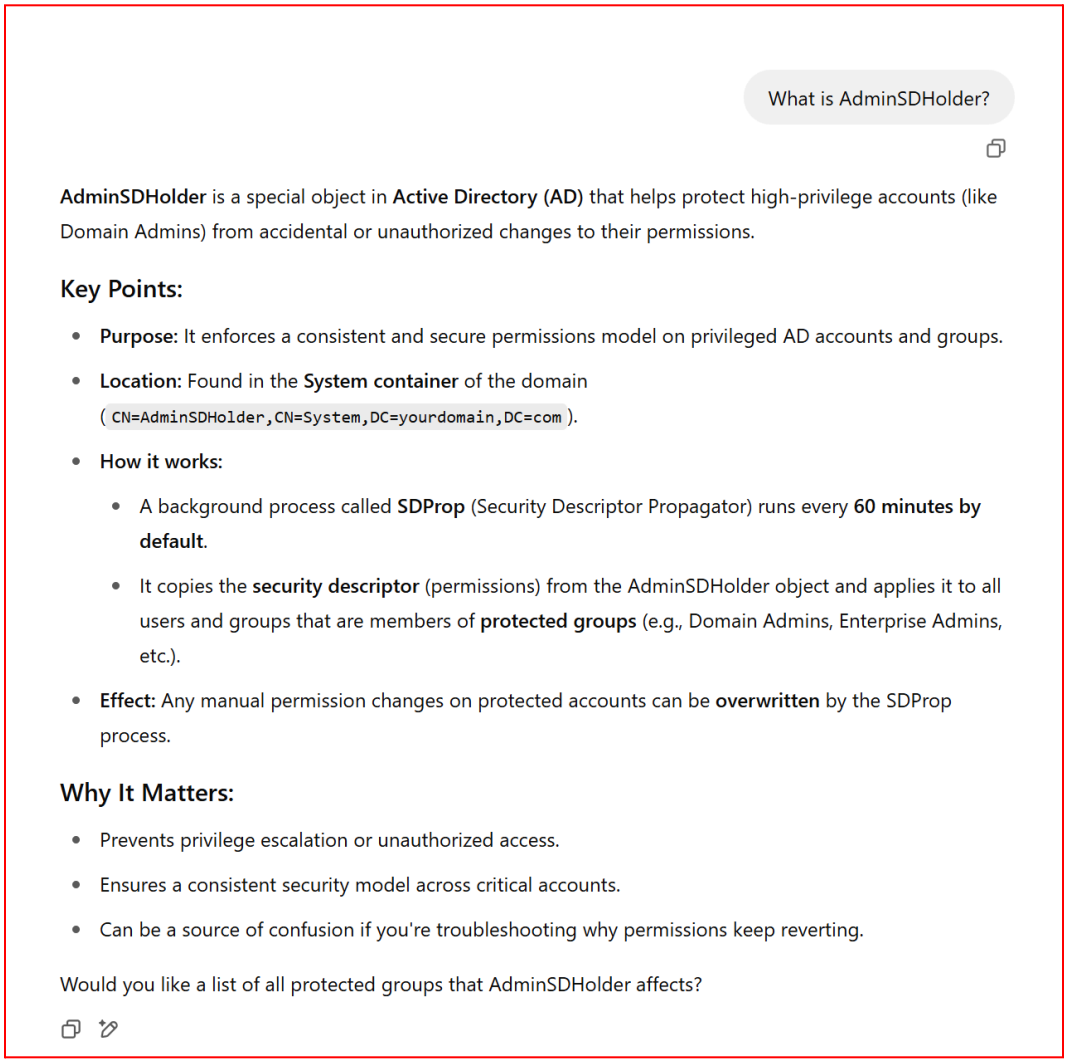


Figure 23 - Screenshot of ChatGPT Answering "What is AdminSDHolder?"

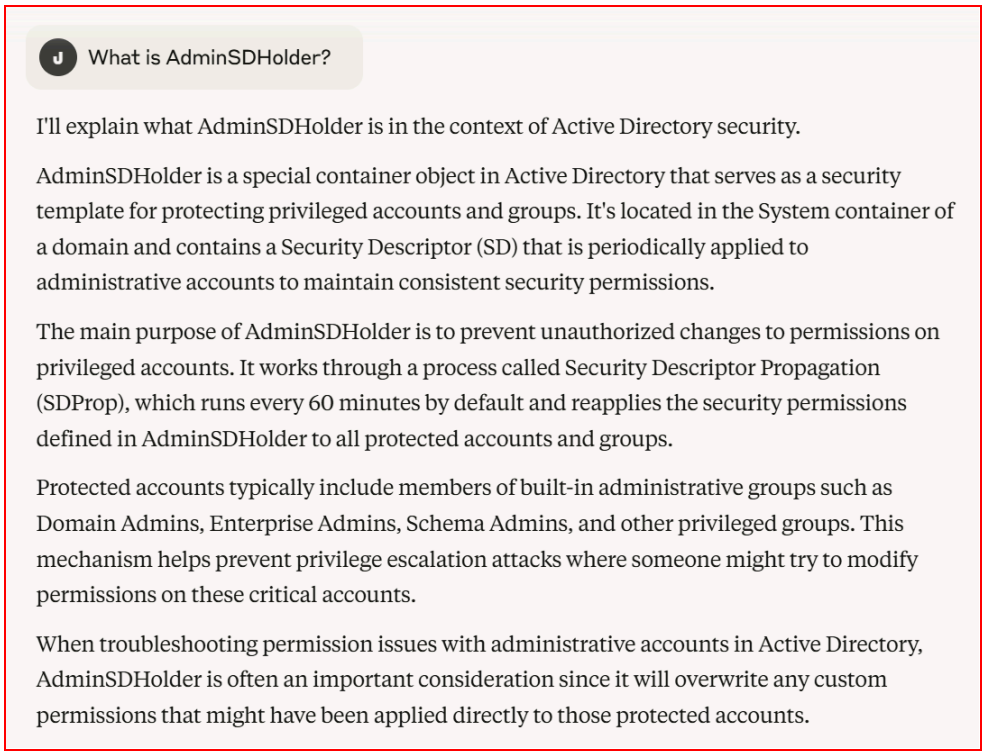


Figure 24 - Screenshot of Claude.ai Answering "What is AdminSDHolder?"

My bet: It's going to take some time to revert this misconception.

Misconception: Changing the SDProp Interval

AD DS Appendix C, and many other websites, state that the SDProp interval can be changed from the default of 60 minutes. This is incorrect and based on the first misconception. The actual SDProp background task does not have a fixed schedule; SDProp runs when it is triggered to run, not on any sort of set timeline.

It's the AdminSDHolder ProtectAdminGroups background task that runs every 60 minutes, by default, on the PDC Emulator FSMO role holder. If you modify the AdminsDProtectFrequency registry value on the PDCe for the domain, you will change the interval for AdminSDHolder.

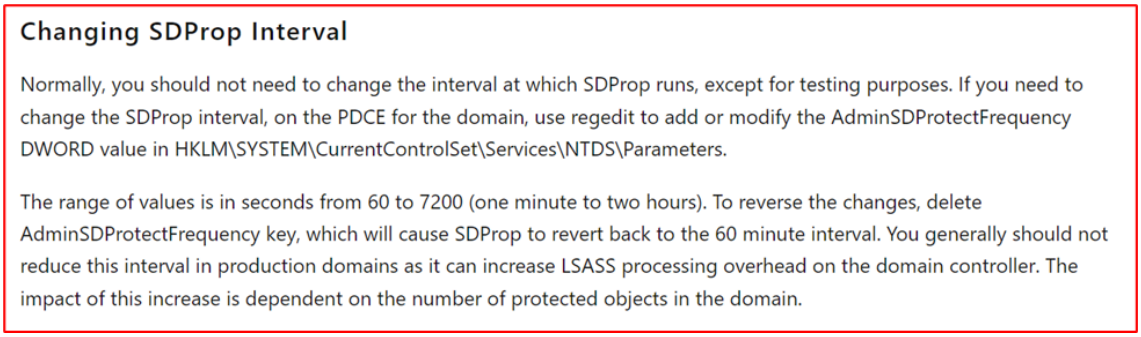


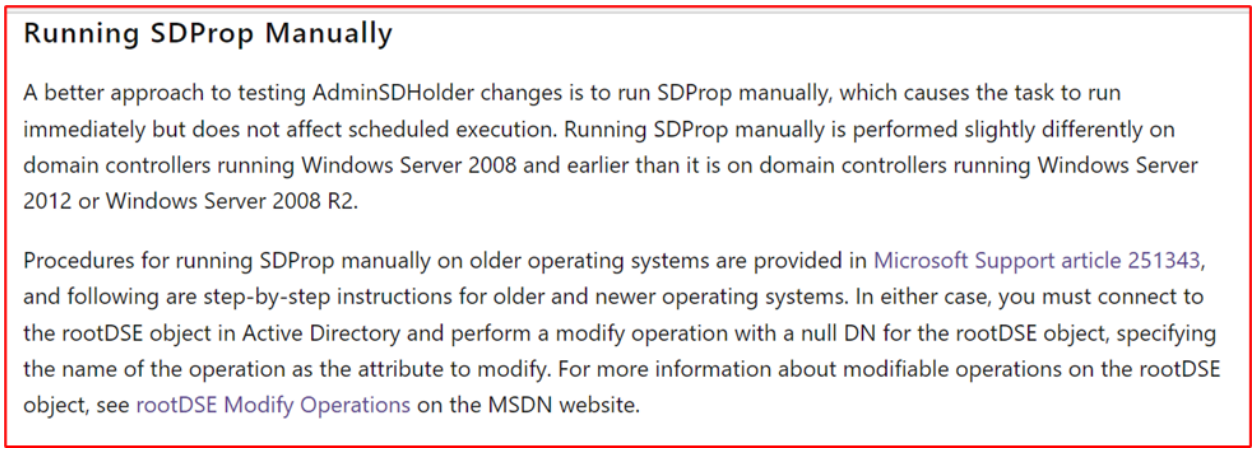
Figure 25 - Changing SDProp Interval

Recommendation: There is nothing you can do to change the interval SDProp runs at because SDProp doesn't run at an interval. For more information on that, see What SDProp Actually Does. If you want to change the default interval that AdminSDHolder ProtectAdminGroups runs at, this is possible, but not recommended. For information on that, see **Misconfiguration: Change AdminSDHolder Interval**.

Misconception: Running Manually

Most guidance on AdminSDHolder mentions that you can run (SDProp) manually. It is both true and correct that you can run SDProp manually and that running SDProp manually will have no effect on protected objects or AdminSDHolder because SDProp has no material effect on AdminSDHolder or protected objects.

Most of the websites above in **Misconception: SDProp** have some mention of running SDProp manually. The Microsoft AD DS Appendix C on Protected Accounts and Groups in AD does:



Running SDProp Manually

A better approach to testing AdminSDHolder changes is to run SDProp manually, which causes the task to run immediately but does not affect scheduled execution. Running SDProp manually is performed slightly differently on domain controllers running Windows Server 2008 and earlier than it is on domain controllers running Windows Server 2012 or Windows Server 2008 R2.

Procedures for running SDProp manually on older operating systems are provided in Microsoft Support article 251343, and following are step-by-step instructions for older and newer operating systems. In either case, you must connect to the rootDSE object in Active Directory and perform a modify operation with a null DN for the rootDSE object, specifying the name of the operation as the attribute to modify. For more information about modifiable operations on the rootDSE object, see rootDSE Modify Operations on the MSDN website.

Figure 26 - Running SDProp Manually

If you create an LDAP operation for fixupInheritance, you will be manually triggering SDProp to run. You can do this on any DC and there's almost no reason to do so unless your AD domain is quite messed up.

The runProtectAdminGroupsTask LDAP operation is the correct method for manually triggering the ProtectAdminGroups background task for AdminSDHolder.

Both of these are rootDSE modify operations. Every version on Windows Server OS includes the fixupInheritance operation across all flavors of AD. The runProtectAdminGroupsTask operation was added as a rootDSE modify operation in Windows Server 2008 R2 and is only available for AD DS.

Both operations require specific control access right permission grants on the appropriate AD objects. FixupInheritance requires Recalculate-Security-Inheritance extended right on the nTDSDSA object associated with the DC the trustee wishes to run it on. This object will be located in the Configuration partition for the AD forest in the Sites subtree. Enterprise Admins for the forest, Domain Admins for the root domain in the forest, and Domain Admins for the domain the DC corresponds to will have this

extended right by default. The runProtectAdminGroupsTask operation requires that the requester be granted the Run-Protect-Admin-Groups-Task control access right on the domain NC root where the PDCe FSMO role holder is located. By default, Administrators, Domain Admins, and Enterprise Admins will be granted this extended right. It is more likely that the permissions at the root of the domain are non-default than that of the nTDSDSA objects.

Recommendation: In order to trigger AdminSDHolder to run manually, the runProtectAdminGroupsTask attribute must be set to "1" via LDP or some other LDAP operation on the DC that currently holds the PDC Emulator FSMO role. The requester must have the Run-Protect-Admin-Groups-Task control access right on the domain NC for that DC.

The most straightforward way I've found to manually invoke the runProtectAdminGroupsTask is via PowerShell:

```
$PDC =
([System.DirectoryServices.ActiveDirectory.Domain]::GetComputerDomain()).
PdcRoleOwner.Name

$PDCERootDSE = [ADSI]"LDAP://$PDC/RootDSE"

$PDCERootDSE.Put('runProtectAdminGroupsTask','1')

$PDCERootDSE.SetInfo()
```

It is generally only necessary to manually trigger AdminSDHolder in lab environments when doing testing, or perhaps during an incident response exercise in an effort to eject a threat actor who has managed to gain persistence via AdminSDHolder? If your incident response (IR) playbook has you double-tapping krbtgt in specific scenarios, you may also want to consider manually triggering AdminSDHolder to run as well.

Misconception: Just Change AdminSDHolder's DACL

We've gone back to the year 2007 and are observing a SysAdmin attempting to integrate a BES instance with Microsoft Exchange on-premises. It's all going surprisingly well, considering how challenging this software was, but the SysAdmin isn't getting email on their BlackBerry even after delegating the BESAdmin service account Send-As rights at the Domain Root (**Note**: Avoid delegating permissions at the Domain Root when specific OUs in a well-designed OU structure would be least-privilege). After hours of banging their head against the wall, the SysAdmin finds something promising while searching Google: KB817433 - Delegated permissions are not available and inheritance is automatically disabled.

Or maybe it was one of the results that still show up on delegating BESAdmin rights on the AdminSDHolder object instead of following the best practice of not mailbox-enabling privileged accounts? DSACLS without understanding how the underlying technology works or why: what could go wrong? A Microsoft KB article encouraging modifying the DACL of AdminSDHolder?

> How can I add the BES user account to these two users without AdminSDHolder taking it back away due to the send as permission? They are currently unable to send emails due to this.
>
> Obviously one solution is to take them out of the admin group, but that is not an acceptable solution.

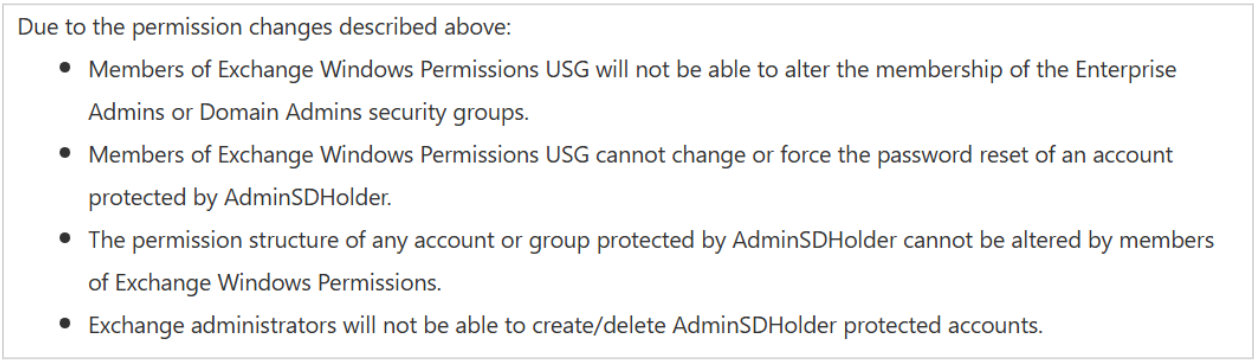
> Okay, I am trying to get my Blackberry working with my domain admin account (yes I know, email with an admin account is not the best idea).

> Is there a solution to this problem which dosen't need the users removing from domain admins or the adminSDholder to be disabled?

> In SBS, if the SBS user was created using the Power User or Administrator user templates, then they will have this issue where the SEND AS will not stay on their Active Directory security.
>
> To be able to get these users to properly keep the Send As permissions that they need to work with BlackBerry without removing them from any groups, we can use the following command to change the default permissions for the AdminSDHolder object:
>
> dsacls "cn=adminsdholder,cn=system,dc=mydomain,dc=com" /G "MYDOMAIN\BlackBerrySA:CA;Send As"

Of all the changes that could possibly be made to the DACL of the AdminSDHolder object, adding a very narrowly defined, least-privilege SendAs ACE that only applies to Users isn't the worst thing that could happen; however, from years of experience, I've noted that well intentioned permission delegations often end up not being least-privileged at all.

If your organization has, or has ever had, Microsoft Exchange integrated into your on-prem AD environment, it's likely that the /PrepareDomain function of the Exchange installer has made at least some modifications to the AdminSDHolder object's DACL. If you're one of the unfortunate organizations that installed the Exchange 2010 Release Candidate (RC), I hope you're aware of the security issue it created by modifying the AdminSDHolder DACL to allow Exchange Windows Permissions USG to change the membership, force password resets, or modify the permissions of any protected object. Seriously, check the document out even if you never used Exchange 2010. It's a reminder that nobody is infallible.


Due to the permission changes described above:

- Members of Exchange Windows Permissions USG will not be able to alter the membership of the Enterprise Admins or Domain Admins security groups.
- Members of Exchange Windows Permissions USG cannot change or force the password reset of an account protected by AdminSDHolder.
- The permission structure of any account or group protected by AdminSDHolder cannot be altered by members of Exchange Windows Permissions.
- Exchange administrators will not be able to create/delete AdminSDHolder protected accounts.

Figure 27 - Release Candidate Issues Resolved in the Exchange 2010 RTM Installer

If you go back far enough in history, Microsoft Windows has even mucked up the DACL of the AdminSDHolder*. Windows 2000 RC3 made changes to the AdminSDHolder object and removed the ability for protected accounts to change their own password. KB232199, which is archived, goes into more detail regarding this vulnerability.

When might be a good time to modify the AdminSDHolder DACL? Maybe if you're implementing a Privileged Identity Management (PIM) solution and want to restrict the PIM service account so that it uses least-privilege access instead of the insecure and unfortunately common circumstances of adding a PIM or Privileged Access Management (PAM) service account to Domain Admins. Appendix I of the AD-DS documents has a writeup on how to configure a PIM solution or roll-your-own just-in-time (JIT) solution with least-privilege access.

Recommendation: Leave AdminSDHolder permissions default. Any permissions that grant Allow Full Control (GenericAll), Write (Modify) or All-ExtendedRights to any security principal that isn't Administrators, Domain Admins, Enterprise Admins, or SYSTEM should be carefully scrutinized. This is not a full list of dangerous permissions that could be applied.

If you've ever used Microsoft Exchange in your on-prem AD environment, it's important to understand that it is not best practice or secure to mail enable highly privileged accounts. A person's standard user account which is used for email, web browsing, and productivity apps should not have administrative privileges. Likewise, an administrative account should not be used for email, web browsing, etc. Therefore, it is prudent to ensure all administrative accounts are not mail enabled and then remove ACEs that were created for the Exchange Permissions Model or Exchange RBAC Permission Model on the AdminSDHolder object.

If your organization believes security-by-obscurity is a worthwhile pursuit, it can be effective to remove the Read ACEs granted to Authenticated Users and Pre-Windows 2000 Compatible Access from the AdminSDHolder DACL. However, if you do so, please remember to replace those ACEs with a new GenericRead grant to a specific domain local security group for the purposes of audit and security tooling that may need access to read these privileged objects. Granting read rights to a security group specific to an audit role is important any time read permissions are removed from unprivileged principals on any object. Without this, audit and security tools your organization uses may give incorrect results. The Principle of Open Design states that a system's security should not rely on the secrecy of its design or implementation. Security-by-obscurity contradicts the Principle of Open Design.

Bonus: Each AD domain has its own AdminSDHolder. In a multi-domain forest, the parent domain and each child or tree domain will have their own AdminSDHolder object, potentially with their own security descriptor. Changing the DACL of a child domain's AdminSDHolder could have disastrous results to the security of that AD Forest. We'll get to that in the next misconception.

Misconception: adminCount

This misconception has a few parts and it is one of the least understood aspects of AdminSDHolder.

Orphaned AdminSDHolder Objects?

The first common misconception around adminCount is that it's a good indicator of who the highly privileged users are in an AD domain or forest. It can be, but it's more accurate to think of accounts with the adminCount attribute set to 1 as accounts that are **or have been** privileged accounts that AdminSDHolder specifically protects. There may be users that are no longer privileged who still have adminCount = 1. There are very likely privileged security principals that do not have adminCount = 1.

How can an account without privileges have an adminCount of 1? When a user is removed from a "protected group", the adminCount is not reset. There is no inverse of the AdminSDHolder background task that un-stamps security descriptors and resets adminCount. These "orphaned AdminSDHolder objects" no longer receive their DACL from the AdminSDHolder object, but it also doesn't get inheritance turned back on. It's in limbo.

These orphaned AdminSDHolder objects must be handled manually. While some guides exist showing how to re-enable inheritance and reset the adminCount, up to and including an old VB Script Microsoft published, the best practice here is to disable or decommission orphaned AdminSDHolder objects and

create a new unprivileged account. The reason is that whenever the account was privileged (AdminSDHolder stamped or tattooed it at some point) it may have been granted, delegated, or taken additional explicit permissions. Even though the privileged group membership no longer exists, a former Account Operators member could still be the Owner of many user and computer objects. It may still have an explicit delegation on OUs, Containers, Objects, or even the Domain Root. It could have individual ACE on privileged file shares or NTFS volumes.

ONCE AN ACCOUNT IS PRIVILEGED, IT SHOULD ALWAYS BE TREATED AS PRIVILEGED.

AdminCount is the Trigger?

Another adminCount misconception is that the AdminSDHolder security descriptor is applied to AD Objects with an adminCount of 1. It's the other way around. When the AdminSDHolder background task stamps the security descriptor of the object, it then sets the adminCount.

I've also heard it said that earlier versions of AD used to set the adminCount to 1, and then based on the adminCount of 1 apply the AdminSDHolder security descriptor. This also appears to be false, at least based on experimentation with all production versions of Windows combined with code review where source code is publicly available.

My lab has AD DCs going all the way back to Windows Server 2000 Release to Manufacturer (RTM). In every environment, I created a user named "adminCount" and set the adminCount attribute of the adminCount user to 1. I then waited plenty of time for the AdminSDHolder background task to run (or manually started it) and reviewed the user accounts. All maintained their adminCount of 1, but none were stamped with the AdminSDHolder security descriptor. Manually changing the adminCount is yet another way you could have an unprivileged account with an adminCount of 1, although I can't think of any benefit to doing so.

Some blogs, even from Microsoft state that prior to Windows 2003, relying on adminCount was an optimization, but that 2003 and onwards no longer use it. My experimentation shows that it hasn't applied since at least Windows Server 2000 RTM. AdminCount wasn't the trigger in Windows Server 2000 RC3 either, but it could have applied in earlier beta and release candidates.

Note: The earlier beta and release candidates prior to RC3 of Windows 2000 require an older machine type which is not available anymore in modern virtualization platforms, although it may be possible to get these running on some legacy x86 emulators.

Question: *How does the AdminSdHolder operation determine whether or not to ACL an account?*

Answer: It is based on transitively expanding the list of (possibly nested) protected groups. The attribute AdminCount was originally used only as an optimization to improve performance, since it was assumed that regardless of group membership, AdminCount being 1 should trigger protection. However from repro's on Windows Server 2003 and source code review, it appears this is no longer enough to actually trigger the AdminSdHolder operation all on its own.

When a Security Principal is a member of a protected group its Security Descriptor is stamped with the SD of the AdminSDHolder Object for that domain. Also the Security Principal's adminCount attribute is set to value 1. If the SD of the security principal in question already matches the SD of the AdminSDHolder Object, the object is left untouched. Consequently its adminCount value could potentially remain 0. So using AdminCount is a pure mark of whether or not a user is protected is not always a good idea – the group membership is the key.

Figure 28 - https://learn.microsoft.com/en-gb/archive/blogs/askds/five-common-questions-about-adminsdholder-and-sdprop

Figure 3. SDProp process updates the ACL on all users and groups with adminCount = 1.

Figure 29 - https://www.tenable.com/blog/securing-active-directory-how-to-prevent-the-sdprop-and-adminsdholder-attack

This screenshot from my lab shows a user named ‘Admin Count’, which has had an adminCount of 1 for several weeks now:

Figure 30 - Screenshot of the Admin Count User with adminCount of 1

And here we have another screenshot from the same lab showing that the security descriptor is not that of AdminSDHolder, even though the AdminSDHolder ProtectAdminGroups background task has run approximately 555 times since the adminCount was set to 1:

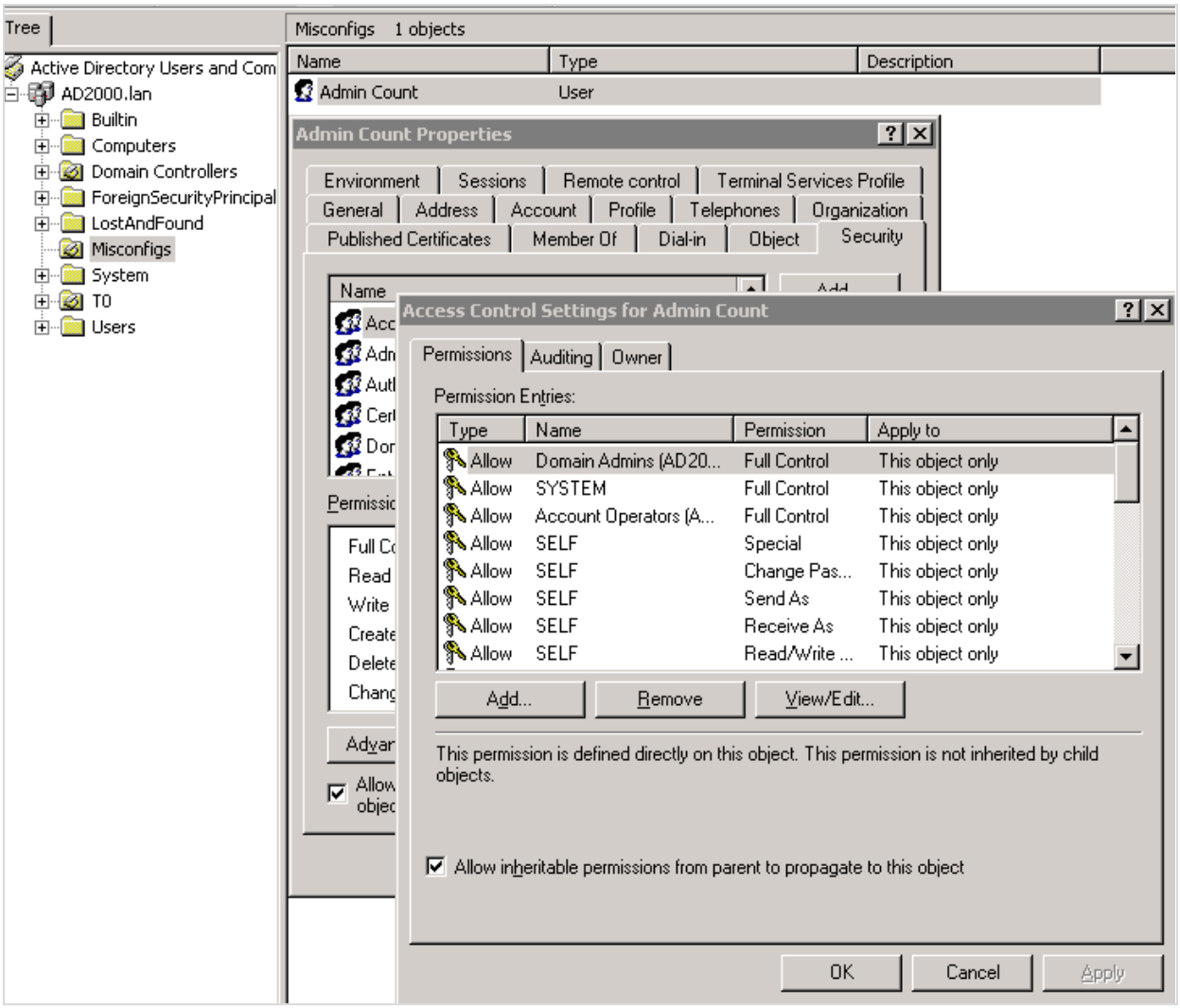


Figure 31 - Screenshot of Admin Count's Security Descriptor

This is further supported by the data in my GitHub across all of the Test-AdminSDHolderPostTest.csv files in the OS Versions folder and the 'Admin Count user.txt' files in the Windows Server 2000 RC3 and Windows Server 2000 RTM folders.

ADMINCOUNT IS NOT THE TRIGGER FOR ADMINSDHOLDER.

AdminCount Counts

If a security principal has a security descriptor that is identical to the AdminSDHolder security descriptor, then the AdminSDHolder Background Task will not modify it. So, another sneaky AD Persistence trick might be to create or compromise a standard user account. Modify the compromised user's security descriptor to exactly match that of AdminSDHolder, including Ownership, DACL, and SACL, then add it to a highly privileged "protected" group; however, this won't get an adminCount of 1 because the security descriptor is already the same as AdminSDHolder. I proved this out in my lab and kept a transcript on my GitHub.

There is a much simpler, post-compromise, method to acquire an administrative account without an adminCount:

1. Acquire an admin account.
2. Clear the adminCount property.

As long as the security descriptor of the acquired account does not change from that of the AdminSDHolder object, the ProtectAdminGroups task will not set the adminCount to 1. I've proved this out in my lab and kept a transcript of it on my GitHub. I even created a scenario in Step 3 of the test where the only account with an adminCount of 1 is the CN=adminCount Test,OU=AdminSDHolderTests,DC=AD2016CU,DC=lan, which has no privileged rights. Every account and group that is highly privileged and still protected by AdminSDHolder has no adminCount.

Note: When inheritable ACEs, whether as part of the DACL or SACL, propagate to a protected object neither of the 'tricks' here will work because the security descriptor is consistently different.

ADMINCOUNT IS NOT A RELIABLE METHOD TO DETERMINE PRIVILEGED STATUS.

AdminCount and (Domain) Trusts

Daniel Ulrichs points out the third misconception about adminCount in the second post of his AdminSDHolder series: adminCount and Trusts

Within an AD Forest, which is the security boundary of AD, when a security principal is added to a Protected Object which is a Universal Security Group (USG) like Enterprise Admins or Schema Admins, the security principal will receive a stamp of the AdminSDHolder from the domain the object resides in. If the object resides in a child domain, it receives the AdminSDHolder of the child domain. If the AdminSDHolder DACL is more permissive than the AdminSDHolder at the forest root domain, that new Enterprise Admin from the child domain still gets the weaker child domain AdminSDHolder DACL. This is especially important to consider in forests where Microsoft Exchange has ever been installed.

THE ADMINSDHOLDER PROCESS CAN ONLY PROTECT OBJECTS IN ITS OWN DOMAIN.

Here is a USG example from my home lab which demonstrates this fact. Within the *magic.lab.lan* forest, there are multiple child domains including *foolus.magic.lab.lan*. First, I'm showing the (awful) security descriptors of both the root domain in the forest (i.e., *magic.lab.lan*) and of the child domain (i.e., *foolus.magic.lab.lan*). Based upon both the Owner and ACEs in the DACL, it's easy enough to differentiate the two,

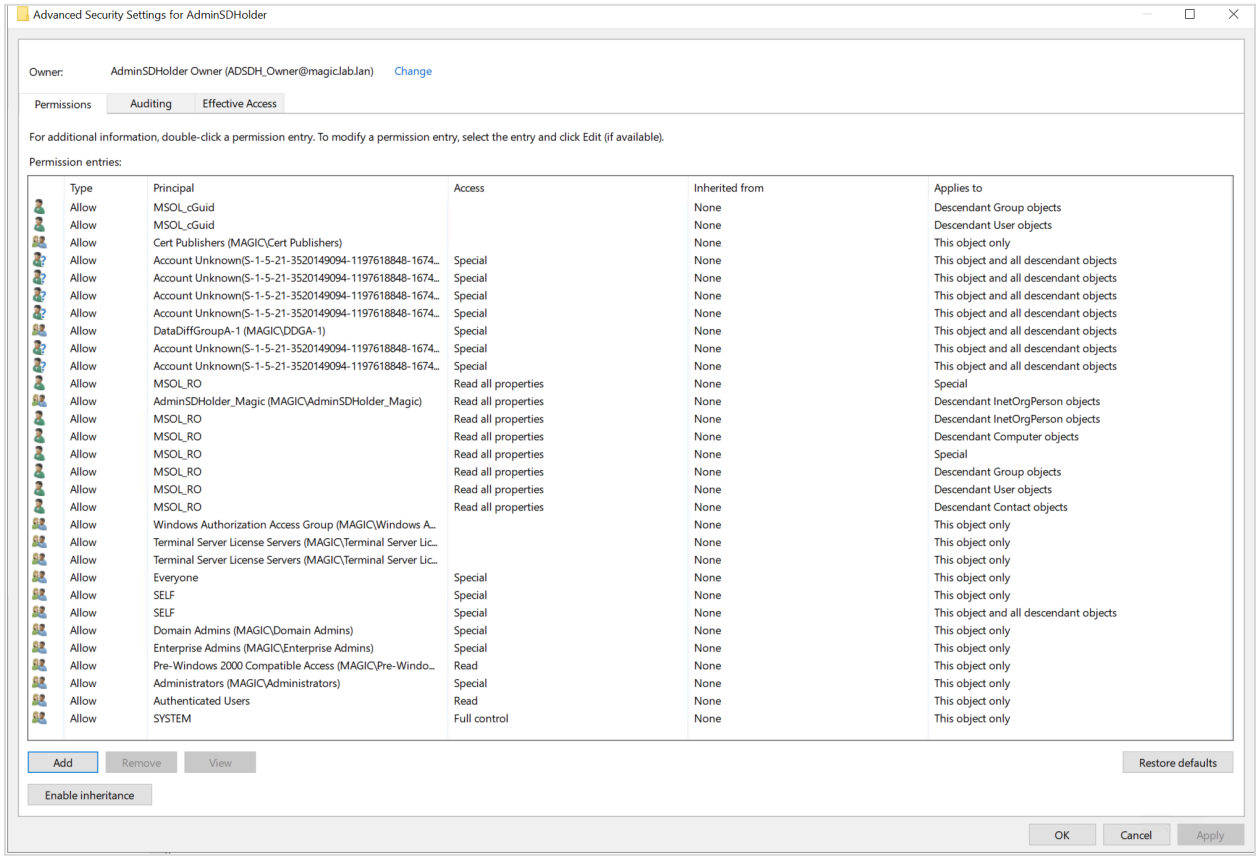


Figure 32 - Forest Root Domain AdminSDHolder Security Descriptor - magic.lab.lan

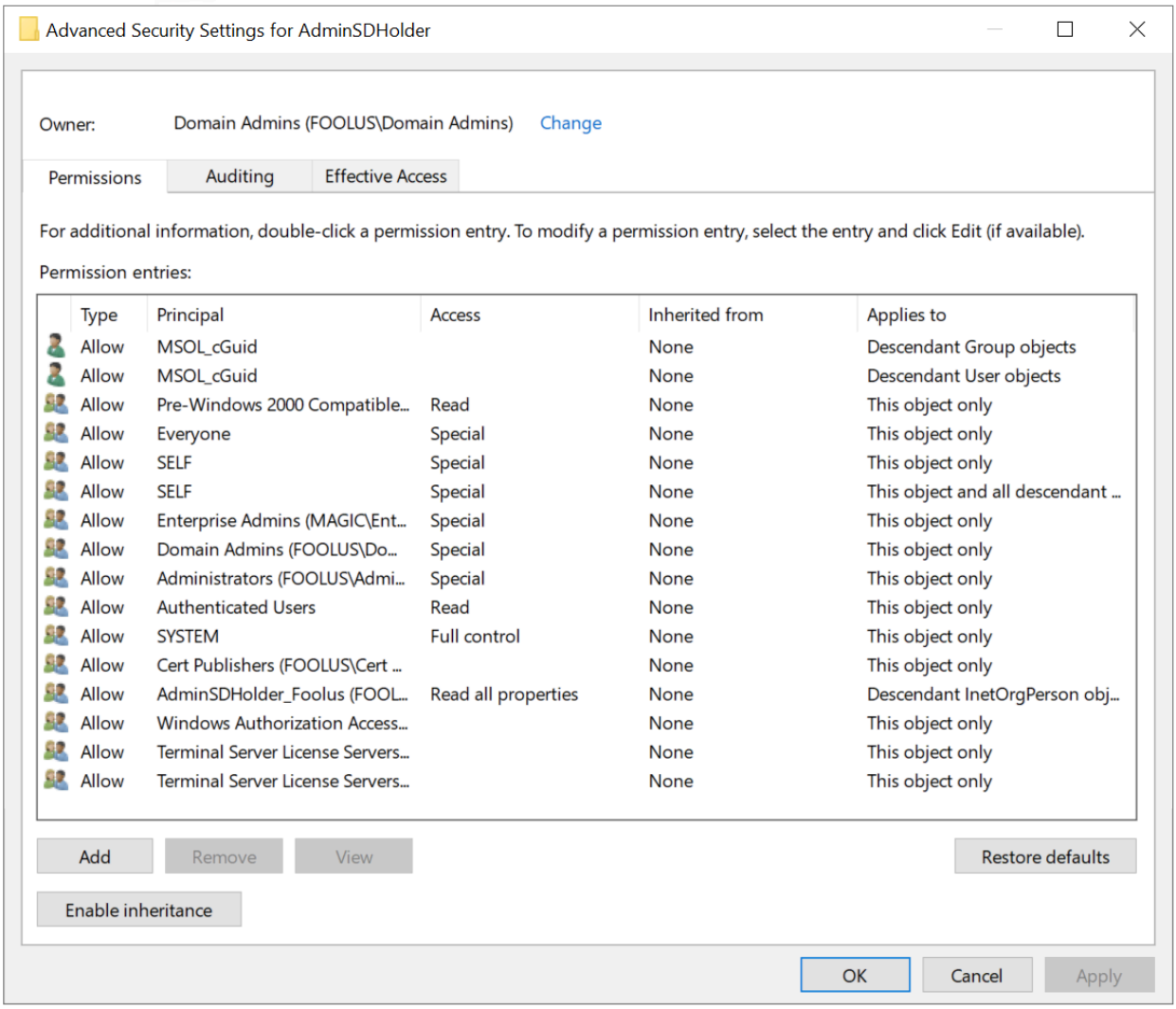


Figure 33 - Child Domain AdminSDHolder Security Descriptor - foolus.magic.lab.lan

I've created a user in the *foolus.magic.lab.lan* child domain called *EA Local-AdminSDHolder*. There is nothing special about this user other than that it has been added to the Enterprise Admins group for this AD Forest, which is located in the root domain of the *magic.lab.lan forest*.

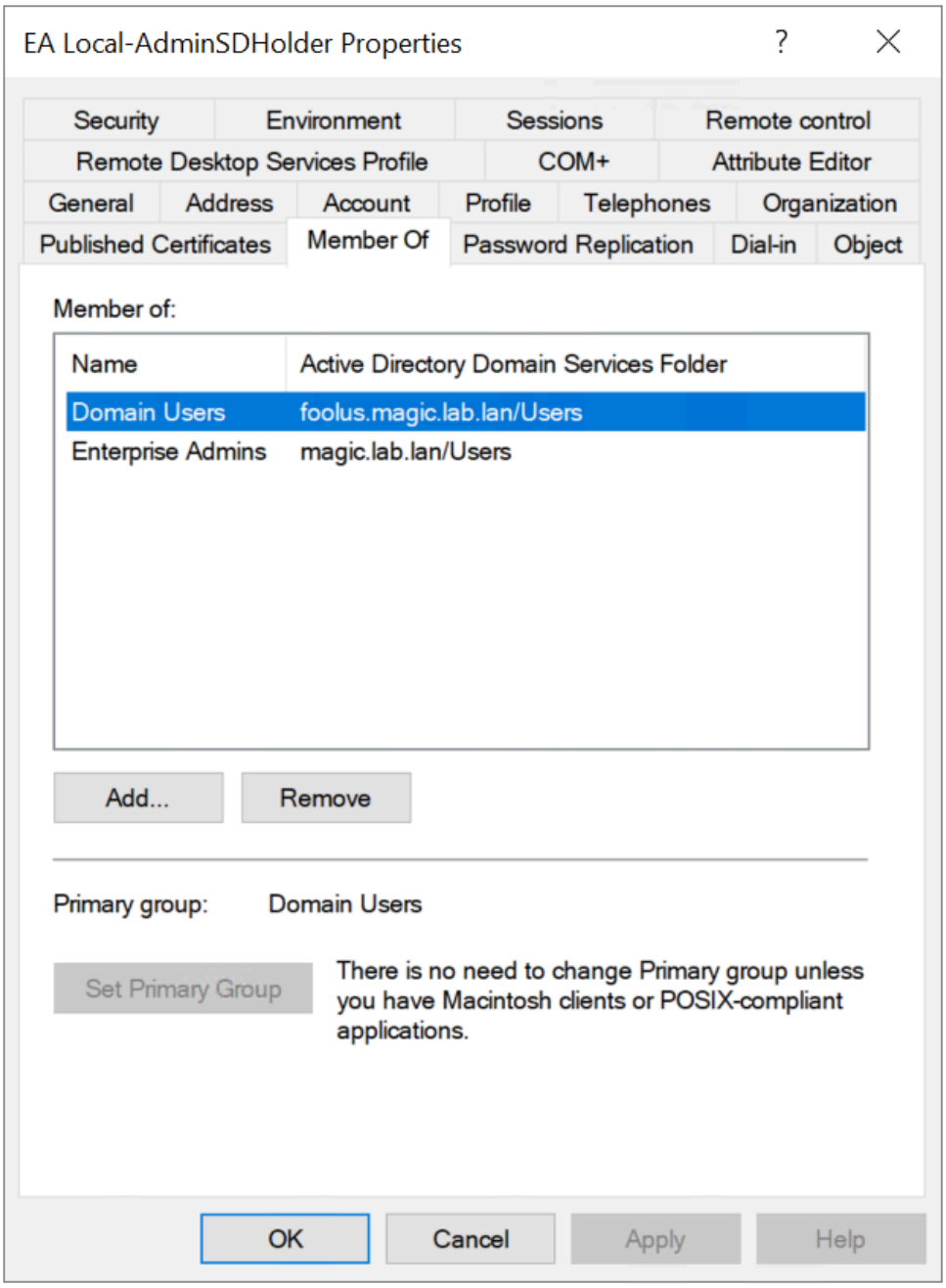


Figure 34 - EA Local-AdminSDHolder User in Child Domain, Which is a Member of Enterprise Admins in the Forest

I wasn't in any particular hurry, so I just waited and worked on other things instead of manually kicking off runProtectAdminGroupsTask. Sure enough, after a while, the *EA Local-AdminSDHolder* had its security descriptor changed and its adminCount set to 1. And when we compare this security descriptor with the AdminSDHolder security descriptors, we can plainly see that this is the security descriptor from the *foolus.magic.lab.lan* child domain, not that of the *magic.lab.lan* forest root where the Enterprise Admins group resides.

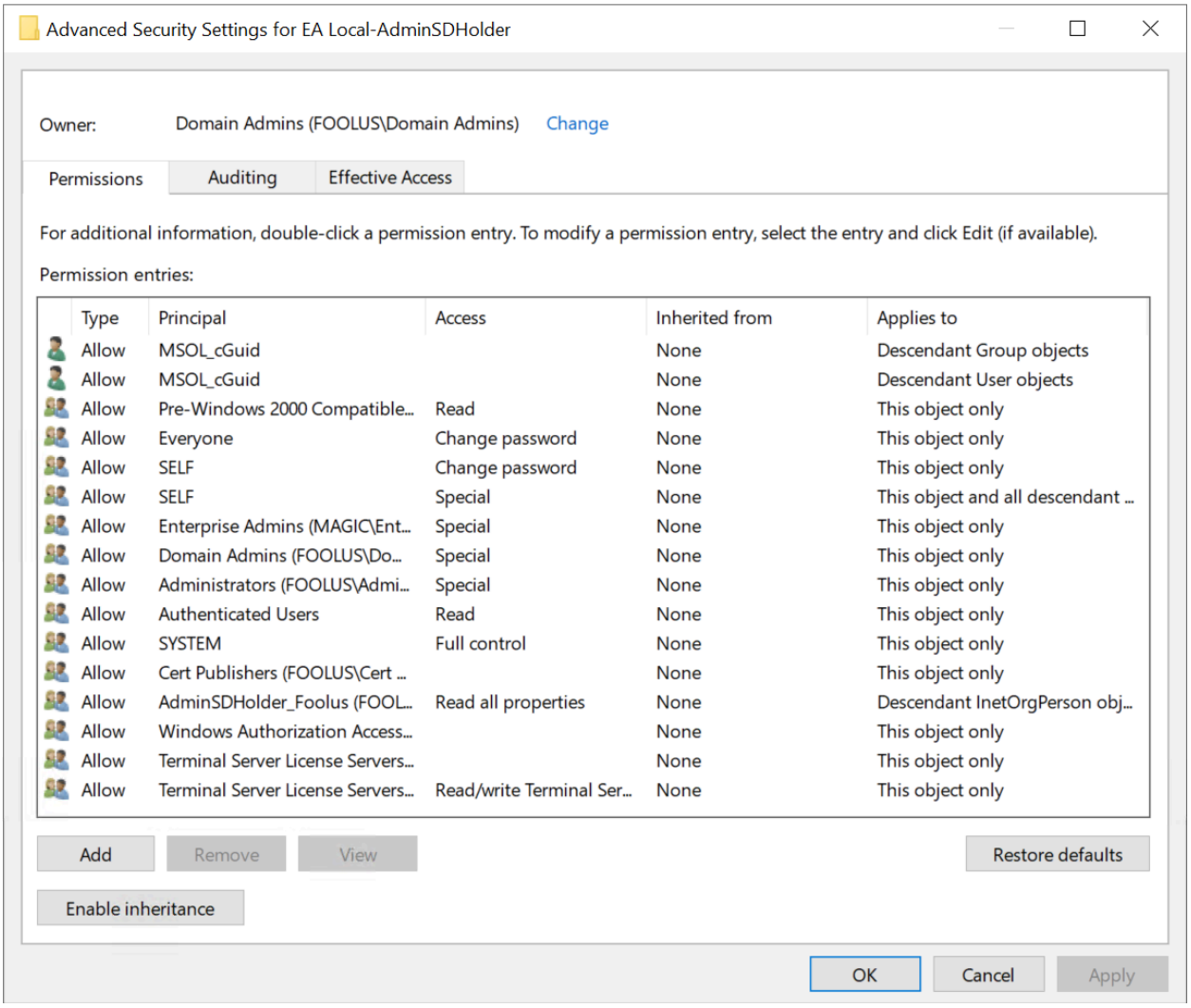


Figure 35 - Security Descriptor of the EA Local-AdminSDHolder User, Which Matches the AdminSDHolder Security Descriptor of the Child Domain

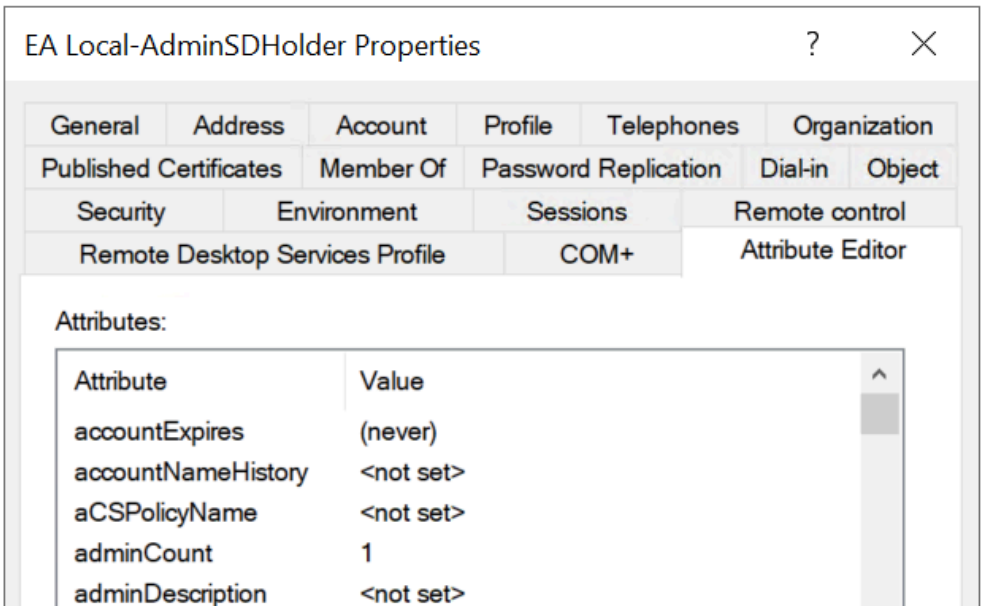


Figure 36 - EA Local-AdminSDHolder User adminCount Attribute Value

This indicates that the AdminSDHolder process in the *foolus.magic.lab.lan* child domain is the thread that protected the user, not the AdminSDHolder process of the forest root domain *magic.lab.lan* where the Enterprise Admins Universal security group is located.

I was very careful to specify that this is the behavior for protected groups with a Universal scope. In a multi-domain forest, groups with a Universal scope store their membership in a global catalog, not in the domain NC. Universal group membership is replicated between global catalog servers via linked-value (*Server 2003+*) replication. Universal group membership can also be cached in branch sites with the caveat that Universal group membership caching does not intercept calls made to the GC port 3268. However, as we will see later in this section, it's all a little less complicated than this.

Please read the Microsoft documentation on understanding security groups to learn about group scope in detail. Understanding group scope is a handy thing to know, especially since it can impact AdminSDHolder protections..

There are two types of groups which can have members: Security and Distribution.

Note: Special identity groups are also defined, but these do not have actual membership you can modify. Special identity groups are entities like Creator Owner and Authenticated Users.

Within the group types, there are several group scopes:

- Universal
- Global
- Domain Local
- Builtin Local

Universal group members can be accounts from any domain in the same forest, Global groups from any domain in the same forest, and other Universal groups from any domain in the same forest. Universal groups can be members of other Universal groups in the same forest, Domain Local groups in the same forest or trusting forests, or local groups on computers in the same forest or trusting forest.

Global group members can be accounts from the same domain or other Global groups from the same domain. Global groups can be members of Universal groups of any domain in the forest, other Global groups in the same domain, and Domain Local groups from any domain in the same forest or any trusting domain.

Domain Local group members can be accounts from any domain or trusted domain; Global groups from any domain or trusted domain; Universal groups from any domain in the same forest; other Domain Local groups from the same domain; and accounts, Global groups, and Universal groups from other forests and from external domains. Domain Local groups can be members of other Domain Local groups from the same domain, local groups on computers in the same domain, excluding built-in groups that have well-known security identifiers.

Domain Local and Global scoped group membership exists within their own domain context, not the forest.

Domain Local groups are especially interesting here as their membership is not present in the Global Catalog and thus not replicated across the entire forest whereas Universal group membership is part of the Global Catalog. A DC in Domain A doesn't have any insight into a Domain Local group in Domain B. We know from above that the domain where an account is located is responsible for applying AdminSDHolder protections.

So if I place an account, Global group, or Universal group from Domain A into a Domain Local group, like Administrators, on Domain B the PDCe in Domain A is not able to enumerate this nested membership in Domain B's Administrator group and does not apply AdminSDHolder protections.

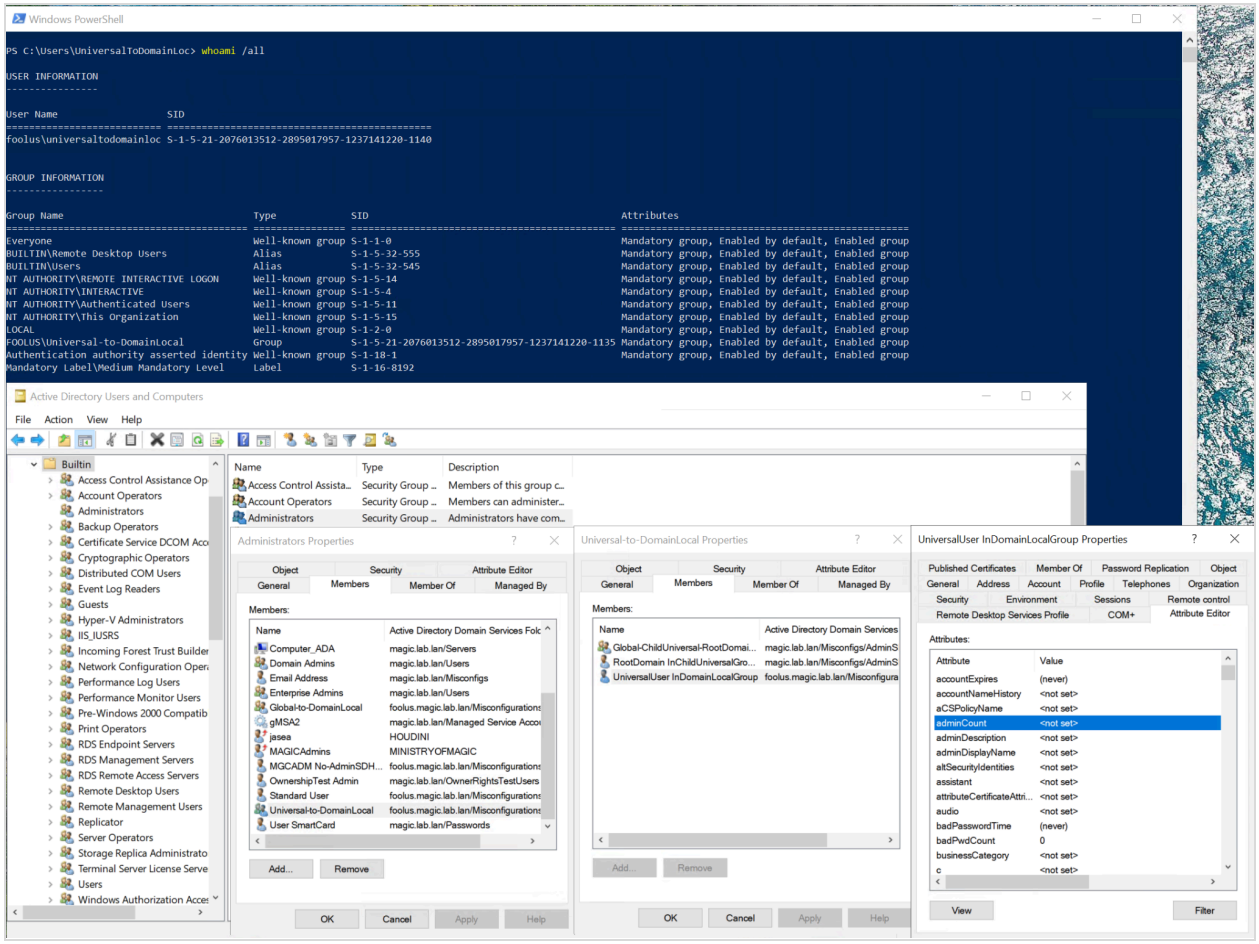


Figure 37 - Screenshot of a Windows PowerShell Session and ADUC open to Universal InDomanLocalGroup Properties

There's a lot going on in this screenshot, but an account from a child domain (i.e., *foolus\universaltodomainloc)* is logged into a workstation that is joined to *magic.lab.lan*. We have a portion of the member list of the Administrators group for *magic.lab.lan*. One member is the Universal-to-DomainLocal Universal group in the *foolus.magic.lab.lan* child domain, which is expanded to show that the user *UniversalUser InDomainLocalGroup* (with a samAccountName *universaltodomainloc*) from the same domain is a member. For the *UniversalUser InDomainLocalGroup* user, we can see that the adminCount property is not set.

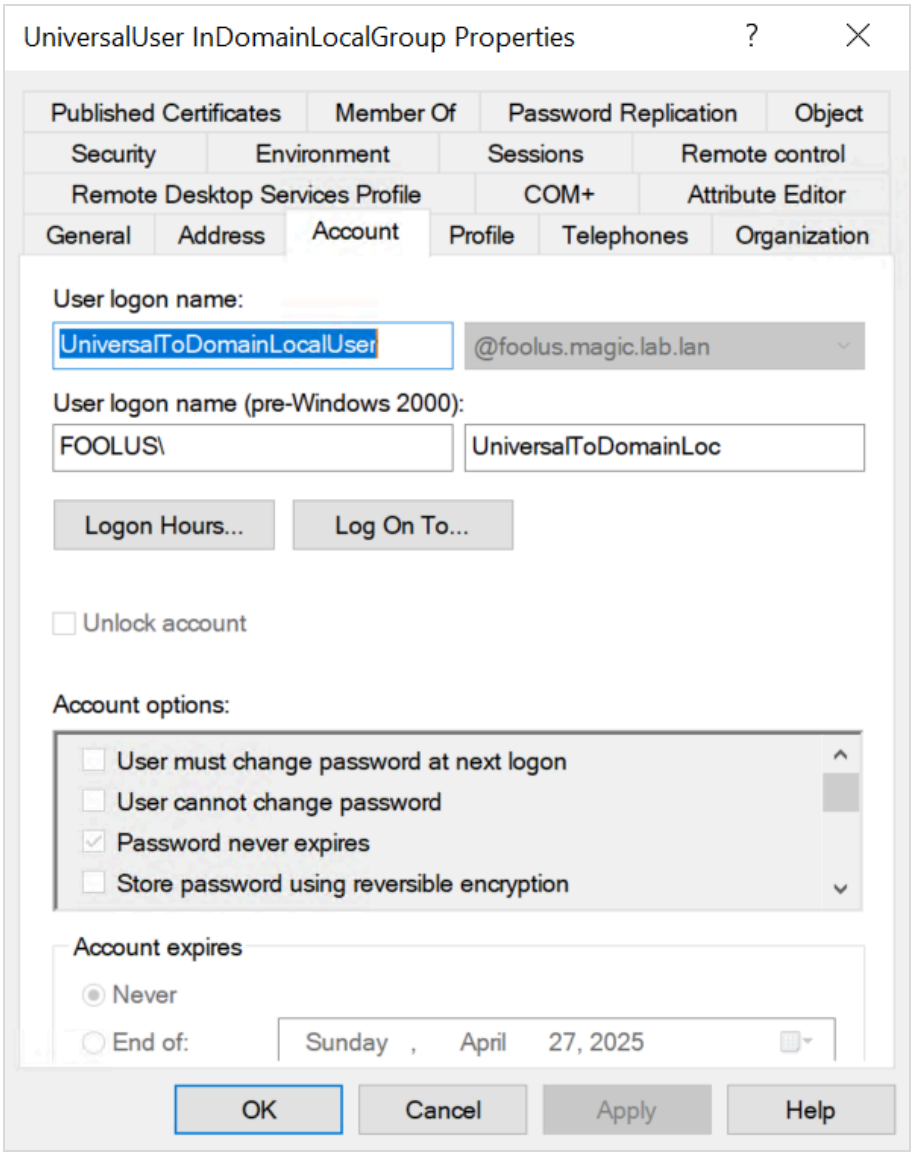


Figure 38 - Screenshot of UniversalUser InDomainLocalGroup Properties

Here we validate the user's displayName to samAccountName.

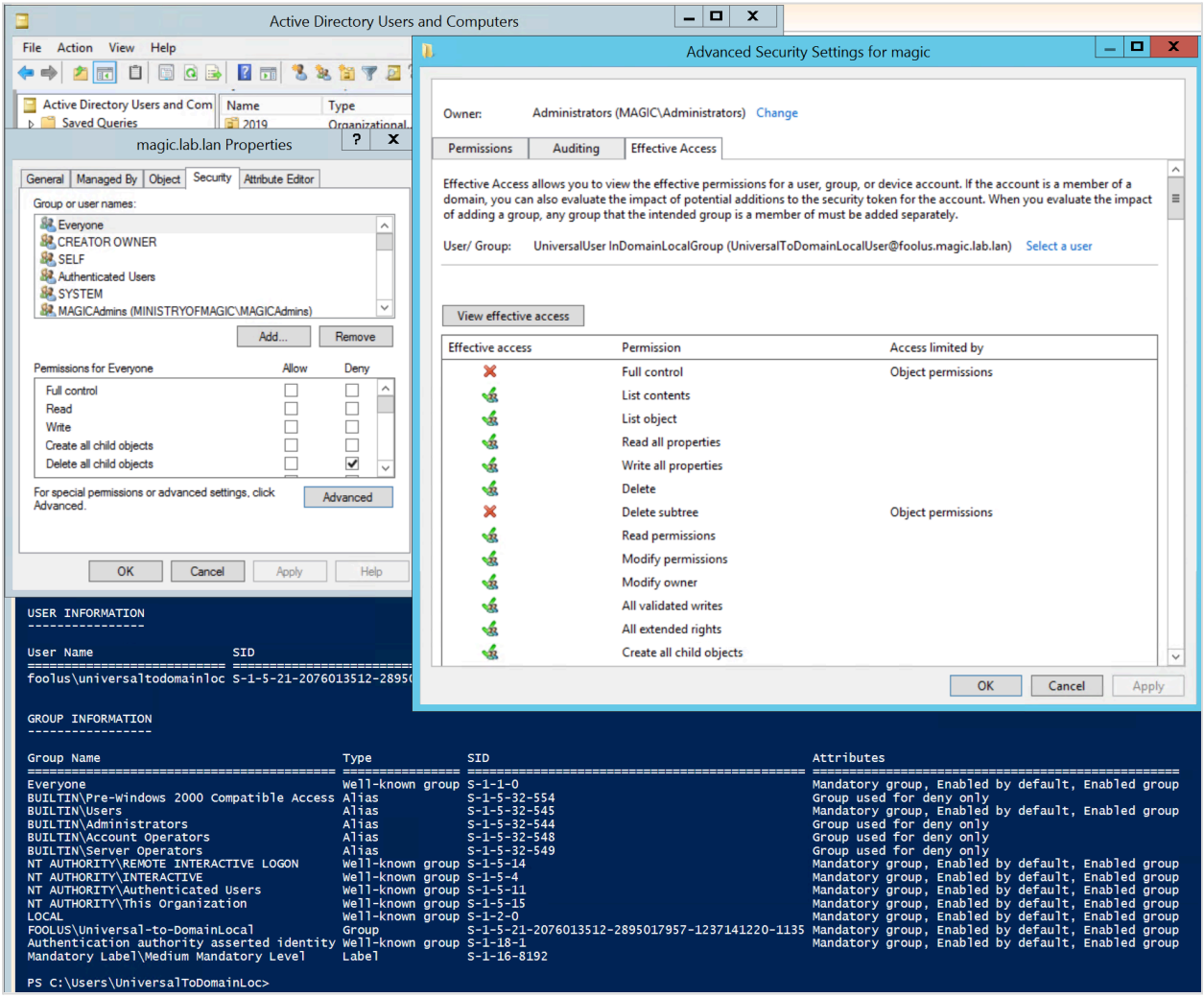


Figure 39 - Screenshot of ADUC Advanced Security Settings for Magic and PowerShell

And here is the *foolus\universaltodomainloc* account logged into *tellerdc02.magic.lab.lan*: a DC in the root domain of the forest. As you can see, in the authentication context of the DC, this account shows as a member of Administrators, Account Operators, and Server Operators. And when we look at the effective permissions on the domain root, we see that it has permissions consistent with a member of the Administrators group.

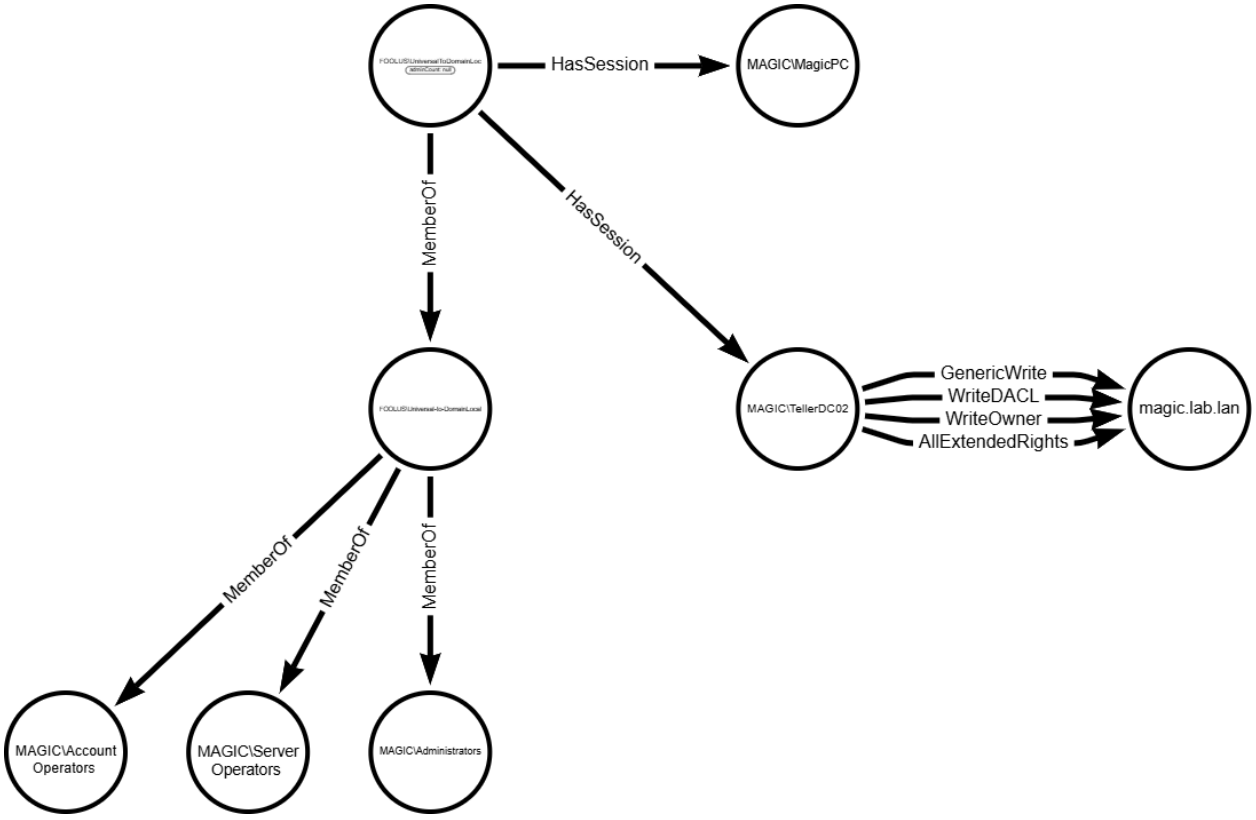
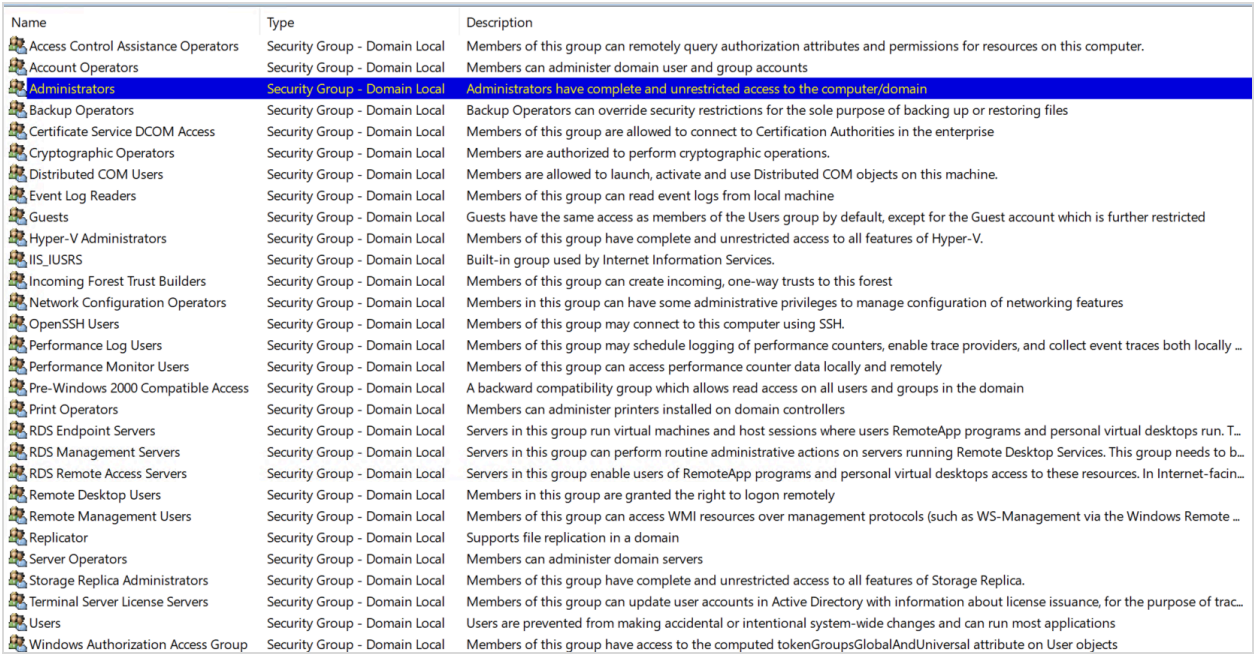


Figure 40 - Arrows.app Diagram of the UniversalUser InDomainLocalGroup Scenario

These Builtin Domain Local groups can be tricky to understand. Being a member of Account Operators in the *magic.lab.lan* domain doesn't grant rights and permissions associated with the Account Operators group across the entire domain. Rather, it only grants those rights and permissions in the authentication context of the DC for that domain. This means members of these Builtin groups must have either an interactive, remote interactive, network, batch, service, or task logon type on a DC for the rights and privileges associated with the Builtin group to be an active part of the user's access token. This is the case for all of the Builtin groups:



Name	Type	Description
Access Control Assistance Operators	Security Group - Domain Local	Members of this group can remotely query authorization attributes and permissions for resources on this computer.
Account Operators	Security Group - Domain Local	Members can administer domain user and group accounts
Administrators	Security Group - Domain Local	Administrators have complete and unrestricted access to the computer/domain
Backup Operators	Security Group - Domain Local	Backup Operators can override security restrictions for the sole purpose of backing up or restoring files
Certificate Service DCOM Access	Security Group - Domain Local	Members of this group are allowed to connect to Certification Authorities in the enterprise
Cryptographic Operators	Security Group - Domain Local	Members are authorized to perform cryptographic operations.
Distributed COM Users	Security Group - Domain Local	Members are allowed to launch, activate and use Distributed COM objects on this machine.
Event Log Readers	Security Group - Domain Local	Members of this group can read event logs from local machine
Guests	Security Group - Domain Local	Guests have the same access as members of the Users group by default, except for the Guest account which is further restricted
Hyper-V Administrators	Security Group - Domain Local	Members of this group have complete and unrestricted access to all features of Hyper-V.
IIS_IUSRS	Security Group - Domain Local	Built-in group used by Internet Information Services.
Incoming Forest Trust Builders	Security Group - Domain Local	Members of this group can create incoming, one-way trusts to this forest
Network Configuration Operators	Security Group - Domain Local	Members in this group can have some administrative privileges to manage configuration of networking features
OpenSSH Users	Security Group - Domain Local	Members of this group may connect to this computer using SSH.
Performance Log Users	Security Group - Domain Local	Members of this group may schedule logging of performance counters, enable trace providers, and collect event traces both locally ...
Performance Monitor Users	Security Group - Domain Local	Members of this group can access performance counter data locally and remotely
Pre-Windows 2000 Compatible Access	Security Group - Domain Local	A backward compatibility group which allows read access on all users and groups in the domain
Print Operators	Security Group - Domain Local	Members can administer printers installed on domain controllers
RDS Endpoint Servers	Security Group - Domain Local	Servers in this group run virtual machines and host sessions where users RemoteApp programs and personal virtual desktops run. T...
RDS Management Servers	Security Group - Domain Local	Servers in this group can perform routine administrative actions on servers running Remote Desktop Services. This group needs to b...
RDS Remote Access Servers	Security Group - Domain Local	Servers in this group enable users of RemoteApp programs and personal virtual desktops access to these resources. In Internet-facin...
Remote Desktop Users	Security Group - Domain Local	Members in this group are granted the right to logon remotely
Remote Management Users	Security Group - Domain Local	Members of this group can access WMI resources over management protocols (such as WS-Management via the Windows Remote ...
Replicator	Security Group - Domain Local	Supports file replication in a domain
Server Operators	Security Group - Domain Local	Members can administer domain servers
Storage Replica Administrators	Security Group - Domain Local	Members of this group have complete and unrestricted access to all features of Storage Replica.
Terminal Server License Servers	Security Group - Domain Local	Members of this group can update user accounts in Active Directory with information about license issuance, for the purpose of trac...
Users	Security Group - Domain Local	Users are prevented from making accidental or intentional system-wide changes and can run most applications
Windows Authorization Access Group	Security Group - Domain Local	Members of this group have access to the computed tokenGroupsGlobalAndUniversal attribute on User objects

Figure 41 - Screenshot of the Builtin Groups From a Windows Server 2025 Domain Controller

Initially, I had a bit of a challenging time understanding the logic flow for the AdminSDHolder ProtectAdminGroups task as it relates to handling accounts that are in different domains than the protected object they are a member of. Based on my observations and knowledge I had at the time, I created this flowchart to document the process flow around AdminSDHolder protection in cross-domain scenarios as I understood it:

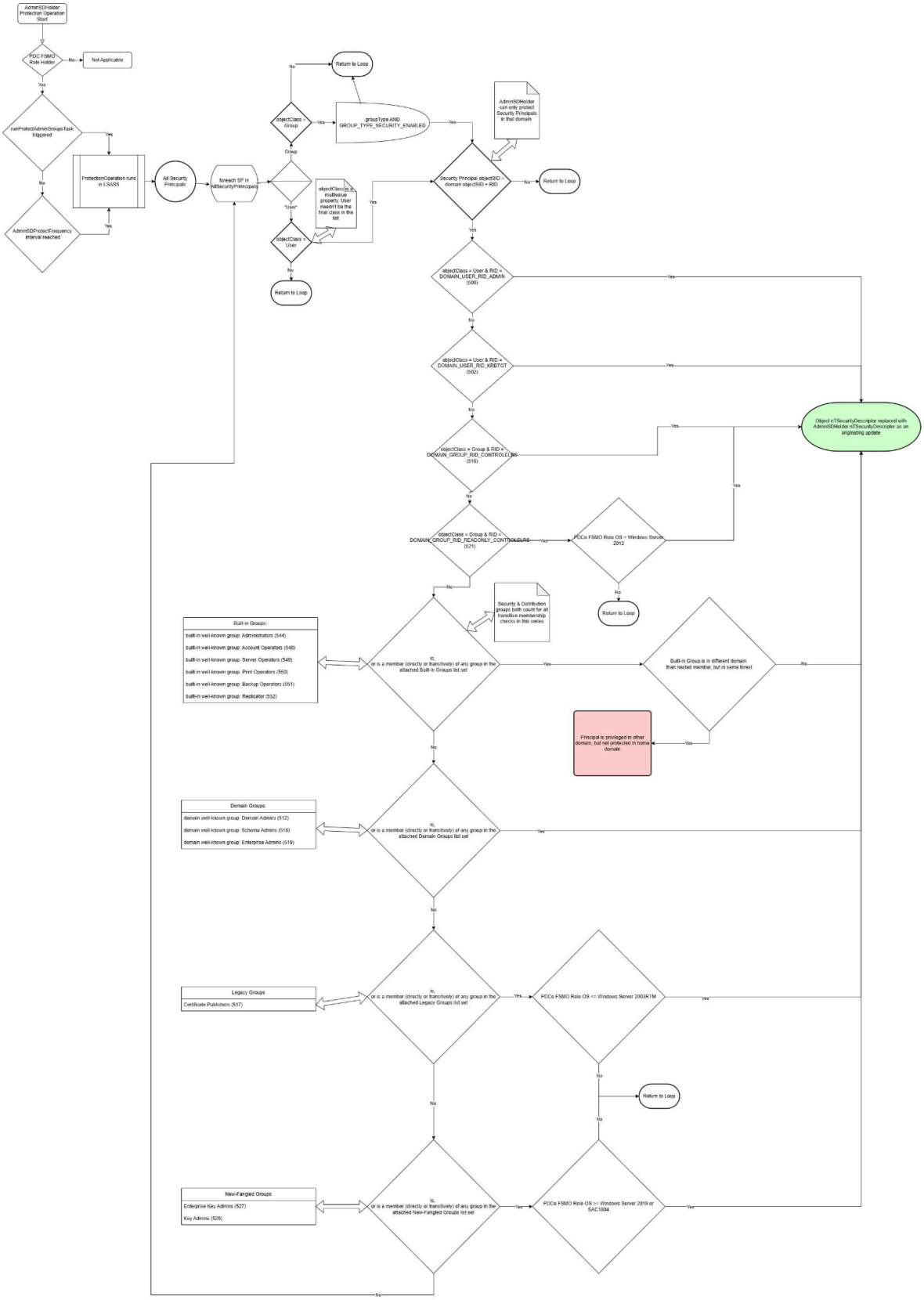


Figure 42 - AdminSDHolderProtectiveOperations-Observed (linked to full size PDF)

From an observability standpoint, the above flowchart arrives at a correct output. This was how I understood the process before I read the Windows NT5.1 source code and saw this very simple explanation for why AdminSDHolder in Domain A can only protect objects which reside in Domain A:

```
1063                //
1064                // We do not have to write anything if the member is not in the
1065                // same domain as this DC is authoritative for. If we are not
1066                // authoritative for this domain then skip.
1067                //
1068
1069                if (0==rpMembers[i]->SidLen)
1070                {
1071                    //
1072                    // Do not need to touch non security principals
1073                    //
1074
1075                    continue ;
1076                }
1077
1078                SampSplitNT4SID(&rpMembers[i]->Sid,&DomainSid,&Rid);
1079
1080
1081
1082                if (!RtlEqualSid((PSID) &DomainSid, &gAnchor.pDomainDN->Sid))
1083                {
1084                    //
1085                    // Not from our domain, skip and try the next entry
1086                    //
1087
1088                    continue;
1089                }
```

Figure 43 - Not From our Domain; Skip and Try the Next Entry - secadmin.c Screenshot

I spent a fair bit of time reviewing the, admittedly old, source code to understand how AdminSDHolder works under the hood. Here are some notes on what I observed versus what is in the code:

- **Observed**: Loop through all security principals, **Code**: Loop through builtin and domain accounts separately

- **Observed**: Filtering on user and user-derived objects, **Code**: All security principals fed into SampBuildAdministratorsSet() but SampFilterWellKnownAccounts() still filters on DomainUsersList
- **Observed**: DCs and Read-Only DCs handled with separate logic, **Code**: DCs and Read-Only DCs flagged with fExcludeMembers
- **Observed**: Nested members of Builtin groups that are in a different domain than the group are not protected, **Code**: Any account not from the same domain as the PDCe running the ProtectAdminGroups task is not protected
- **Observed**: Universal groups like Enterprise Admin and Schema admins are the only groups where cross-domain membership counts, **Code**: Enterprise Admins, Schema Admins, and presumably Enterprise Key Admins are handled separately and uniquely by being gathered before even the builtin and domain groups are gathered
- **Observed**: Transitive group membership determined somehow, **Code**: Transitive group membership determined by a thorough routine in SampBuildAdministratorsSet()

It's also important to note that the source code I reviewed is basically a Windows Server 2003 release candidate (RC). Changes were made to AdminSDHolder prior to Windows Server 2003 being RTM and, since then, Microsoft made a few other changes:

- dSHeuristics filtering of BuiltinDomainSecureAdminTable added in Windows Server 2000 SP4 (with appropriate KB installed) and Windows Server 2003 RTM
- Certificate Admins added to AccountDomainSecureAdminTable for Windows Server 2000 SP4 and Windows Server 2003 RTM and then removed in Windows Server 2003 SP1
- Read-Only DCs added to AccountDomainSecureAdminTable with fExcludeMembers = true in Windows Server 2012
- Key Admins added to AccountDomainSecureAdminTable with Windows Server 2016. Based on data collection, observation, and extrapolation I speculate that Enterprise Key Admins was also added to the AccountDomainSecureAdminTable in Windows Server 2016, but due to the way that the AccountDomainSecureAdminTable is handled in the process, a Universal security group would not be protected, as is the case in Server 2016.
- Presumably, the 'Build SIDs of Enterprise Admins & Schema Admins' process along with the associated EnterpriseAndSchemaAdminsDsNames was modified to include Enterprise Key Admins in Server 2019

Of these changes, dSHeuristics is likely the most complex of the changes, but still presumably a filter on the BuiltinDomainSecureAdminTable. Adding Enterprise Key Admins to the EnterpriseAndSchemaAdminDsNames would take a small bit of work. The rest of the changes are presumably simple modifications to the AccountDomainSecureAdminTable.

I then created another flowchart to describe the process as I now understood it from the code and extrapolating the changes made to the process since Windows Server 2003 RC:

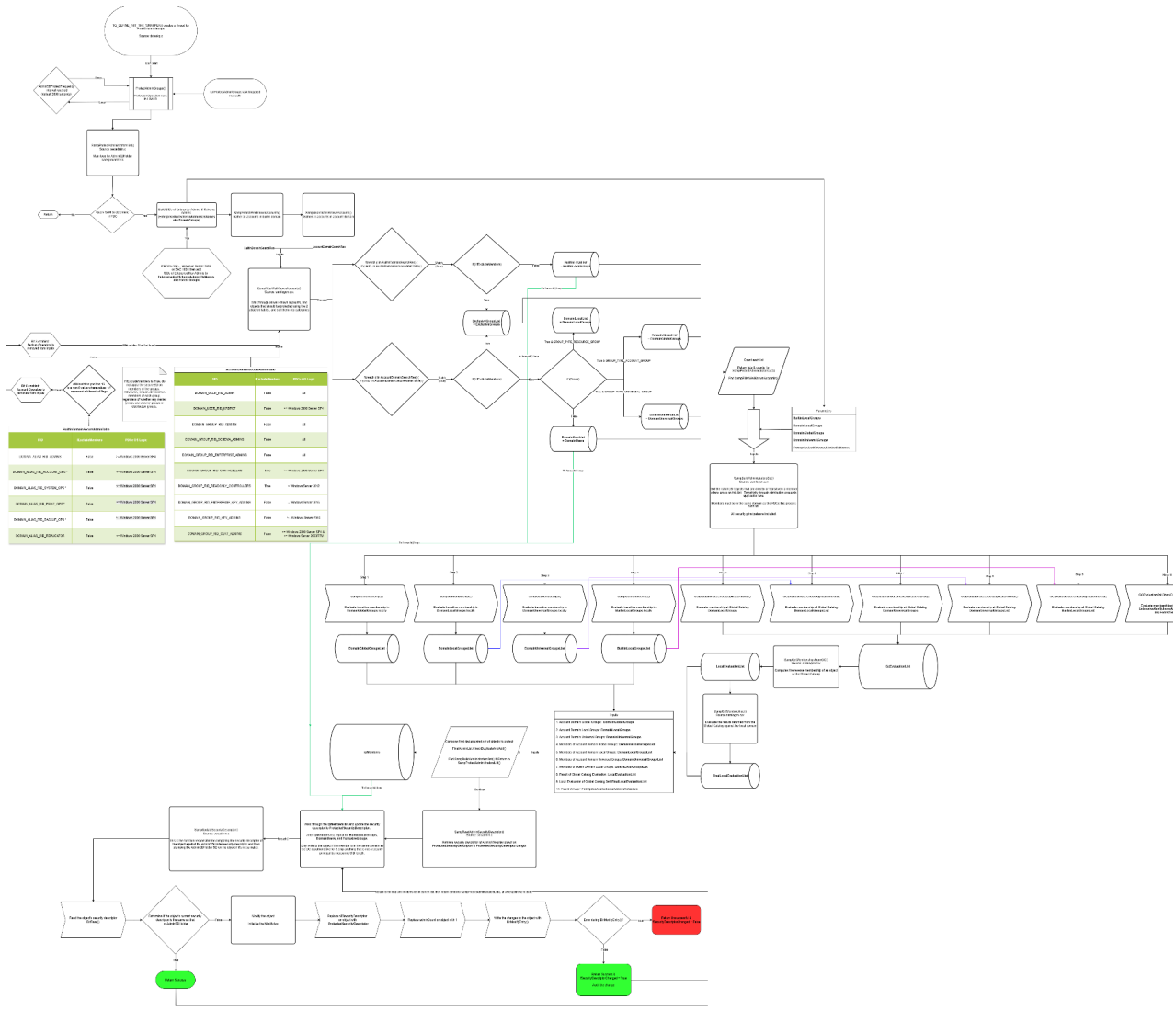


Figure 44 - AdminSDHolderProtectiveOperations-CodeReview (linked to full size PDF)

This flowchart design accounts for the code for the ProtectAdminGroups task which is primarily in the secadmin.c and samlogon.cxx source code files for WinNT5.1. The above code screenshot from secadmin.c accounts for why only objects in the same domain as the PDCe performing the ProtectAdminGroups task are protected. The rest of the oddities in processing Builtin Domain Local groups can be accounted for by this code comment in samlogon.cxx:

```
3710            //
3711            // Evaluate memberships of following sets at G.C.
3712            //
3713            // 1. account domain local groups
3714            // 2. members of account domain local groups
3715            // 3. account domain universal groups
3716            // 4. members of account domain universal groups
3717            // 5. members of builtin local groups
3718            // 6. enterprise admins and schema admins
3719            //
3720            // Note: Do NOT send
3721            //    account domain global groups or
3722            //    builtin domain local groups
3723            // to G.C.
3724            //
```

Figure 45 - Do NOT Send Account Domain Global Groups or Builtin Domain Local Groups to GC - samlogon.cxx Screenshot

The group scopes matter, and because they matter the ProtectAdminGroups task takes them into account accordingly.

To reiterate, cross-domain nesting of an account, Global group, or Universal group into a protected builtin Domain Local group, like Administrators, in another domain in the same forest results in a privileged account or group that is not protected by AdminSDHolder. This could be abused by an attacker for persistence.

Let's walk through a potential attack path abusing how AdminSDHolder only applies to principals that are in the same domain as the PDCe running the ProtectAdminGroups task. In some scenarios an account UserA from a child domain could be a member of the Enterprise Admins group in the root domain. An attacker has compromised the UserB account, also in the child domain. UserB is a MemberOf Domain Admins in the child domain, but currently has no privileged rights in the root domain of the forest.

Note: Bear in mind that the forest is the security boundary in AD, so this would not be considered a serviceable security issue by Microsoft.

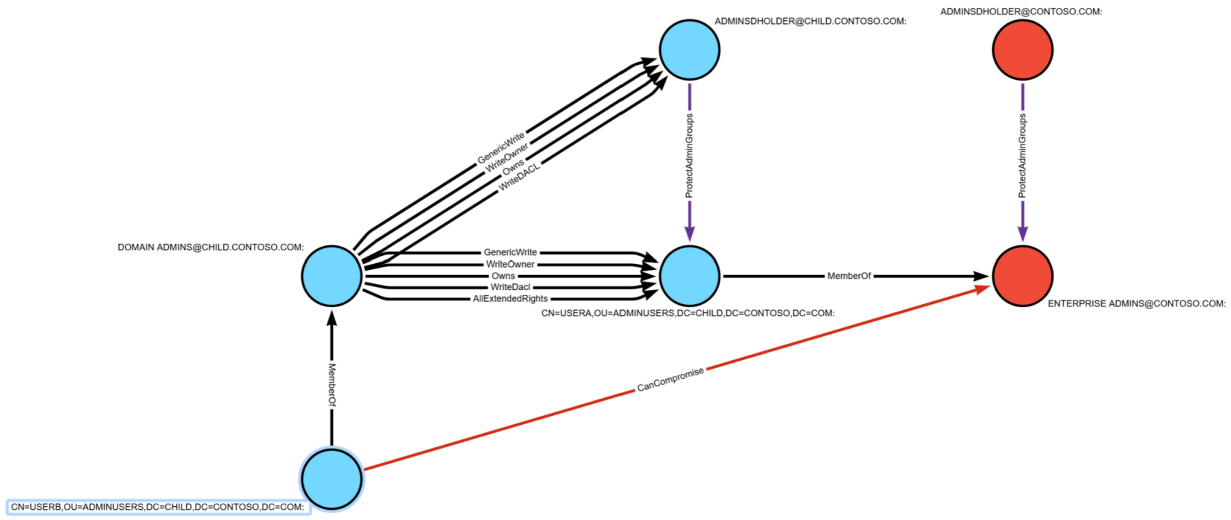


Figure 46 - Cross-Domain Attack Path

The ProtectAdminGroups background task in the child domain tattoos the AdminSDHolder SD on protected objects in that child domain. The ProtectAdminGroups background task in the root domain tattoos the AdminSDHolder SD on protected objects in the root domain. Because Enterprise Admins is a security group that the ProtectAdminGroups task will endeavor to enumerate recursive membership of across domain trusts, the ProtectAdminGroups task in the child domain will tattoo the AdminSDHolder SD from the child domain on the UserA object. In other words, even though the Enterprise Admins group is located in the root domain of the forest, this specific member will be protected by AdminSDHolder in the child domain, not of the root. Since UserB is a member of Domain Admins in the child domain they can modify the AdminSDHolder SD in the child domain and either wait or manually kick off ProtectAdminGroups. Or UserB can modify UserA's SD directly, grant UserB an abusable account takeover primitive by modifying a property on UserA, and let ProtectAdminGroups clean up part of the evidence.

As I said earlier, the forest is the security boundary and there are multiple ways to accomplish movement from a child domain to a forest root domain. BloodHound 7.6.0 and many versions earlier would map out this attack path using the standard edges in black. This is merely an explanation of why that attack path is possible.

AdminCount and Forest Trusts

Between AD Forests, things get even more dangerous when foreign security principals are placed in privileged groups. In the foreign forest, the account will not be stamped by AdminSDHolder at all. It won't receive an adminCount either, because that foreign, trusted forest doesn't know anything about the privilege its local account has in another forest.

Here's an example from one of my lab environments:

- The AD forest *ministryofmagic.lab.lan* is built as an administrative forest. There are inbound trusts from multiple other forests, which allow *ministryofmagic.lab.lan* to authenticate to those resource forests.
- The AD forest *mindfreak.lab.lan* is built as a resource forest. This forest is one of those that has an outbound trust to *ministryofmagic.lab.lan*, which is an inbound trust from the perspective of that forest.

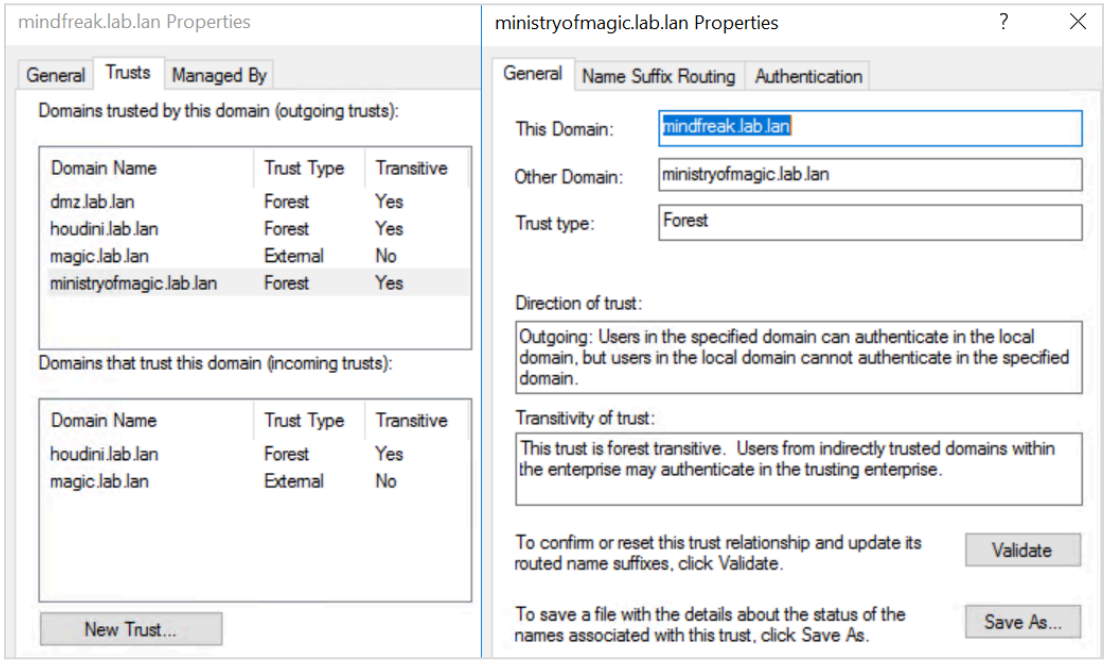


Figure 47 - Screenshot of ministryofmagic.lab.lan Properties

- In the *mindfreak.lab.lan* forest, there are multiple members of the Builtin Administrators group for the root domain of that forest, one of which is “MINDFREAKAdmins”. We can note that this group member has a little red up arrow by its icon, denoting it as a Foreign Security Principal (FSP). In this case, the FSP is a group from MINISTRYOFMAGIC, which is the NetBios name for the root domain of the *ministryofmagic.lab.lan* forest.

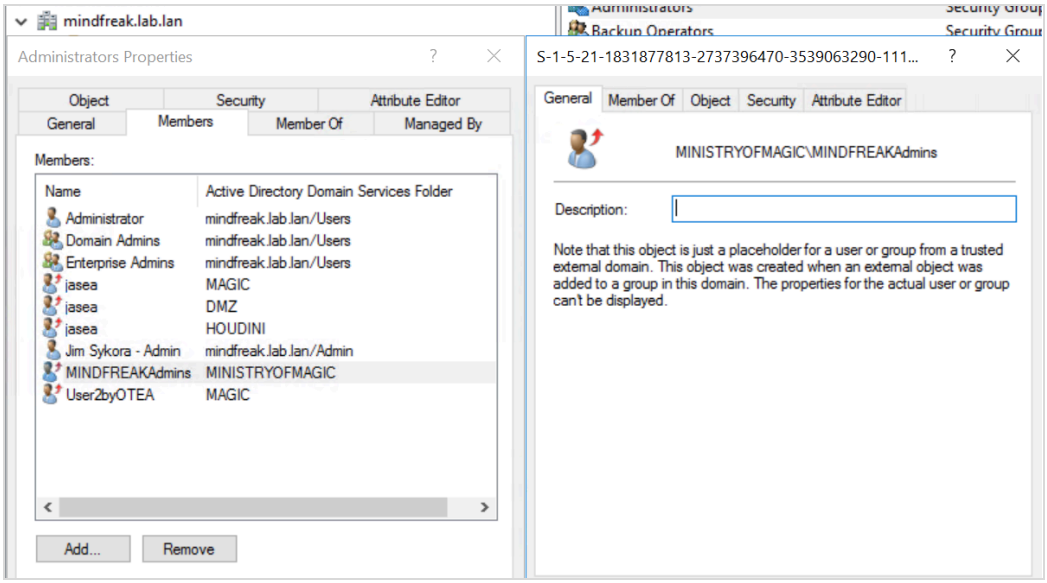


Figure 48 - Screenshot of MINDFREAKAdmins FSP

- Looking a little closer at the FSP, we can see that the FSP object is not protected by AdminSDHolder as it doesn't have a restrictive security descriptor and the adminCount property is also not set:

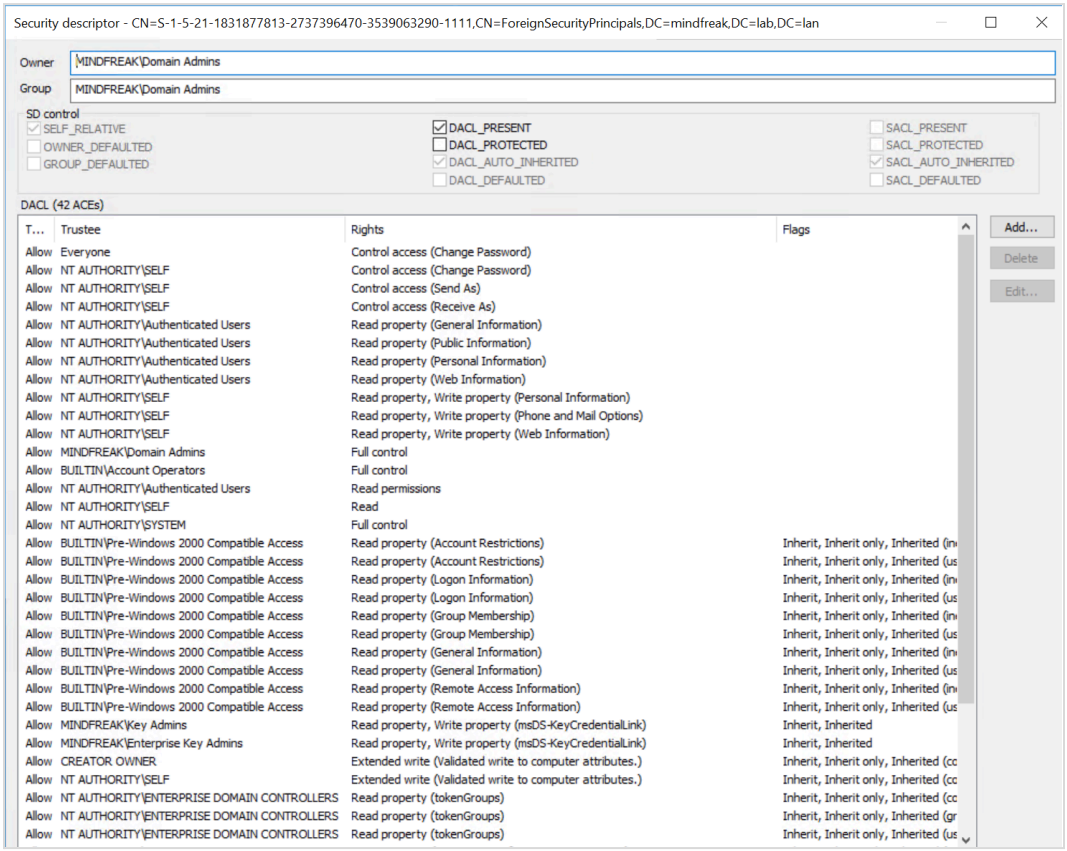


Figure 49 - Screenshot of FSP Security Descriptor

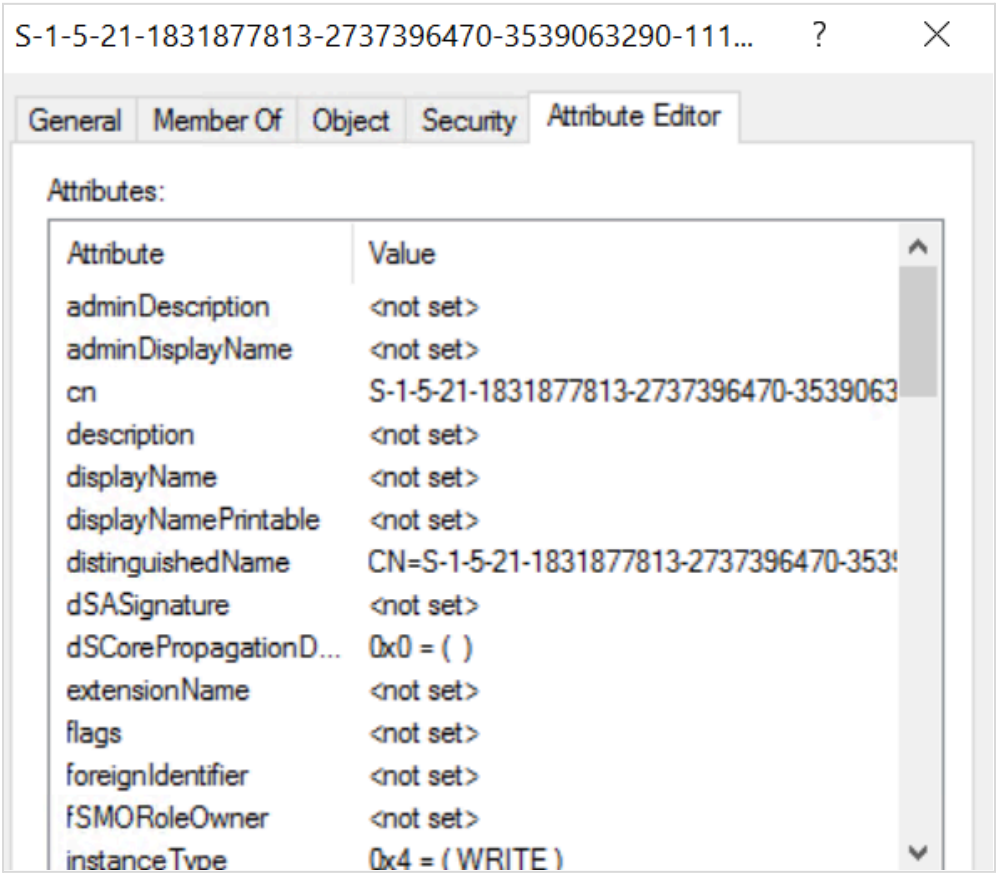


Figure 50 - Screenshot of FSP Properties

- Another thing of note about the FSP is that its SID is S-1-5-21-1831877813-2737396470-3539063290-1111, as is the name and part of the distinguishedName.
- Comparing the SID of the FSP to the SID of the MINDFREAKAdmins group in *ministryofmagic.lab.lan*, we have a match:

```
------------
Expanding base 'CN=MINDFREAKAdmins,OU=Groups,OU=Managed-T0,OU=Admin,DC=ministryofmagic,DC=lab,DC=lan'...
Getting 1 entries:
Dn: CN=MINDFREAKAdmins,OU=Groups,OU=Managed-T0,OU=Admin,DC=ministryofmagic,DC=lab,DC=lan
    cn: MINDFREAKAdmins;
    distinguishedName: CN=MINDFREAKAdmins,OU=Groups,OU=Managed-T0,OU=Admin,DC=ministryofmagic,DC=lab,DC=lan;
    dSCorePropagationData (5): 1/2/2024 2:20:15 PM Central Daylight Time; 1/2/2024 2:05:43 PM Central Daylight Time; 1/2/2024 2:04:29 PM Central Daylight Time; 1/2/2024
        2:02:33 PM Central Daylight Time; 0x1 = ( NEW_SD ), 0x1 = ( NEW_SD ), 0x1 = ( NEW_SD ), 0x1 = ( NEW_SD );
    groupType: 0x80000008 = ( UNIVERSAL_GROUP | SECURITY_ENABLED );
    instanceType: 0x4 = ( WRITE );
    member: CN=Jim Sykora - Managed Forest Admin,OU=Accounts,OU=Managed-T0,OU=Admin,DC=ministryofmagic,DC=lab,DC=lan;
    name: MINDFREAKAdmins;
    objectCategory: CN=Group,CN=Schema,CN=Configuration,DC=ministryofmagic,DC=lab,DC=lan;
    objectClass (2): top; group;
    objectGUID: 5cb6b78d-121f-441d-af8a-e90f2fbc67de;
    objectSid: S-1-5-21-1831877813-2737396470-3539063290-1111;
    sAMAccountName: MINDFREAKAdmins;
    sAMAccountType: 268435456 = ( GROUP_OBJECT );
    uSNChanged: 28946;
    uSNCreated: 13003;
    whenChanged: 1/4/2024 10:39:02 AM Central Daylight Time;
    whenCreated: 8/21/2023 8:22:25 AM Central Daylight Time;
```

Figure 51 - Screenshot of MINDFREAKAdmins Group Properties from LDP

- The security descriptor of the MINDFREAKAdmins group is not restrictive and there is no adminCount:

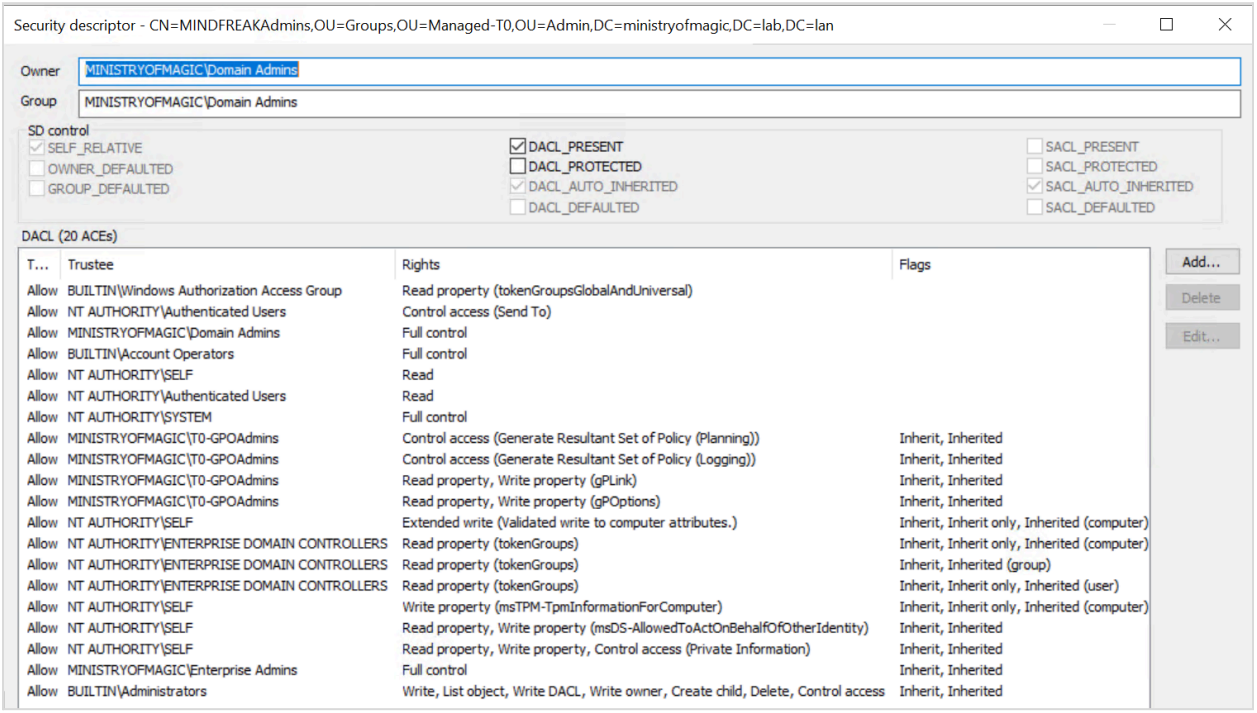


Figure 52 - Screenshot of MINDFREAKAdmins Security Descriptor

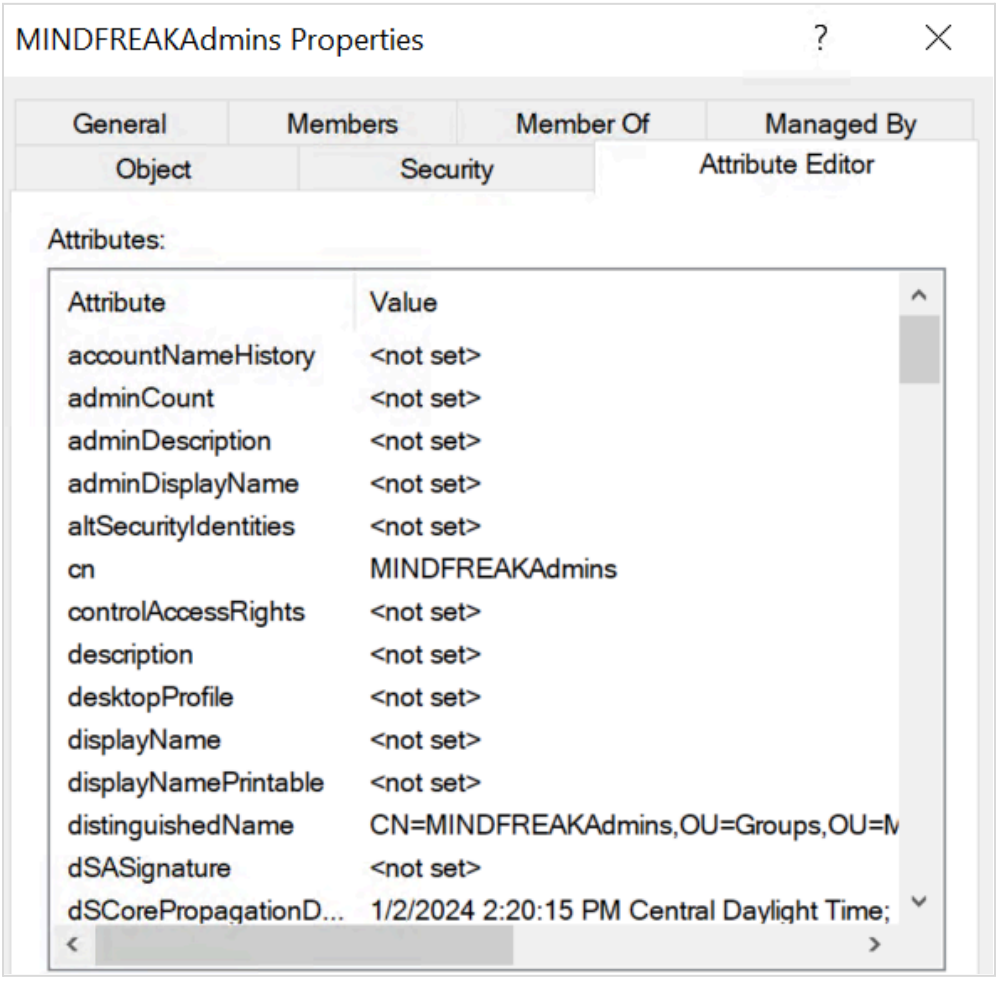


Figure 53 - Screenshot of MINDFREAKAdmins Properties

- The MINDFREAKAdmins group has one member:

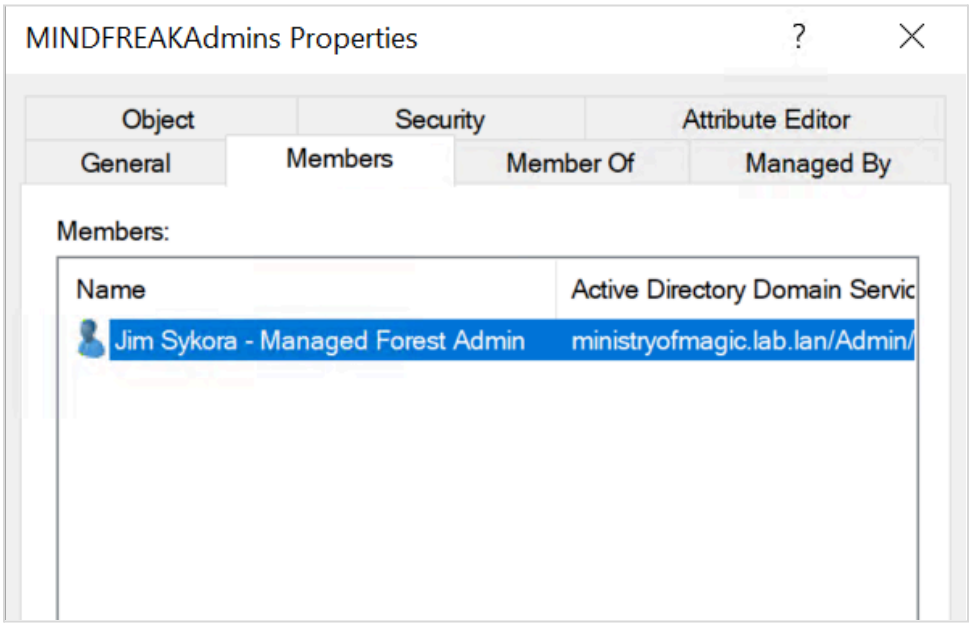


Figure 54 - Screenshot of MINDFREAKAdmins Members

- AdminSDHolder does not protect this user, which is a nested member of the Administrators group in the *mindfreak.lab.lan* root domain

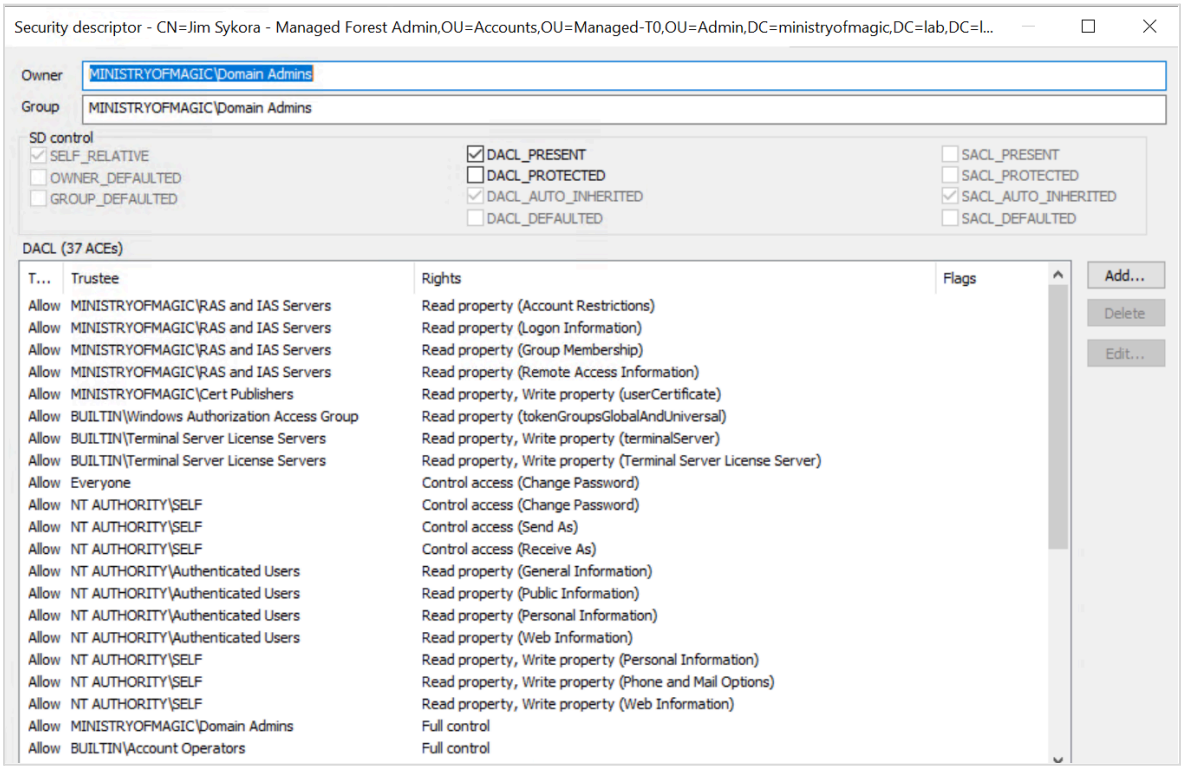


Figure 55 - Screenshot of Jim Sykora - Managed Forest Admin Security Descriptor

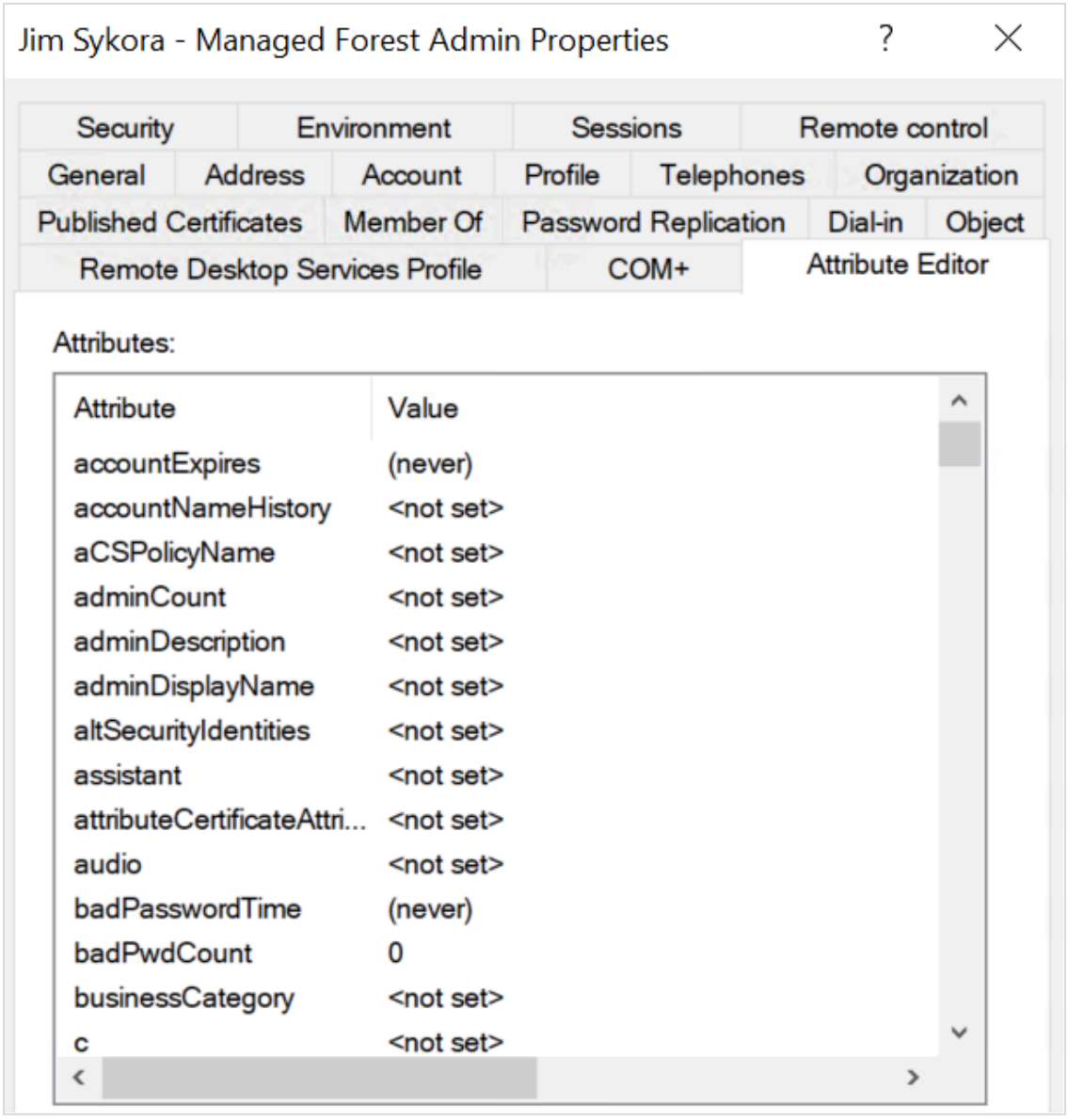


Figure 56 - Screenshot of Jim Sykora - Managed Forest Admin Properties

Done ~~correctly~~ perfectly, cross-forest administration can be one of the most secure ways to administer AD; however, when done poorly, it opens the environment up to even more risks.

There are still plenty of ESEA or Red forests out there. They are difficult to implement and even harder to maintain. Even when using functionality specific to the PAM AD Optional Feature, like Shadow Principals, it is highly likely that AdminSDHolder does not protect the most privileged of admin accounts in your collection of forests.

Recommendations:

Monitor for highly privileged users in more ways than just relying on the adminCount attribute. Sometimes if the adminCount is 1, it's not privileged anymore and sometimes it could be privileged without an adminCount.

Also, when removing privileges from an account, don't fall into the trap of setting the adminCount to 0 and enabling inheritance on that object. The account might not be privileged because of that specific group anymore, but it could easily still be privileged in other ways either through explicit ACEs on objects, nonstandard object ownership, or delegation. Disable and decommission the formerly privileged account and create a new unprivileged account for that purpose.

Misconception: Permanence and Persistence

The objects AdminSDHolder protects are not written in stone. Certainly, they have a strict DACL applied that, by default, only allows Administrators, Domain Admins, and Enterprise Admins to modify the object...but what if we're in a post-compromise scenario where persistence might be interesting?

Utilizing AdminSDHolder as a post-compromise persistence method is a topic commonly blogged about. Sean Metcalf was, I believe, the first to give a talk about abusing AdminSDHolder for sneaky AD persistence back at DerbyCon 2015. Sean also wrote a blog post about this topic on adsecurity.org. Sneaky Active Directory Persistence #15: Leverage AdminSDHolder & SDProp to (Re)Gain Domain Admin Rights has a few misconceptions in it, but DACL abuse and AD persistence content references this post time and again.

What if someone (or something) followed through with **Misconception 4: Just Change AdminSDHolder's DACL** and weakened the stringent DACL enough to give an attacker an avenue to gain and regain Domain Admin rights? Are there other scenarios where attackers can abuse or subvert AdminSDHolder?

Recall that the AdminSDHolder Background Task runs every 60 minutes, by default, on the PDC Emulator for the domain. If an attacker in a post-compromise position can modify the DACL on a highly-privileged and otherwise-protected AD security principal, like a member of Domain Admins, they have just under 60 minutes to make further modifications on that account before AdminSDHolder re-stamps it.

How might this work? I'm not sure. Maybe an attacker gains SYSTEM on a DC but it's too risky to dump credentials or perform a DCSync attack due to anticipated logging, endpoint detection and response (EDR), or agents. They could use the SYSTEM access to modify AD Objects like the Domain Admin account, perform tasks that appear to be routine domain administration tasks, and then quickly revert the security descriptor on the Domain Admin account back to the exact same one that AdminSDHolder has so it won't be re-stamped.

Regardless, if an attacker somehow compromises an AD Object that AdminSDHolder protects, they have a window of time until the next AdminSDHolder Background Task on the PDCe runs. If the PDCe was unavailable or otherwise unable to replicate the security descriptor back to other DCs in the domain/forest, then that incorrect and insecure configuration would remain until the PDCe or the replication issues with the PDCe were corrected.

To test this, I utilized the *Inheritance.AD2025.lan* domain in my lab:

1. First, I used `Get-ADDomain | Select-Object PDCEmulator` to determine which DC of the two in the domain is the PDCe using. This returned *SD-DC2025-I1.Inheritance.AD2025.lan*.
2. I disconnected the virtual NIC for SD-DC2025-I1 so that SD-DC2025-I2 could not reach it:

```
PS C:\Users\Administrator.INHERITANCE> ping SD-DC2025-I1.Inheritance.AD2025.lan -t

Pinging SD-DC2025-I1.Inheritance.AD2025.lan [10.10.15.126] with 32 bytes of data:
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Reply from 10.10.15.126: bytes=32 time<1ms TTL=128
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
Request timed out.
```

Figure 57 - Screenshot of PowerShell Session Ping Command

3. On SD-DC2025-I2, I created a new user named *PDCeTest* and added it to the Domain Admins group. Then I waited because, as with the PDCe down, there is no way for me to manually trigger the ProtectAdminGroups task to run. In this domain, the AdminSDHolder interval is the default 60 minutes.
4. After waiting for over an hour, I checked the status of the *PDCeTest* account from SD-DC2025-I2 and validated that AdminSDHolder did not protect it. No 4780 events were created on SD-DC2025-I2, as expected.

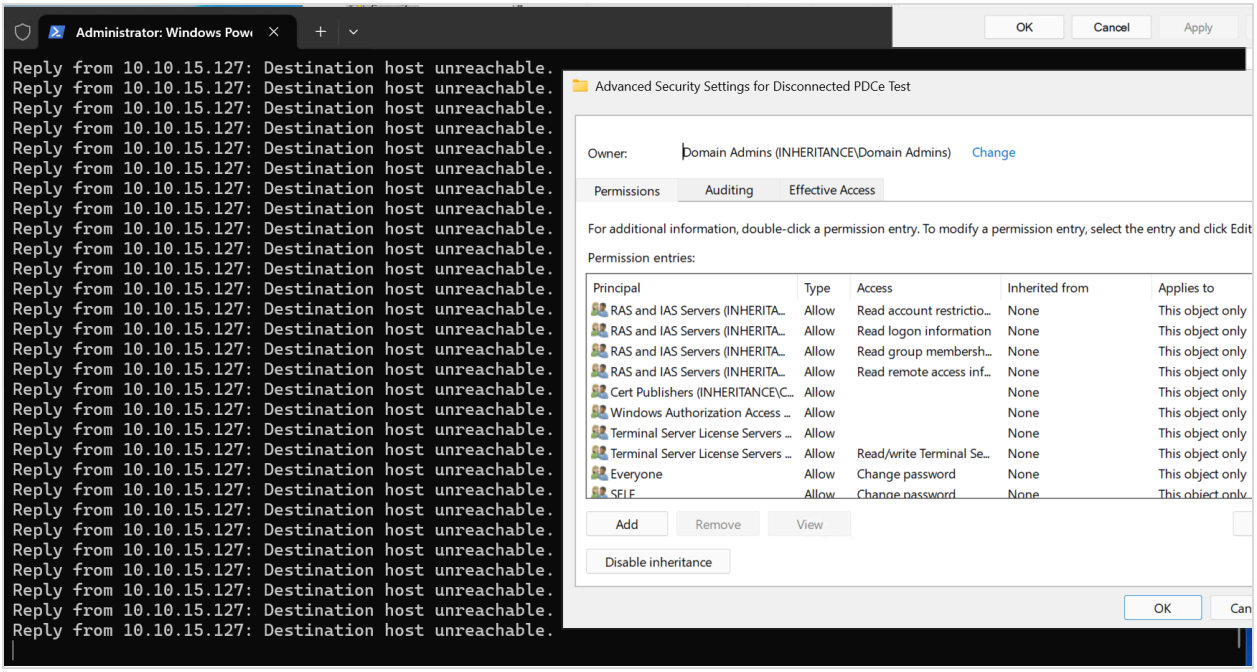


Figure 58 - Screenshot of PowerShell Terminal Ping Command and ADUC Security Descriptor

5. The *PDCeTest* user account was not protected on the PDCe (i.e., SD-DC2025-I1) either because this account was not replicated to this DC upon creation because of the down network connection.

6. After re-enabling the network connection on SD-DC2025-I1, the *PDCeTest* user account was replicated there from SD-DC2025-I2 and, on the next ProtectAdminGroups task cycle, AdminSDHolder protected it.

Removing connectivity to the rest of the domain from the PDCe FSMO role holder does prevent AdminSDHolder from protecting any objects in the rest of the domain's DCs; however, disabling the network connection of the PDCe or otherwise performing some form of denial-of-service (DoS) is likely to have unintended consequences in a production environment.

Another scenario which can prevent the ProtectAdminGroups, and other Background Tasks, from running is when `gfDisableBackgroundTasks = TRUE` in the context of the Directory Services Agent. How can we arrive at this state?

- If IsSetupRunning() evaluates to true
- If the SuppressBackgroundTasksHeuristic is set

At first I thought the SuppressBackgroundTasksHeuristic was some ancient configuration of the dSHeuristics attribute we'll be covering in the Misconfiguration: dSHeuristics section. However, it turns out that there is a separate DSA Heuristics string in AD, which is a registry value that can be configured on each DC in the following registry subkey:
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\NTDS\Parameters

DSA Heuristics does not exist by default and to create it, there must be a new String Value with the name 'DSA Heuristics'. The DSA Heuristics registry key uses a series of bits in an array to determine whether features are enabled or not, similar to the dSHeuristics attribute. For the purposes of disabling background tasks, the bit that must be enabled is Heuristic[5], which is based on a 0 indexed array starting with Heuristic[0]. A DSA Heuristics value of '000001' should result in suppressing all background tasks, which it does, but only for the first AdminSDProtectFrequency period, after which the ProtectAdminTask is added to the queue 15 minutes later. At first, I thought DSA Heuristics may be a method to permanently prevent the ProtectAdminGroups task from running on the PDCe; however, in lab testing, this proved to not be the case. Regardless of the Windows Server OS I tried, I was unable to utilize DSA Heuristics to indefinitely pause the ProtectAdminGroups background task. Perhaps someone other than me will have better results with abusing DSA Heuristics.

Recommendation: Regularly review the AdminSDHolder security descriptor for changes. Implement proper Advanced Auditing Policy settings for AD, especially for the DCs in the Domain Controllers OU. Ship these logs off to a security information and event management (SIEM) solution or other log analytics tool and alert on changes to privileged users and groups. Also, AD replication issues can become security issues.

Bonus: I mentioned EID 4780 above. Part of the SampUpdateSecurityDescriptor() function in secadmin.c is to return fSecurityDescriptorChanged = True when the function successfully performs the DirModifyEntry() call. This, in turn, is accounted for in the SampFilterWellKnownAccounts() function after the return of the SampUpdateSecurityDescriptor() status. If the status was successful and fSecurityDescriptorChanged is false, the function continues because the security descriptor was not changed. If fSecurityDescriptorChanged is true, and AuditingEnabled then a call to LSAIAuditSamEvent() occurs, which creates an EID 4780 in the Security Event Log in modern Windows Server 2008 and newer, and either a 684 or 685 EID on previous OSes.

Figure 59 - Event ID 4780: ACL was Set on Accounts Which are Members of Administrators Groups

This audit event is not enabled by default, much like many other important events for Windows and AD. To enable it, a group policy object (GPO) must be linked to the Domain Controllers OU. That GPO should preferably have Advanced Audit Policy Configurations configured, instead of Basic Audit Policies. The Advanced Audit Policy category that generates EID 4780 is Account Management -> Audit User Account Management -> Success

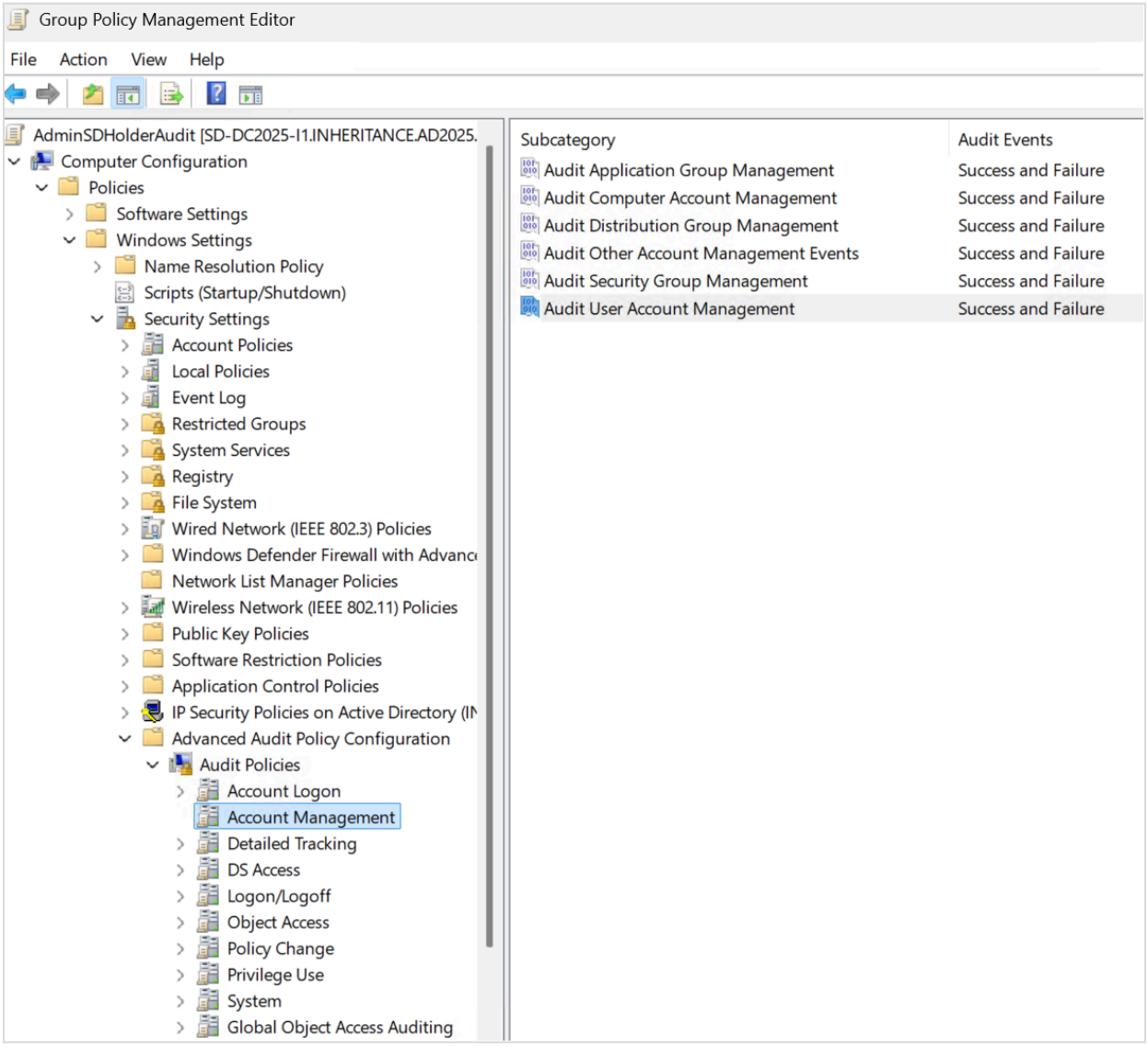


Figure 60 - Screenshot of Group Policy Management Editor Account Management Audit Policies

If, for some reason, Advanced Audit Policy Configuration cannot be utilized in your environment, then the Audit account management basic Audit Policy can be configured for Success audits to generate this event.

As the AdminSDHolder ProtectAdminGroups background task only runs on the PDC Emulator FSMO role holder, this audit event will only be found on the DC thatis the PDCe FSMO role holder.

You can find more information on this audit event here: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4780

> Every hour, the domain controller that holds the primary domain controller (PDC) Flexible Single Master Operation (FSMO) role compares the ACL on all security principal accounts (users, groups, and machine accounts) present for its domain in Active Directory and that are in administrative or security-sensitive groups and which have AdminCount attribute = 1 against the ACL on the AdminSDHolder object. If the ACL on the principal account differs from the ACL on the AdminSDHolder object, then the ACL on the principal account is reset to match the ACL on the AdminSDHolder object and this event is generated.
>
> For some reason, this event doesn't generate on some OS versions.

Figure 61 - EID 4780 Description

To the best of my knowledge, this event isn't generated on some OS versions because the 4780 event is only valid on Windows Server 2008 and newer. In Windows Server 2003 and earlier, this would be a 684 or 685 event.

Misconception: Protected ~~Group~~s Objects

This is where we start to get into the territory of misconceptions leading to misconfigurations. You may have noted that I've been putting "protected groups" in quotes so far. AdminSDHolder is not uniform on how it protects objects, which was surprising to me, but it appears to be by design.

When the boolean value fExcludeMembers is true in the structures where Windows defines the protected objects, the AdminSDHolder security descriptor is not applied to any members of the group. Both DOMAIN_GROUP_RID_CONTROLLERS and DOMAIN_GROUP_RID_READONLY_CONTROLLERS have fExcludeMembers set to true. Translating these constants to their domain relative identifiers (RIDs) can be done in MS-ADTS 2.2.15 Domain RID Values.

The gist of this is that the Domain Controllers group (RID 516) and the Read-Only Domain Controllers group (RID 521) are protected objects, but their members are not protected.

We can see how this is designed in the code, or at least how it was designed in the code for WinNT5.1. The concept is still the same in current Windows Server OSes, but the BuiltinDomainSecureAdminTable and AccountDomainSecureAdminTable structures have been updated and the code for the next misconfiguration was added.

```
3213    //
3214    // Table of well known accounts (account domain / builtin domain)
3215    //
3216    // The system will automatically apply the same security descriptor
3217    // of AdminSDHolder object on each every entry in the following table
3218    // if BOOLEAN fExcludeMembers is TRUE, we will not apply the secure SD
3219    // on members of the groups. Otherwise, include all transitive members
3220    // of each group.
3221    //
3222    //
3223
3224
3225    typedef struct _SAMP_SECURE_ADMIN_TABLE
3226    {
3227        ULONG   Rid;
3228        BOOLEAN fExcludeMembers;
3229    } SAMP_SECURE_ADMIN_TABLE, *PSAMP_SECURE_ADMIN_TABLE;
3230
3231
3232    SAMP_SECURE_ADMIN_TABLE BuiltinDomainSecureAdminTable[] =
3233    {
3234        { DOMAIN_ALIAS_RID_ADMINS, FALSE },
3235        { DOMAIN_ALIAS_RID_ACCOUNT_OPS, FALSE },
3236        { DOMAIN_ALIAS_RID_SYSTEM_OPS, FALSE },
3237        { DOMAIN_ALIAS_RID_PRINT_OPS, FALSE },
3238        { DOMAIN_ALIAS_RID_BACKUP_OPS, FALSE },
3239        { DOMAIN_ALIAS_RID_REPLICATOR, FALSE }
3240    };
3241
3242
3243    SAMP_SECURE_ADMIN_TABLE AccountDomainSecureAdminTable[] =
3244    {
3245        { DOMAIN_USER_RID_ADMIN, FALSE },
3246        { DOMAIN_USER_RID_KRBTGT, FALSE },
3247        { DOMAIN_GROUP_RID_ADMINS, FALSE },
3248        { DOMAIN_GROUP_RID_SCHEMA_ADMINS, FALSE },
3249        { DOMAIN_GROUP_RID_ENTERPRISE_ADMINS, FALSE },
3250        { DOMAIN_GROUP_RID_CONTROLLERS, TRUE }
3251    };
3252
```

Figure 62 - Screenshot of samlogon.cxx Source Code File From Leaked Windows Server 2003 RC Code

Most all of the other websites on AdminSDHolder have a list of Protected Groups or maybe Protected Groups and Accounts, but what matters is the definition in [MS-ADTS] Section 3.1.1.6.1.2 Protected Objects **and even more-so what the code actually does**. I say this because while MS-ADTS is one of the most correct sources of information on AdminSDHolder, it hasn't been updated with changes made in Server 2016 which cause Key Admins to be added to the lists of Protected Objects. The corrected addition of Enterprise Key Admins to the Protected Objects lists in Server 2019 or SAC1804 is also missing.

In AdminSDHolder, Protected Groups, and Security Descriptor Propagator, a "protected group" is defined as such:

What is a protected group?

A protected group is an Active Directory group that is identified as a privileged group. This group and all its members should be protected from unintentional modifications.

AD-DS Appendix C contains a list of "Protected Groups". Previously, this site had a handy table which is further defined as Protected Accounts and Groups in AD by OS:

Protected Groups

The following table contains the protected groups in Active Directory listed by domain controller operating system.

Protected Accounts and Groups in Active Directory by Operating System

Windows Server 2003 RTM	Windows Server 2003 SP1+	Windows Server 2012, Windows Server 2008 R2, Windows Server 2008	Windows Server 2016
Account Operators	Account Operators	Account Operators	Account Operators
Administrator	Administrator	Administrator	Administrator
Administrators	Administrators	Administrators	Administrators
Backup Operators	Backup Operators	Backup Operators	Backup Operators
Cert Publishers			
Domain Admins	Domain Admins	Domain Admins	Domain Admins
Domain Controllers	Domain Controllers	Domain Controllers	Domain Controllers
Enterprise Admins	Enterprise Admins	Enterprise Admins	Enterprise Admins
Krbtgt	Krbtgt	Krbtgt	Krbtgt
Print Operators	Print Operators	Print Operators	Print Operators
		Read-only Domain Controllers	Read-only Domain Controllers
Replicator	Replicator	Replicator	Replicator
Schema Admins	Schema Admins	Schema Admins	Schema Admins
Server Operators	Server Operators	Server Operators	Server Operators

Figure 63 - AD DS Appendix C "Protected Groups" Table

The following table is up to date and correct as of June 2025.

Security Principal	BHE Tier0	TierZeroTable	fExcludeMembers	dSHeuristics	Windows 2000 Server RTM through Windows 2000 Server SP3	Windows 2000 Server SP4 through Windows Server 2003 RTM	Windows Server 2003 SP1 through Windows Server 2012 R2	Windows Server 2016	Windows Server 2019 & Windows Server SAC1804 through Windows Server 2025
Account Operators		Y		Y	Available	Protected	Protected	Protected	Protected
Administrators	Y	Y			Available	Protected	Protected	Protected	Protected
Backup Operators		Y		Y	Available	Protected	Protected	Protected	Protected
Cryptographic Operators		Y			N/A	N/A	N/A	Available	Available
Performance Log Users		N			N/A	N/A	Available	Available	Available
Print Operators		Y?		Y	Available	Protected	Protected	Protected	Protected
Replicator		-			Available	Protected	Protected	Protected	Protected
Server Operators		Y		Y	Available	Protected	Protected	Protected	Protected
Administrator	Y	Y			Protected	Protected	Protected	Protected	Protected
Allowed RODC Password Replication Group		N			N/A	N/A	N/A	Available	Available
Cert Publishers		Y			Available	Protected	Available	Available	Available
Denied RODC Password Replication Group		N			N/A	N/A	N/A	Available	Available
DnsAdmins		Y			Available	Available	Available	Available	Available
Domain Admins	Y	Y			Protected	Protected	Protected	Protected	Protected
Domain Controllers	Y	Y	Y		Available	Protected	Protected	Protected	Protected
Enterprise Admins	Y	Y			Protected	Protected	Protected	Protected	Protected
Enterprise Key Admins	Y	Y			N/A	N/A	N/A	Available	Protected
Enterprise Read-only Domain Controllers		N			N/A	N/A	N/A	Available	Available
Group Policy Creator Owners		N			Available	Available	Available	Available	Available
Key Admins	Y	Y			N/A	N/A	N/A	Protected	Protected
krbtgt		Y			Available	Protected	Protected	Protected	Protected
Protected Users		-			N/A	N/A	N/A	Available	Available
Read-only Domain Controllers		N	Y		N/A	N/A	N/A	Protected	Protected
Schema Admins	Y	Y			Protected	Protected	Protected	Protected	Protected

Figure 64 - Protected Objects Table

Table key:

- 'Available' denotes that the group or security principal exists in that version of the Windows Server OS and, as such, it is part of the AD forest if the group is in the root domain of the forest or the group scope is Universal or part of the AD domain for a non-root domain or if the group scope is Global or Local, regardless of whether the DC of that OS holds a specific FSMO role

- 'Protected' denotes that the group or security principal local to that domain, through transitive membership or not, is a protected object when the PDCe FSMO role holder for the domain is running that specific version of the Windows Server OS.
- 'Available' in red font color denotes a Tier Zero object, as defined by the TierZeroTable, that is not protected when the PDCe FSMO role holder of that domain is running the specified Windows Server OS.
- The TierZeroTable column references data from this website: https://specterops.github.io/TierZeroTable/
- The dSHeuristics column references data from the next section of this paper.

A full reference of AdminSDHolder protections across all OS levels is gathered here: https://github.com/JimSycurity/AdminSDHolder/blob/main/AdminSDHolderTruthMatrix.xlsx

A note on nested membership:

> Note Membership in a protected group is defined as either direct membership or transitive membership using one or more security or distribution groups. Distribution groups are included because they can be converted to security groups.

Figure 65 - KB318180: AdminSDHolder Thread Affects Transitive Members of Distribution Groups

It may come as a surprise that the membership of distribution groups is accounted for in the AdminSDHolder process, as explained in KB318180, but this has been the case in Windows Server versions beyond Windows 2000 RTM. Transitive membership in a privileged group, such as Domain Admins, via a distribution group does not convey any rights or privileges associated with that privileged group. The Domain Admins SID, for example, is not in the access token of any members of a distribution group nested within Domain Admins. Members nested in Domain Admins via a distribution group are protected all the same.

Per my current understanding, this is the only instance in AD where a non-security group is treated the same as a security group. But as the KB article says, any distribution group can be converted to a security group.

Recommendation: Establish a design of a secure OU structure for Tier Zero assets and security principals where inheritance from the domain root is disabled. Extend that design concept to the Domain Controllers OU so that the AD Computer Objects for Domain Controllers and Read-Only Domain Controllers are protected from default or inherited permissions. Scrutinize existing DC and RODC AD Computer Objects for non-standard object ownership issues and overly permissive DACLs.

Misconfiguration: dSHeuristics

Speaking of Protected Objects, it's been possible since the installation of KB327825 (New resolution for problems with Kerberos authentication when users belong to many groups) and/or Windows Server 2004 SP4 to utilize dsHeuristics to exclude a subset of the default protected objects, reverting to a closer format of the Protected Objects that was included with Windows Server 2000 RTM. Account Operators, Server Operators, Print Operators, and Backup Operators can be excluded from AdminSDHolder protection, as was mentioned above in **Misconception: Protected Groups Objects** and also detailed in KB817433.

I'm not certain on the history of WHY Microsoft made so many changes to AdminSDHolder specifications in the Windows Server 2000 era, but AD was a very new technology at the time and methods to protect and attack it evolved very quickly early on. I would guess that the additional Protected Objects from Windows Server 2000 RTM to Windows Server 2000 SP4 was based on customer demand to secure more privileged objects. And the additional protected objects make sense. As for why the dsHeuristic option to disable the protection of these four groups was introduced, my gut instinct is that it relates to Microsoft Exchange and being able to mail-enable "lesser" privileged accounts.

It's Exchange All the Way Down. AdminSDProp?!

A problem here is that every one of these four groups has privilege escalation paths in AD by default (in most environments).

Account Operators (Bit 0) was designed to be able to manage users, computers, and groups in AD. So, in many environments it is treated as a Help Desk group. An issue with this is that Account Operators principal is granted a default explicit ACE on the DACL of any user or computer account in AD when it is created. This is not an inherited permission, so it doesn't flow down from parent objects. Certainly, Account Operators can't modify Protected Objects where AdminSDHolder is pulling its weight, but it can modify VIP accounts (CEO, CFO, HR), the computer objects that IT admins are likely using to perform AD administration if Privileged Access Workstations (PAWs) are not enforced, and also the AD Objects that AdminSDHolder doesn't protect (See **What AdminSDHolder Doesn't Do**). This is enough in most any environment to escalate privileges from a "lesser" help desk account to something critical.

Print Operators (Bit 2), by default, have the ability to log on locally (SeLogonPrivilege) to DCs and load and unload device drivers (SeDriverPrivilege) on DCs. Hosting a print queue or running the print spooler service on a DC is a bad idea. Print spooler running on a DC allows for authentication coercion. Printer code in Windows is what I would call legacy code and it has a large attack surface, ala PrintNightmare, and the nightmare isn't necessarily over yet.

Backup Operators (Bit 3), by default, have the rights assignment to backup (SeBackupPrivilege) and restore (SeRestorePrivilege) files. In the case of AD, those files are the AD DS datastore on disk: NTDS.dit and SYSTEM. And it extends to backing up objects inside of the AD datastore as well. If an attacker can back up the NTDS.dit and SYSTEM files, they can extract all the information from those files, including the secret credentials. The result is a complete AD forest compromise. Additionally, Backup Operators are granted an Allow Read ACE on the winreg registry key, which allows for read access via Remote Registry.

Server Operators (Bit 1), by default, also have the rights assignment to backup (SeBackupPrivilege) and restore (SeRestorePrivilege). On top of that they are also able to log on locally to DCs, start and stop services on DCs, manage network shares on DCs, shut down the DC locally or remotely, and change the system time.

These built-in "lesser" privileged groups are not lesser at all. They shouldn't be used at all, in favor of targeted least-privilege delegations instead.

KB817433 includes several different workaround options for dealing with these four built-in groups as it relates to their new (as of Server 2003 or Server 2000 SP4) inclusion as Protected Objects. All of these workarounds are awful in their own special way.

Recommendation: Leave the dsHeuristics setting default or remediate the misconfiguration if it exists. If the Account Operators, Server Operators, Print Operators, and Backup Operators groups are utilized in the environment, consider a gradual AD administration redesign such that:

- Help Desk staff have least-privilege delegated accounts instead of Account Operators membership. Preferably delegated as part of a well-designed OU structure.
- Tier 1 member server admins have least-privilege delegated accounts that are members of the local administrator groups on specific servers via the use of GPO Protected Groups or Local Users and Groups assignments rather than being members of Server Operators. Preferably delegated as part of a well-designed OU structure.
- Stop using printers 😊. Seriously though, the Printer Operators group should not be used. The Print spooler service should not be running on any DC or other Tier Zero asset. Don't let SpoolSample turn your domain into a PrintNightmare!
- Delegate least-privilege access for backup software service accounts that applies directly to the server and computer objects that need to be backed up. This can be done with an OU structure that allows for a GPO with Protected Groups or Local Users and Groups settings to add respective backup service accounts to the local Backup Operators group on individual member servers. Backup Operators may be required for the completely separate Tier Zero backup service account used to properly back up the Active Directory DCs, and more importantly the Forest Partition Data.

Bonus: What's the dsHeuristic value in your domain? Try this PowerShell one-liner:

```
# Query Current dSHeuristics in current AD Domain
(Get-ADObject ("CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration," +
(Get-ADDomain).DistinguishedName) -Properties dSHeuristics).dSHeuristics
```

Misconfiguration: Change AdminSDHolder Interval

By default, the AdminSDHolder (not the SDProp) Background Task runs in the LSA subsystem service (LSASS) on the PDC Emulator DC every 60 minutes. This can be modified by creating or modifying the AdminSDProtectFrequency DWORD in HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\NTDS\Parameters. The range of values is in seconds from 60 to 7200. Deleting the AdminSDProtectFrequency key causes the PDCe to revert back to the default of 60 minutes.

If the AdminSDProtectFrequency DWORD value doesn't exist, the default of 60 minutes (3600 seconds) is used. A value of 600 would cause AdminSDHolder Background Task to run in LSASS on the PDCe every 10 minutes. The most likely outcome of reducing the frequency of this background task is increasing LSA processing overhead or possibly even crashing LSASS on the most important DC in your domain. Leave it default and if you absolutely need AdminSDHolder to run at a shorter duration, run it manually for that specific instance of time.

Recommendation: My favorite Uncle used to say "Here's Lever A and here's Lever B. Now Lever B!"

Misconfiguration: Enabling Inheritance on AdminSDHolder

This isn't as much of a misconception as it is a misconfiguration. I haven't been able to determine the exact logic behind scenarios where DACL inheritance has been enabled on the AdminSDHolder object, but I have noted it in a couple of very large environments.

Enabling DACL inheritance removes the Protected DACL bit on the security descriptor of the object. This enables the object's security descriptor to inherit permissions from parent objects.

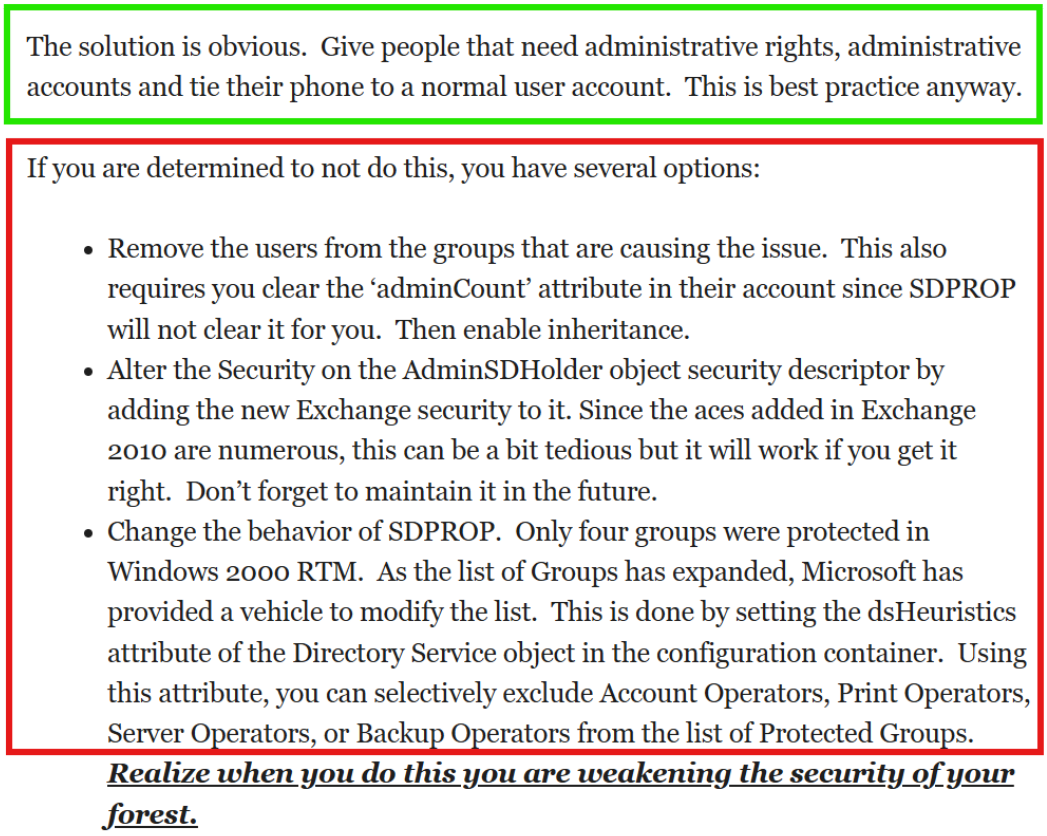
As the System container where AdminSDHolder resides allows inheritance of permissions from domain root, the AdminSDHolder object would then inherit those permissions. Ideally, the domain root permissions would be clean and secure. In practice, they rarely ever are.

Ultimately, the inherited permissions propagated to the AdminSDHolder object itself from the domain root and the System container are largely inconsequential as they are never actually applied to any protected objects for any discernible length of time. Only the explicit ACEs matter for this purpose.

What matters most is the effective permissions on every protected object. The protected objects will inherit permissions from whatever parent object(s) are in their tree, wherever that may be. In many cases, they'd inherit permissions from the domain root and any other OUs along the way. This results in a scenario where it is unlikely that any protected object will have a propagated DACL that matches that of the AdminSDHolder object. The data I've collected and the audit events generated in my lab back up this notion.

And yet, for nearly as long as Microsoft Exchange has been fully integrated into AD, there have been Microsoft Exchange blogs which recommend enabling inheritance on AdminSDHolder along with other misconceptions and misconfigurations to solve a variety of email woes.

When Exchange 2010 changed how ActiveSync functioned under the hood from being a user attribute to child objects of the user, this caused several blogs to come out with recommendations that strongly recommended using separate administrator accounts from the regular mail-enabled user (good), and also included the workarounds that lead to misconfigurations (bad).


The solution is obvious. Give people that need administrative rights, administrative accounts and tie their phone to a normal user account. This is best practice anyway.

If you are determined to not do this, you have several options:

- Remove the users from the groups that are causing the issue. This also requires you clear the 'adminCount' attribute in their account since SDPROP will not clear it for you. Then enable inheritance.
- Alter the Security on the AdminSDHolder object security descriptor by adding the new Exchange security to it. Since the aces added in Exchange 2010 are numerous, this can be a bit tedious but it will work if you get it right. Don't forget to maintain it in the future.
- Change the behavior of SDPROP. Only four groups were protected in Windows 2000 RTM. As the list of Groups has expanded, Microsoft has provided a vehicle to modify the list. This is done by setting the dsHeuristics attribute of the Directory Service object in the configuration container. Using this attribute, you can selectively exclude Account Operators, Print Operators, Server Operators, or Backup Operators from the list of Protected Groups.

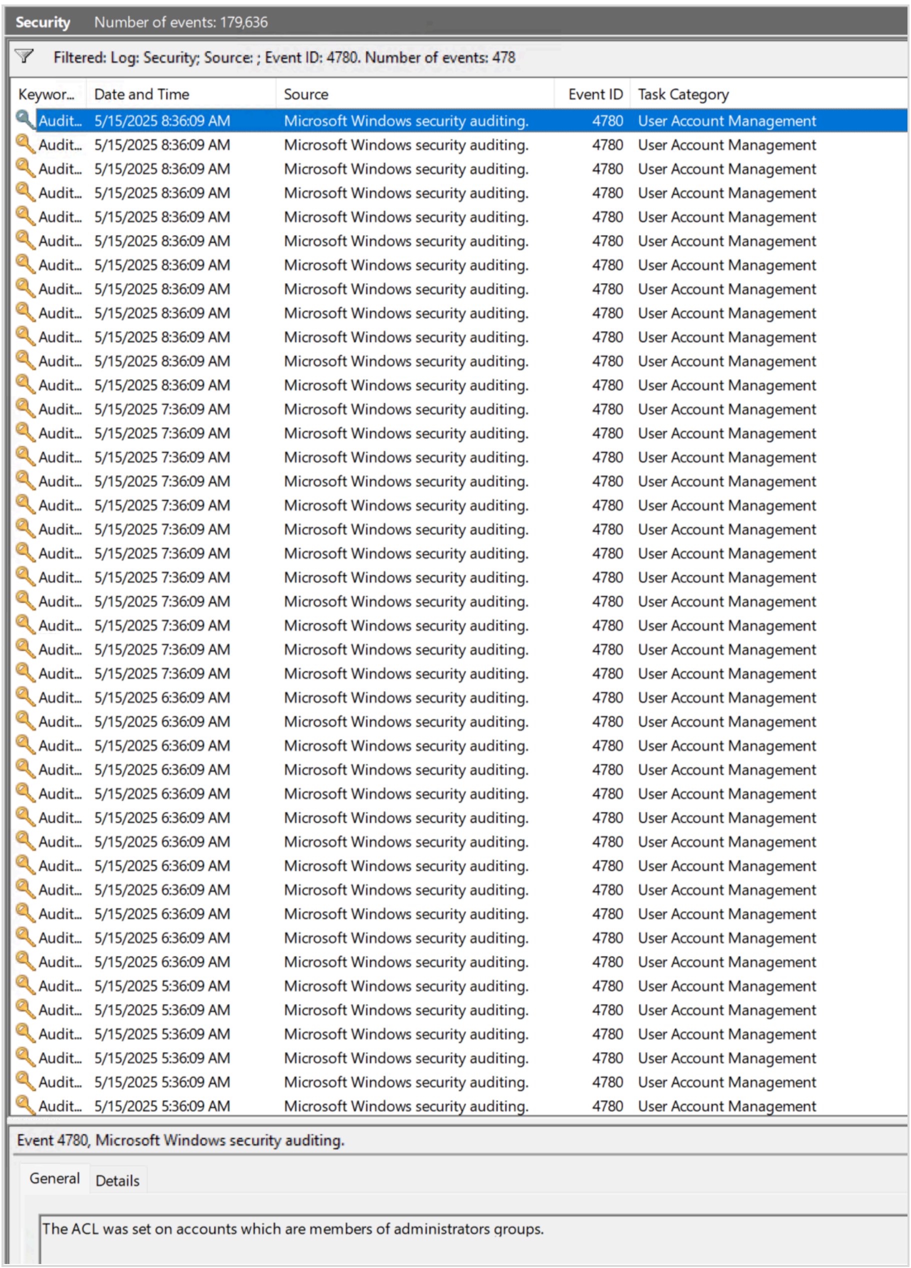
Realize when you do this you are weakening the security of your forest.

Figure 66 - Screenshot From https://musingsoftheproletariat.com/exchange-2010-phones-and-adminsdholder/

Basically, enabling inheritance on AdminSDHolder defeats almost the entire point of the AdminSDHolder object and process. Unfortunately, in an informal poll of environments, SpecterOps found that nearly **21% of the environments** had one or more AD domains with inheritance enabled on the AdminSDHolder object for that domain.

When inheritance is enabled on AdminSDHolder, one certain outcome is that the security descriptor of the protected objects will reset every 60 minutes by default as the security descriptor of objects in a different OU or container structure will have different inherited permissions than the AdminSDHolder object in the System container.

For example, here is the Security event log on SD-DC2025-I1 in the *inheritance.ad2025.lan* domain of my lab where AdminSDHolder has inheritance enabled. The log view is filtered to show only EID 4780. Note that every hour, the ProtectAdminGroups background task re-stamps the security descriptor of multiple objects and, by the total count of 4780 events, this happens consistently and constantly.



Security Number of events: 179,636

Filtered: Log: Security; Source: ; Event ID: 4780. Number of events: 478

Keywor...	Date and Time	Source	Event ID	Task Category
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 8:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 7:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 6:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 5:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 5:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 5:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 5:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 5:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management
Audit...	5/15/2025 5:36:09 AM	Microsoft Windows security auditing.	4780	User Account Management

Event 4780, Microsoft Windows security auditing.

General Details

The ACL was set on accounts which are members of administrators groups.

Figure 67 - A Series of Unfortunate 4780 Event Log Entries

In a large enough environment where there are many sites and a significant amount of protected objects, this could even potentially cause hourly replication storms.

Enabling inheritance on the DACL of AdminSDHolder creates a scenario similar to the troubleshooting article '[A batch of Event ID 4780 are logged in the primary domain controller]'. Except instead of for audit ACEs, it's for security ACEs. The audit ACEs issue in the troubleshooting document is understandable as inheritance is enabled by default on the SACL of AdminSDHolder. Inheritance is not enabled by default for the AdminSDHolder DACL.

I've heard as justification for having DACL inheritance enabled on AdminSDHolder that it used to be enabled by default. I have spun up AD forests of every possible production Windows Server OS for DCs, and even several release candidates, in my lab. Not a single version of Windows Server produces a default configuration where AdminSDHolder has DACL inheritance enabled. I've included data captures from my lab in my GitHub here: https://github.com/JimSycurity/AdminSDHolder in the OS Levels folder.

And even if it were true that DACL inheritance was enabled by default at one point in the past, that was the past. We shouldn't continue making poor choices just because they were once assumed to be good.

Replication storms aren't great, but what about attack paths? Let's consider the *ad2025.lan* forest in my lab, with the child domain *inheritance.ad2025.lan* where inheritance is enabled on the AdminSDHolder object.

- For the root domain in the forest, the AdminSDHolder object has a default security descriptor, which will protect the root of the forest quite well, right?

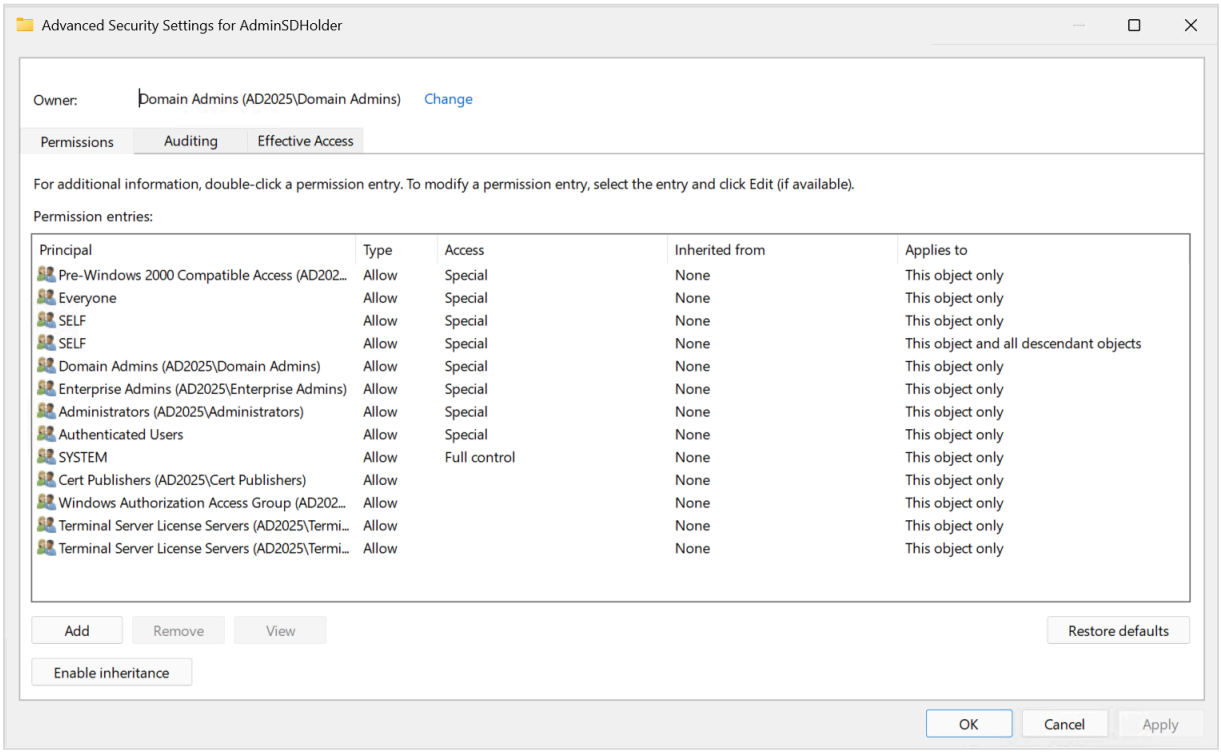


Figure 68 - Advanced Security Settings for AdminSDHolder

- In the *inheritance.ad2025.lan* child domain, the AdminSDHolder object has DACL inheritance enabled and looks more like this:

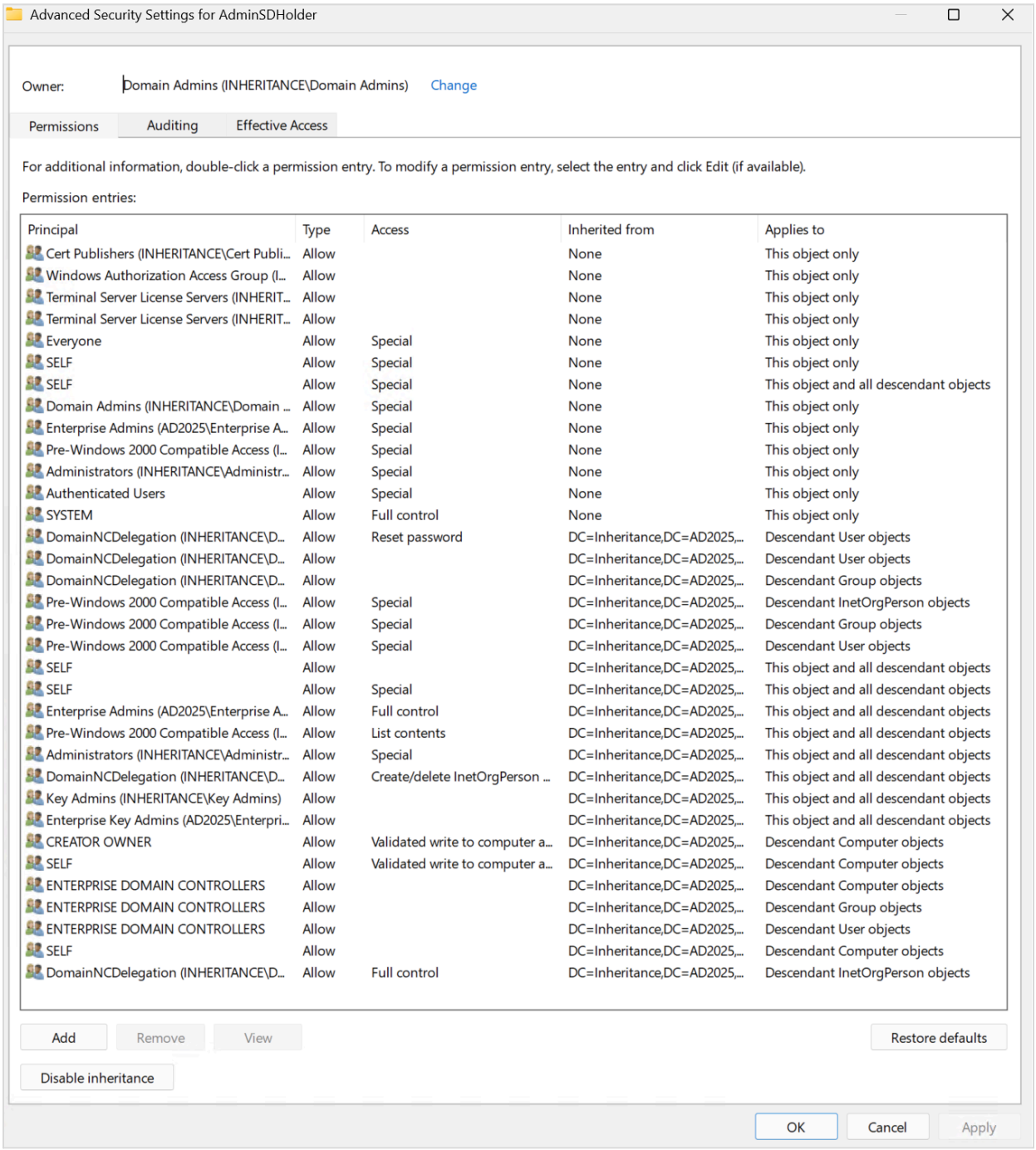


Figure 69 - Screenshot of Advanced Security Settings for AdminSDHolder

- No worries; we thought ahead and created a Tiered OU structure where the Tier Zero OU has DACL inheritance disabled:

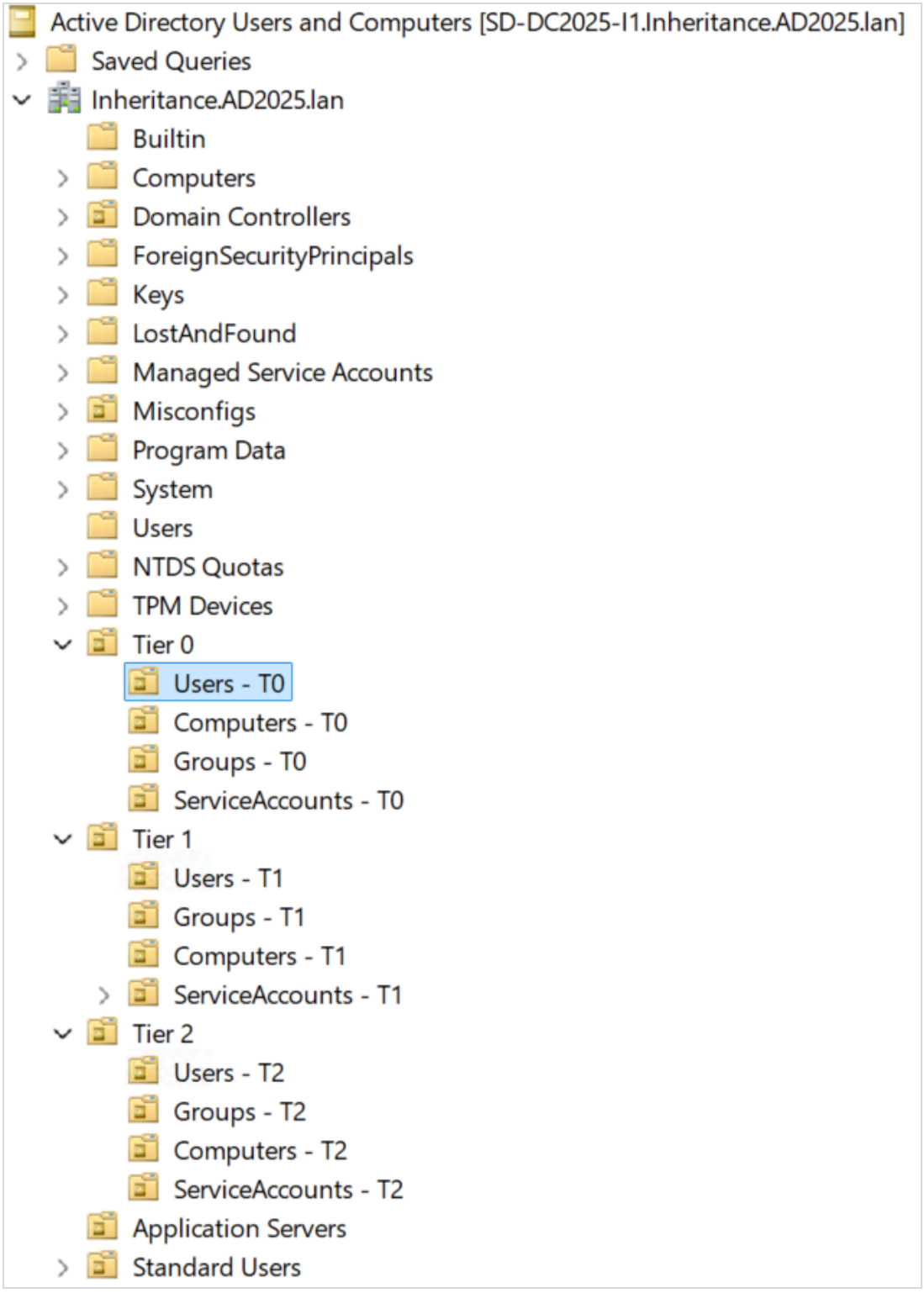


Figure 70 - Screenshot of ADUC OU Hierarchy

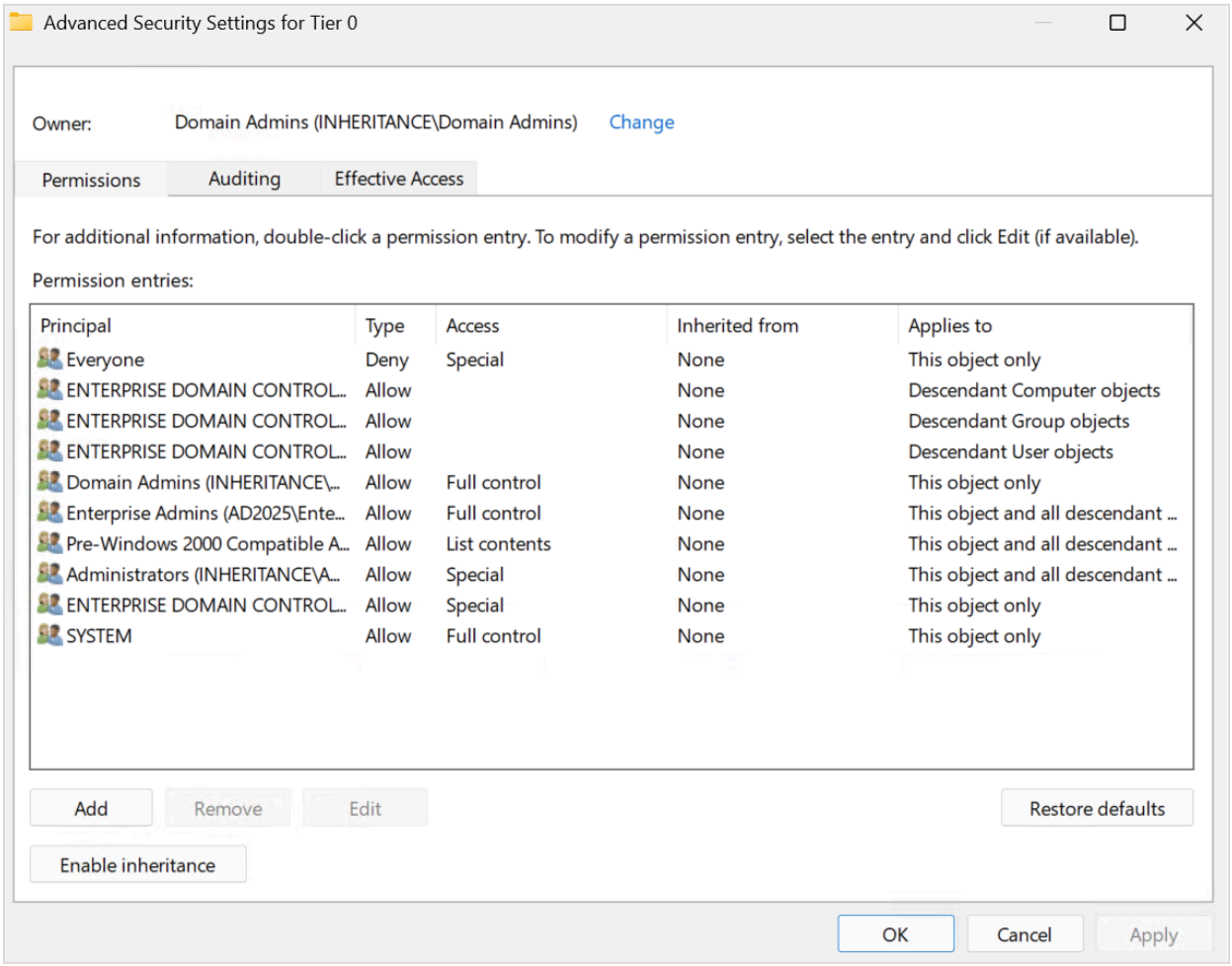


Figure 71 - Screenshot of Advanced Security Settings for Tier Zero OU

- This results in a security descriptor on the Domain Admins that are placed in the 'Users - T0' OU which isn't of any more risk than a Domain Admin in the *ad2025.lan* parent domain would be, even with DACL inheritance enabled:

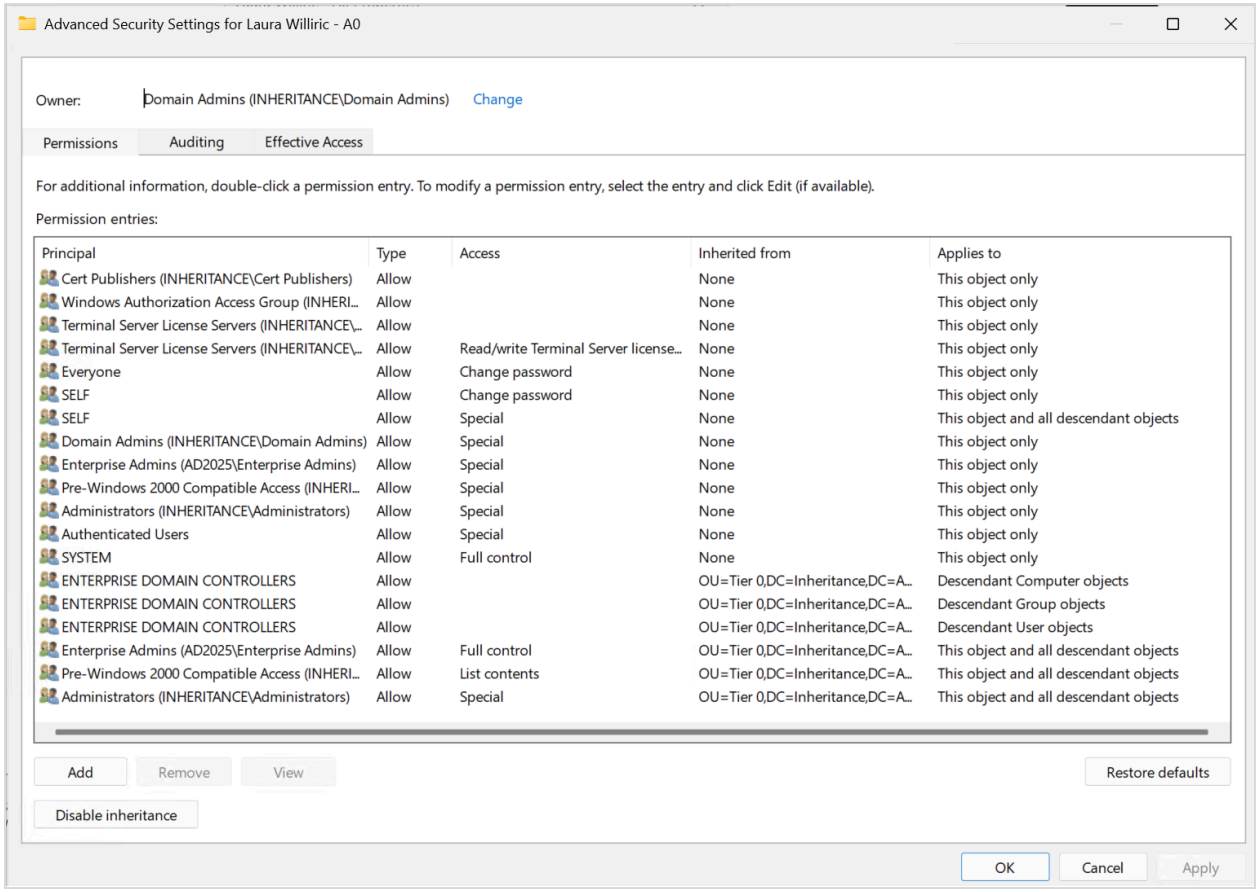


Figure 72 - Screenshot of Advanced Security Settings for Laura Williric - AO User

- To protect Tier Zero from attack paths, we can just move all of our Tier Zero objects to the Tier Zero OU so they're all protected, right? We would've gotten away with it too, if it wasn't for those pesky systemFlags:

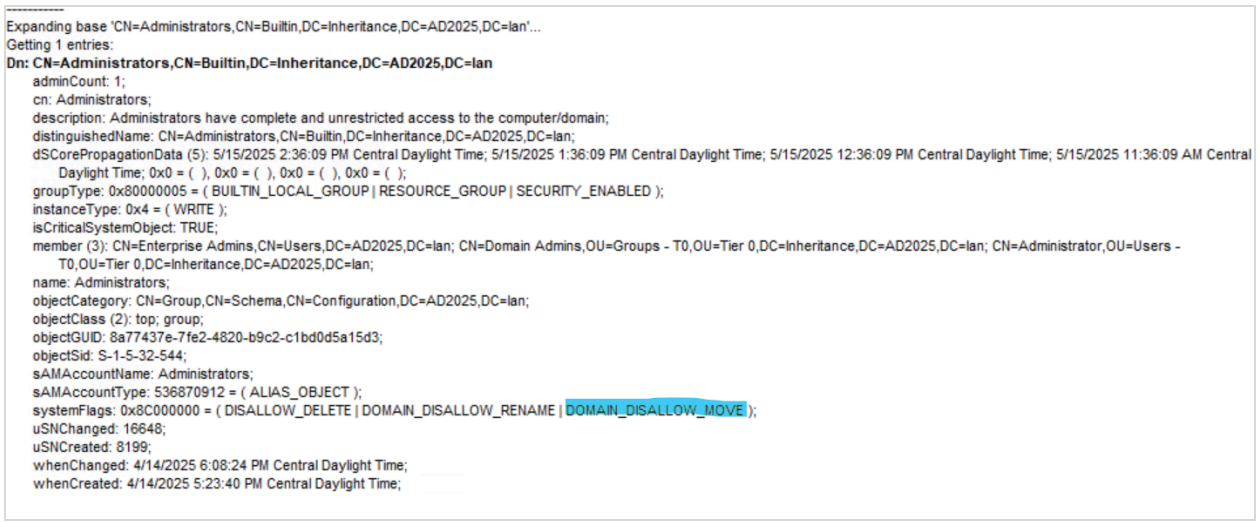


Figure 73 - Screenshot of Administrators Group Highlighting DOMAIN_DISALLOW_MOVE systemFlag

- We remembered to disable DACL inheritance on the Builtin container in addition to our Tier Zero OU, right? Nope:

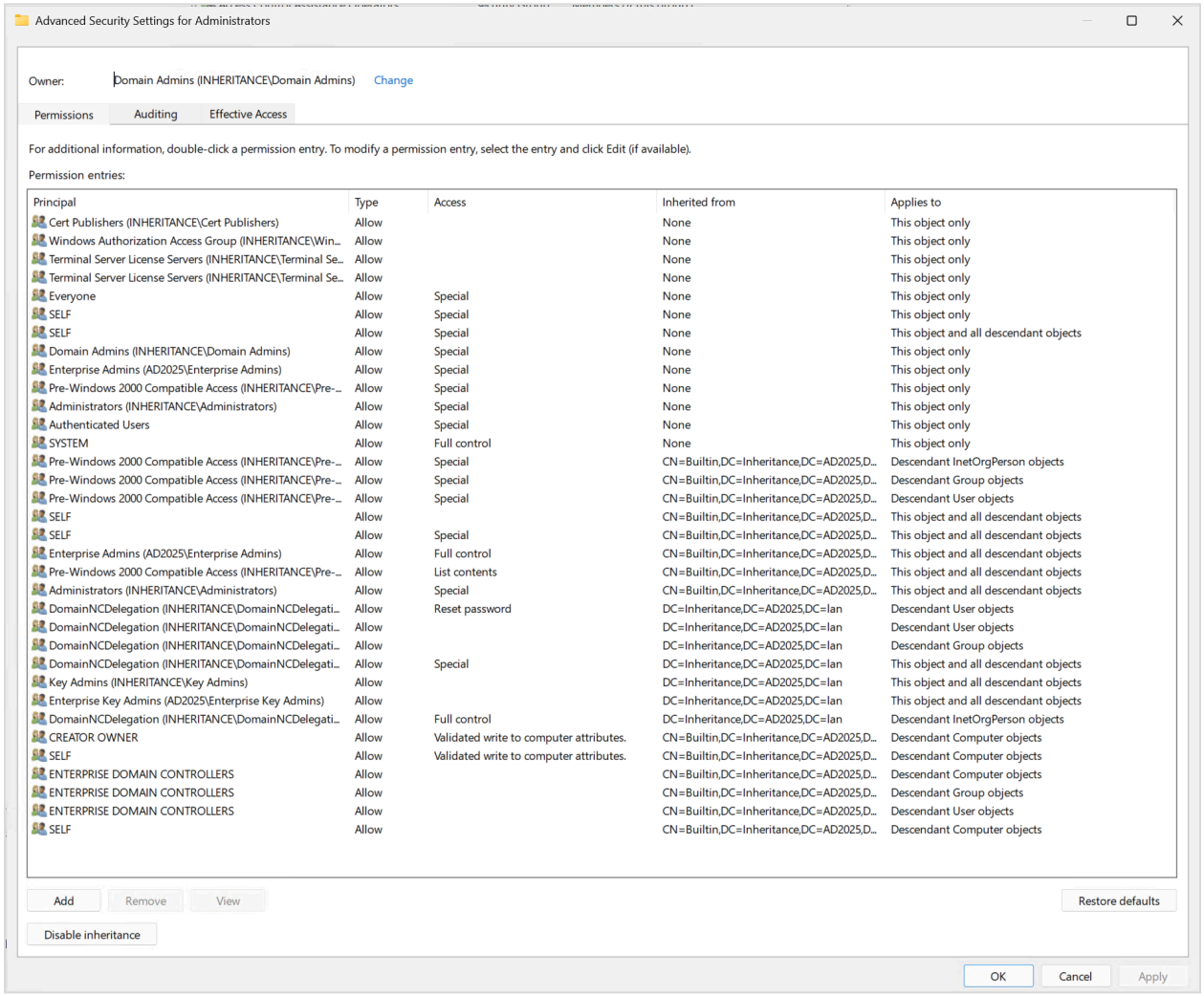


Figure 74 - Screenshot of Advanced Security Settings for Administrators Group

- Without getting everything perfect, there are likely some attack paths that wouldn't exist if inheritance was disabled. For example, this attack path from the standard user *Jane Doe* to the Builtin Administrators group:

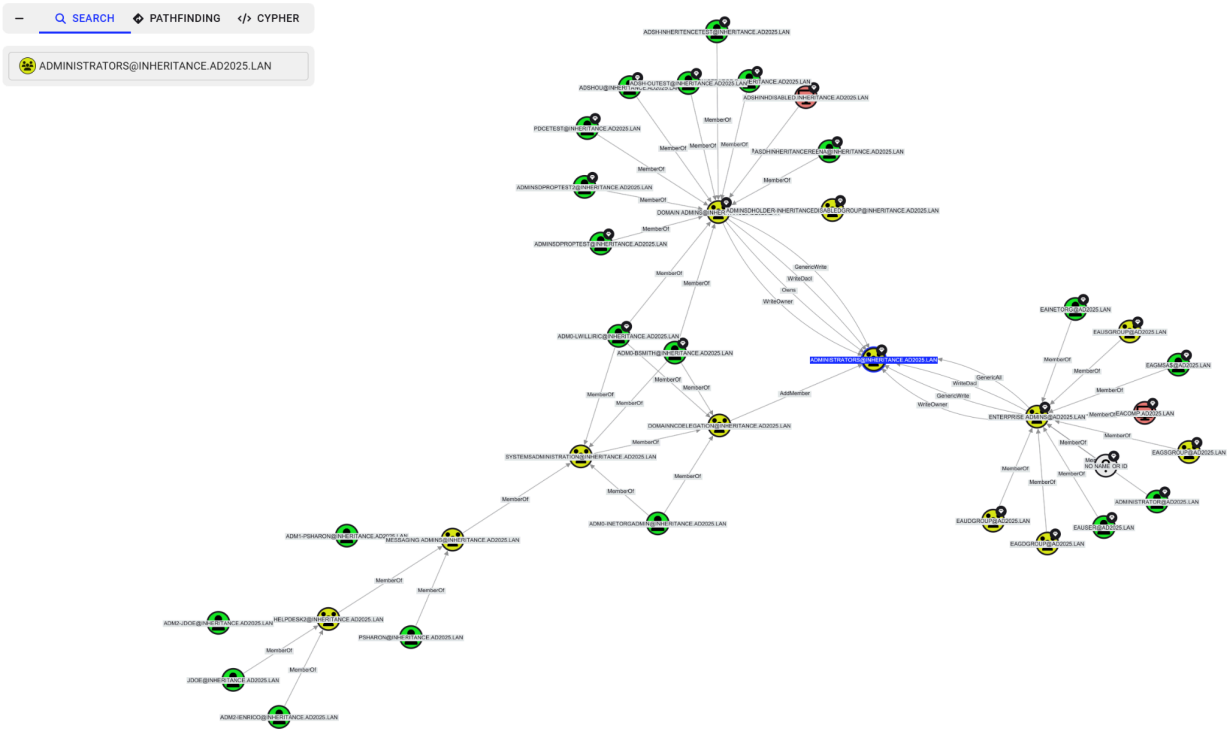


Figure 75 - Screenshot of BloodHound Attack Graph

- Let's set aside the fact that the forest is the security boundary in AD for a moment. At the start of this exercise, I pointed out that the parent domain in this forest, *ad2025.lan,* is quite clean. And so compromising the entire forest should require some complicated attack path involving detection-ripe techniques, right?

Figure 76 - Screenshot of BloodHound Attack Graph

- Not exactly...

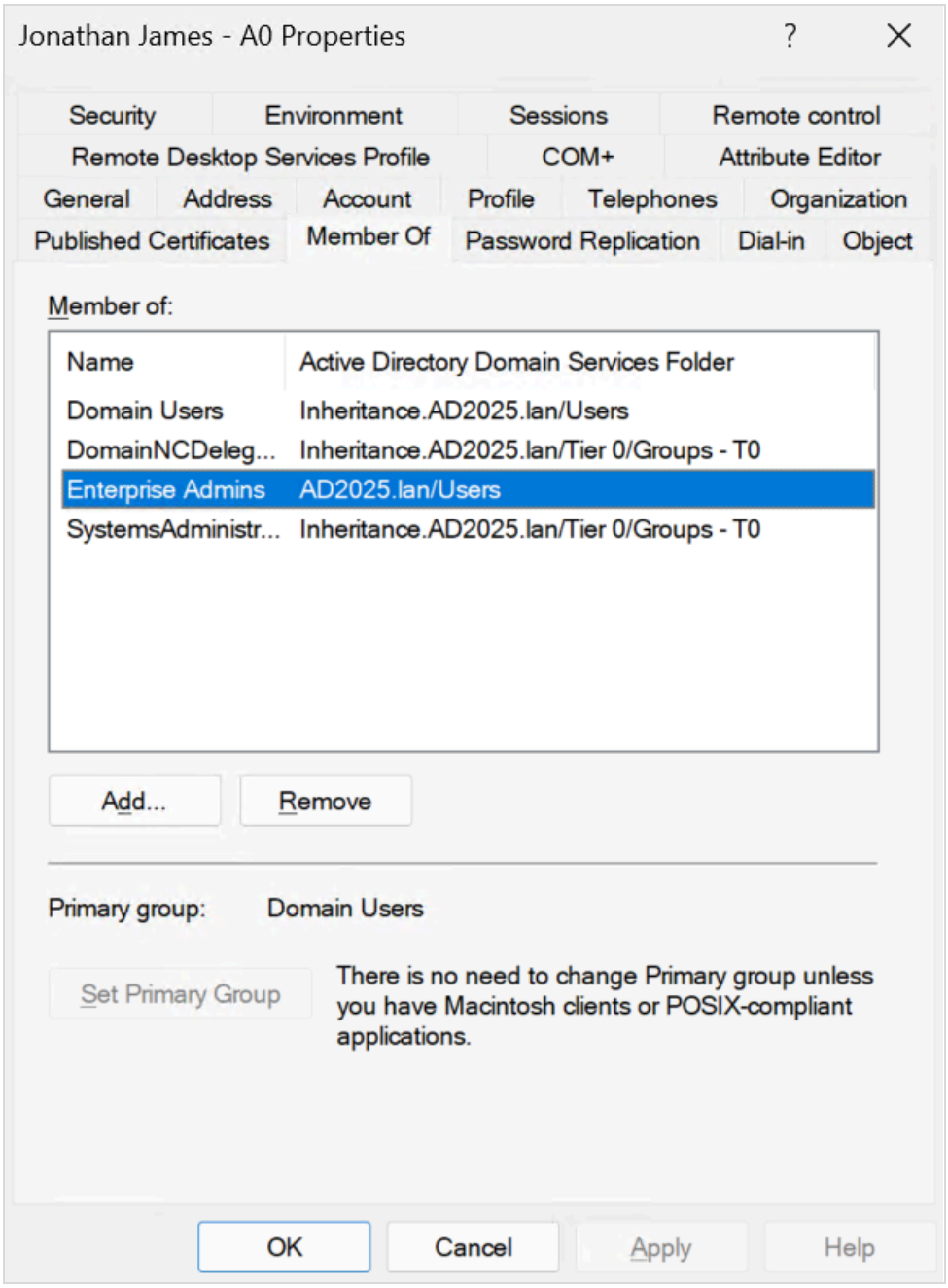


Figure 77 - Screenshot of Jonathan James - A0 Membership Properties

Of course, this is a lab environment and a contrived experiment. However, it is based directly on configurations I have noted in production environments during my career.

I collected some extra data from the *inheritance.ad2025.lan* domain and placed it in GitHub. It was interesting to note that, based on the data I collected while DACL inheritance was enabled on AdminSDHolder (InheritanceTest02-AdminSDHolder.xlsx), there are 11 distinct security descriptors

across the 24 protected objects in that domain. Additionally, there is a SharpHound collection for this domain for the same DACL inheritance enabled state.

After collecting the data with DACL inheritance enabled, I modified the AdminSDHolder security descriptor again to disable DACL inheritance then collected data again. As expected, the security descriptors of the 24 protected objects in the domain are identical. Another SharpHound collection was performed in this more secure state as well. And sure enough, the attack path to and through the Builtin Administrators groups is no longer possible:

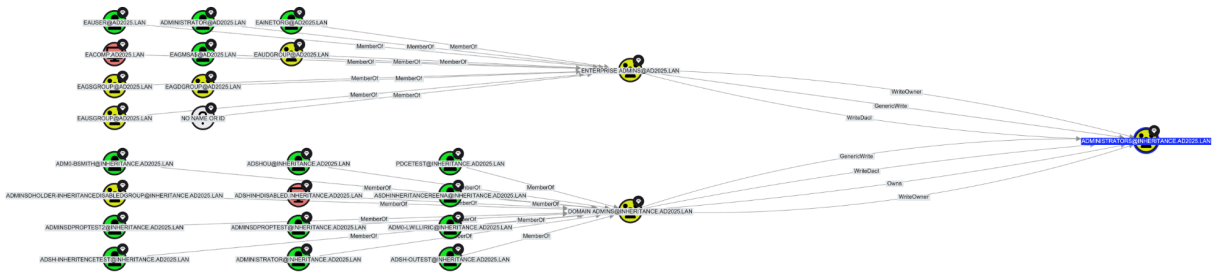


Figure 78 - BloodHound Attack Graph

Unfortunately, there is still an attack path through Domain Admins in its stead. However, there are multiple options to remediate this attack path, none of which have to do with AdminSDHolder.

Figure 79 - BloodHound Attack Graph

Recommendation: Do not enable inheritance on the AdminSDHolder object. This will significantly decrease the security of the entire AD environment in all but the most pristine and perfectly tiered environments, which are exceedingly rare.

If you're unsure if inheritance is enabled on AdminSDHolder in your environment, here is a BloodHound cypher query you can use to determine the current configuration:

```
MATCH (x:Container)
WHERE x.distinguishedname STARTS WITH "CN=ADMINSDHOLDER,CN=SYSTEM,DC="
AND x.isaclprotected = False
RETURN x
```

If inheritance is enabled on AdminSDHolder in your environment, I would love to understand the whys around this.

- Is this a recent change configured for a specific purpose? If so, what was that intended purpose?
- Is this a change that administrators who are no longer with the organization made in the past? If so, did they perhaps document why they made the change?
- Are you aware of any attack paths in the environment that this configuration is the root cause for?
- Have you identified any violations of the Clean Source Principal in the environment due to this configuration?

Caution: If inheritance is enabled on AdminSDHolder in your environment, I would recommend determining the impact of reverting AdminSDHolder to the more secure default SD_Protected state prior to making the change. Data that should be collected includes which inherited ACEs on existing protected objects will be removed if inheritance is re-disabled. PowerShell can help with this.

Misconception: It's OK if Microsoft Changes the AdminSDHolder's DACL, Right?

This is a bit of an extension of and addition to the **Just Change AdminSDHolder's DACL** misconception above. In that misconception section I highlighted how it's usually not a good idea to make your own changes to AdminSDHolder's security descriptor.

But what if Microsoft changes it for you? How bad could it be if the Microsoft Exchange or Microsoft Entra Connect (formerly Azure AD Connect sync) installers or utilities make changes to the AdminSDHolder DACL?

It depends™. As noted in [Just Change AdminSDHolder's DACL], there have been some mistakes on Microsoft's part. But that's probably the exception, not the rule?

There are also some common misconfigurations around the Entra Connect AD DS Connector account. Some of these are design choices with older versions of Azure AD Connect sync that made changes to AdminSDHolder by default. Some of these are choices hybrid identity administrators made when

installing the Azure AD Connect sync software or provisioning the environment for custom installs. Some of these are choices made while using Microsoft's own PowerShell ADSyncConfig.psm1 modules with the -IncludeAdminSdHolders switch (not recommended).

Adding the -IncludeAdminSdHolders switch (not recommended, but included in the code examples) to either of the cmdlets below will grant that Entra Connect AD DS Connector account rights to either change the membership of AD Admin groups or reset the password of AD Admins:

```
Set-ADSyncUnifiedGroupWritebackPermissions -ADConnectorAccountName
MSOL_8675309 -ADConnectorAccountDomain contoso.com -IncludeAdminSdHolders

Set-ADSyncPasswordWritebackPermissions -ADConnectorAccountName
MSOL_8675309 -ADConnectorAccountDomain contoso.com -IncludeAdminSdHolders
```

That's not great if someone compromises your Entra Connect server(s) or service accounts, but if Password Hash Synchronization is enabled, it's probably the least of your worries.

What about Microsoft Exchange? Well, I generally consider Microsoft Exchange to be a Tier Zero or near-Tier Zero asset when it's configured in the default Shared Permissions model vs the Split permissions model. There's a lot of reasons for this, all relating to how Exchange integrates itself into the AD schema and the permissions it grants itself, by default. It doesn't help that there's a seemingly continuous stream of Exchange security vulnerabilities that require patching of a notoriously hard-to-patch infrastructure.

But here we're talking about permissions, specifically those on the AdminSDHolder container. I regularly see configurations where Exchange Trusted Subsystem or Exchange Enterprise Servers are granted WriteProperty permissions on the AdminSDHolder container for the property sets Public-Information and Personal-Information along with the attribute servicePrincipalName.

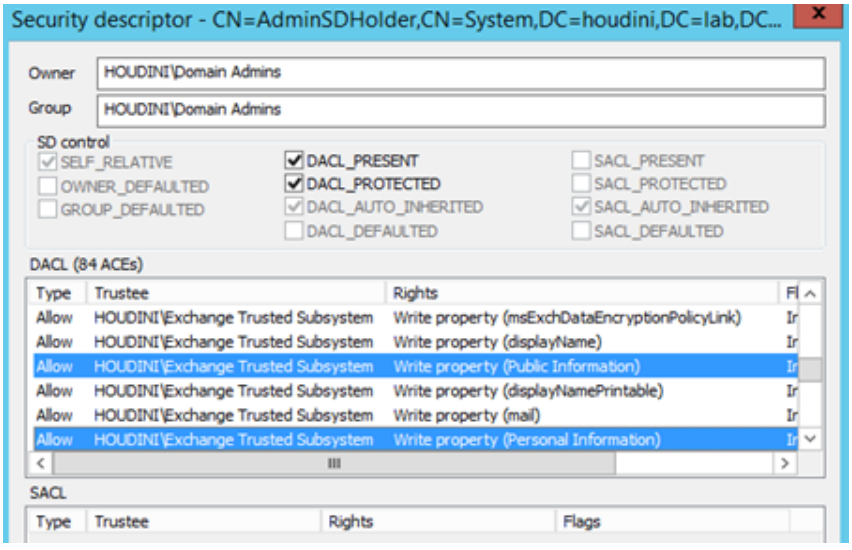


Figure 80 - Screenshot of Security Descriptor of AdminSDHolder

Public-Information and Personal-Information? How bad could that be? Well, first we need to understand what a Property Set is. In AD, a Property Set is a set of related attributes that can be specified instead of individual attributes when defining a security descriptor in order to grant or deny access to all of the attributes in the Property Set via an ACE.

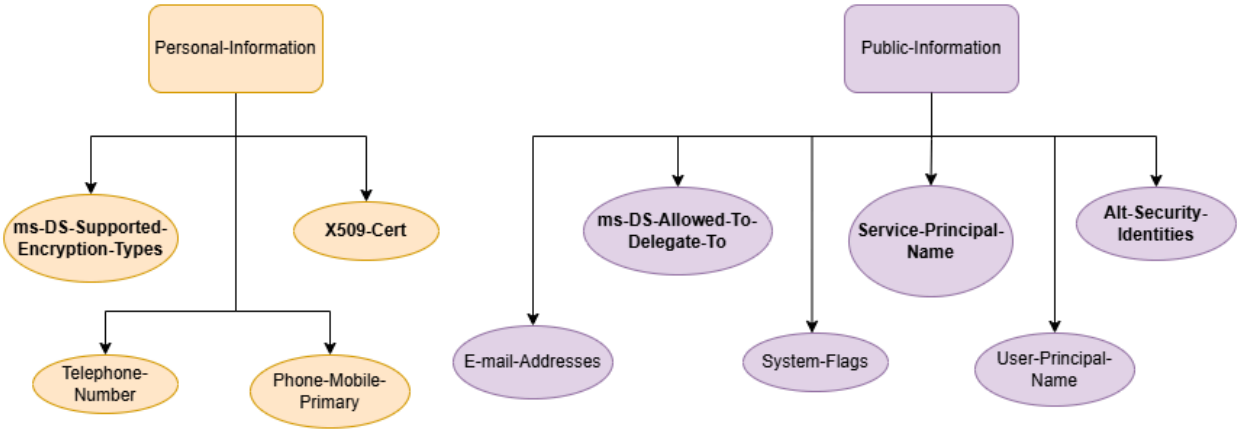


Figure 81 - Interesting Attributes of Personal-Information and Public-Information Property Sets

Property sets are required for the complicated and flexible nature of AD object ACEs as adding individual ACEs for every individual attribute would quickly balloon the size of the DACL in the security descriptor and potentially to the point of reaching the 64 Kb maximum DACL size.

However, lumping properties together in sets can cause downsides as well because the only way to allow everything in a property set without allowing a few specific attributes is to combine deny ACEs on the individual attributes with the allow ACE for the property set. Using targeted deny ACEs is the approach that Microsoft took with the June 2019 Quarterly Release to remediate the Exchange privilege escalation issues presented in Write Public-Information ACE leads to Kerberoasting from Exchange security groups and Public-Information property set includes Alt-Security-Identities, allows x509 certificate mapping to privileged users.

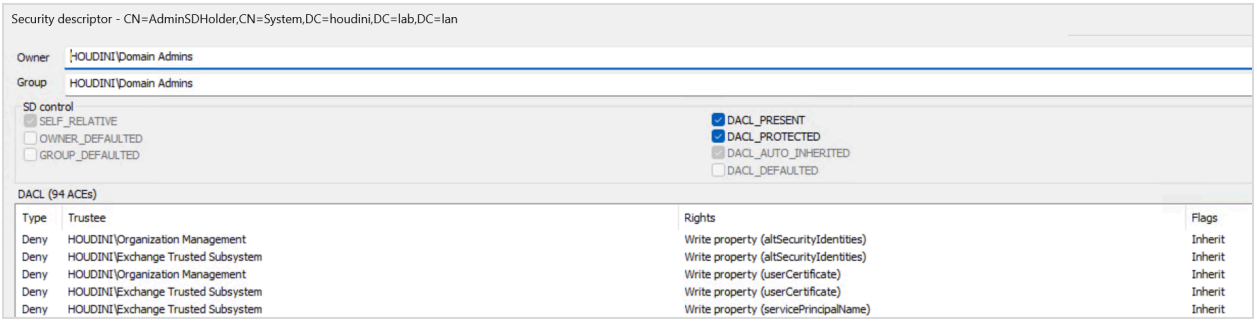


Figure 82 - Deny ACEs Added to AdminSDHolder by June 2019 Exchange Quarterly Release

Note: The Microsoft Exchange ADPrep isn't the first time Microsoft has had issues with AdminSDHolder and property sets. Microsoft created the default security descriptor for AdminSDHolder with multiple malformed and duplicative ACEs since they released the first version of AD with Windows Server 2000. The **Misconfiguration: Default ACEs with InheritedObjectType** section contains details on this misconfiguration.

The act of preparing Active Directory and domains for Microsoft Exchange introduces many additional Property Sets into the configuration partition for the forest, which interact with the numerous additional classes and attributes that the Exchange PrepareSchema portion of the setup adds to the AD forest. After the schema has been extended, the PrepareAD and PrepareDomain portions of the Exchange setup will create objects and containers within the root domain, configuration partition, and any additional domains you select. Along with that, the Exchange setup will also modify permissions on several key AD objects, including the domain NC and AdminSDHolder. To get a better understanding of the many ways that adding Exchange to AD can decrease the security of AD, please read Pwned by the Mail Carrier by Jonas Bülow Knudsen.

With Exchange, it's not just Property Sets that are an issue as it relates to AdminSDHolder. The /PrepareDomain operation for Exchange 2010 RC modified the DACL on the AdminSDHolder object, like most Exchange domain preparation operations, but this one granted the Exchange Windows Permissions group the ability to modify membership of groups, force reset passwords, and modify permissions. The Exchange Windows Permissions group is largely empty by default other than the Exchange Trusted Subsystem group nested inside, of which Exchange servers are members. Exchange servers are well known for their attack surface and exploitability and anyone with admin rights on an Exchange server can perform actions as Exchange Windows Permissions. Any member of Exchange Organization Management can also alter the membership of these other Exchange groups.

Microsoft is aware of many of the violations of least-privilege access models when it comes to Exchange and has been working to reduce permissions required to run Exchange. But more can be done with a thorough understanding of your own AD environment.

Recommendation: Review the permissions on AdminSDHolder in all of the domains in your forest(s).

Do your highly privileged administrator accounts have email addresses and use Microsoft Exchange regularly? For your sake, I sincerely hope not. The correct answer here is an emphatic NO; none of our privileged accounts are mail enabled. Great! If none of your privileged accounts are mail enabled, then Microsoft Exchange and its security groups don't need any special rights on privileged accounts, so all of the Exchange permissions can be carefully removed from the AdminSDHolder container.

Are your highly privileged on-prem administrator accounts synchronized to Microsoft Entra ID? Again, I hope not. Syncing privileged accounts from on-prem to Entra ID opens up avenues of attack from the cloud to on-prem and vice-versa. If you are not syncing on-prem privileged accounts to Entra ID, which you shouldn't be, then the Entra ID Connect AD DS Connector account doesn't need to be granted any permissions on AdminSDHolder either.

Misconception: AdminSDHolder Doesn't Protect Computer Objects

I've seen it said several times now that AdminSDHolder doesn't protect computer objects. I feel it is accurate to state that AdminSDHolder wasn't intended to protect computer objects. Looking back at the misconception on Protected Objects, we remember that the Domain Controllers and Read-Only Domain Controllers groups are protected objects; however, their members are not transitively protected like every other group that is protected. This is intentional in the code and based on the fExcludeMembers property in the SecureAdminTable structures which define the protected objects.

The notion that members of these particular two groups are not protected is, I believe, not very widely known, thus the misconception. I've wondered if the knowledge that AdminSDHolder doesn't protect the members of these two groups which generally contain computer objects provides the misconception that computer objects aren't protected?

> Be aware that AdminSDHolder also protects the members of the above groups, including nested members, but not computer object members. The AdminSDHolder protection only works because the AdminSDHolder object has ACL inheritance disabled. The protection will become ineffective if you enable ACL inheritance on the object.

Apologies to my manager Jonas for this one.

While it's true that, by default, AdminSDHolder does not protect DC computer objects, it doesn't mean that computer objects aren't protected. They just need to be added to one of the other multiple groups that transitively protect all their members.

Computer objects are a child class of the user object class. We could say that computers are user-derived, just as group-managed service accounts (gMSAs) are computer-derived. They have an adminCount attribute, just like users, and if you were unwise enough to add a computer object to Schema Admins or Domain Admins; the next ProtectAdminGroups operation the security descriptor of that object would be stamped with the protected DACL of AdminSDHolder.

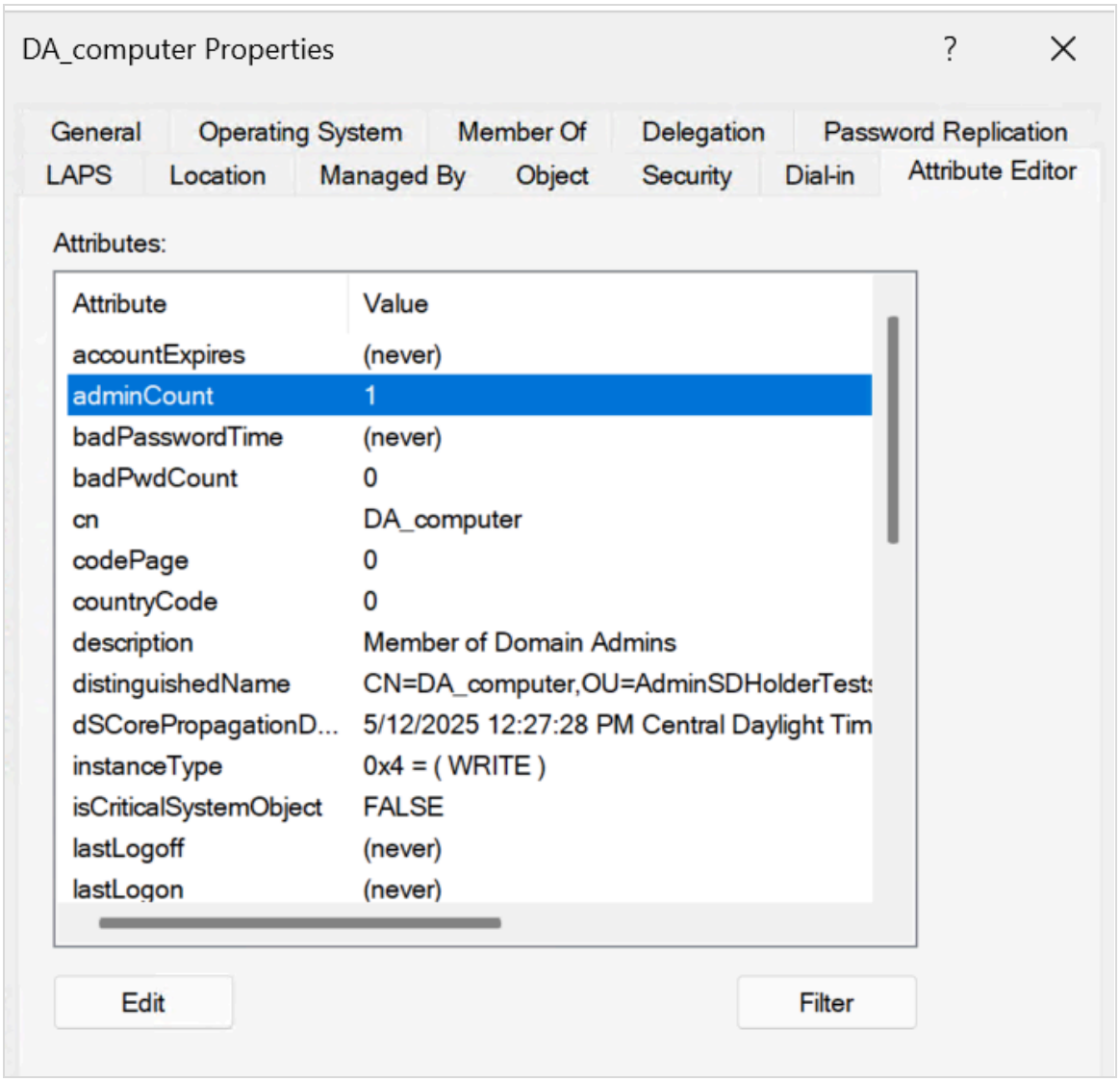


Figure 83 - Screenshot of DA_computer Object Properties in ad2025.lan Domain

If you review the results of the various Test-AdminSDHolderPostTest csv or xlsx files in my GitHub, you'll note that computer objects are protected across all Windows Server OSes, as long as they are added to a group where the fExcludeMembers property in the structure is not set to true.

In other words, a computer object added to Domain Admins will be protected, but a user (or any other) object added to the Domain Controllers group will not. But realistically, computer objects shouldn't be added to any of the builtin administrative groups.

Individual DC computer objects are not well protected from ACL Attack Paths. Neither are other Tier Zero computer objects such as AD CS PKI servers, Entra ID Connect sync or cloud sync computers, Exchange servers, privileged jump hosts, or many of the other computer objects that correspond to a

host that handles privileged information or credentials. And AdminSDHolder won't step in to automatically protect them like the default privileged users and groups.

Recommendation: Ensure Tier Zero computer objects are placed in a well-designed and secure OU structure. Disable DACL inheritance at the OU level where privileged computer objects are placed. This includes modifying the DACL of the default Domain Controllers OU to disable inheritance and ensure the DACL is restricted.

Disabling inheritance alone will not fully solve the problem, though, especially if the Account Operators group is populated or used at all. Every newly created user, group, computer, and OU includes an explicit ACE from the AD Schema which grants Account Operators Full Control of that object. DC computer objects, thankfully, have this explicit delegation removed, but any other highly privileged computer objects will not. Review the DACL of each highly privileged computer object and remove the explicit Account Operators ACE while also ensuring the Account Operators group remains empty.

Misconfiguration: Default ACEs With InheritedObjectType

While writing tests for some changes I made to SharpHoundCommon for this project, I stumbled upon some security descriptor output that didn't make sense to me. At first, I incorrectly thought something was going on with a .NET class, but it turns out that the default AdminSDHolder security descriptor has included multiple malformed ACEs for the Pre-Windows 2000 Compatible Access (Pre-Win2k) group since Active Directory was first released with Windows Server 2000 25+ years ago. With Windows Server 2003 RTM, the amount of malformed ACEs doubled with the addition of a set of ACEs that attempt to cover the inetOrgPerson object class with an InheritedObjectType. The inetOrgPerson class was added in Windows Server 2003 RFC 2798 standards compliance and X.500 compatibility. It is effectively a duplicate of the user class for standards purposes. These InheritedObjectType ACEs appear to be an attempt to limit Pre-Win2k read access to specific property sets on specific object types.

In the **AdminSDHolder Purpose & Intentions** section, I documented the three schema versions that have set the default security descriptor for AdminSDHolder. For convenience, I've included the default security descriptors in SDDL format again with the malformed ACEs in the DACL highlighted in bold red. Additional unnecessary system-audit ACEs are highlighted in bold purple. SDDL uses acronyms for common trustees and, more specifically, the 'RU' acronym stands for Pre-Windows 2000 Compatible Access and 'WD' represents the Everyone well-known SID.

Schema version 13:

```
O:DA
G:DA
D:PAI(A;;LCRPLORC;;;AU)(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)
(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)
```

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)

(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)

S:AI(AU;CIIDSAFA;CCDCSWWPDTCRSDWDWO;;;WD)

Schema versions 30-31:

O:DA

G:DA

D:PAI(A;;LCRPLORC;;;AU)

(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)

(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;PS)

(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;LCRPLORC;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8a58d456d2;;S-1-5-32-560)

(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5-32-561)

S:AI(AU;SA;WPWDWO;;;WD)

(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)

(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)

Schema versions 44-91:

O:DA

G:DA

D:PAI(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;4c164200-20c0-11d0-a768-00aa006e0529;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;bc0ac240-79a9-11d0-9020-00c04fc2d4cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;59ba2f42-79a2-11d0-9020-00c04fc2d3cf;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;RPWP;bf967a7f-0de6-11d0-a285-00aa003049e2;;CA)

(OA;;RP;46a9b11d-60ae-405a-b7e8-ff8a58d456d2;;S-1-5-32-560)

(OA;;RPWP;6db69a1c-9422-11d1-aebd-0000f80367c1;;S-1-5-32-561)

(OA;;RPWP;5805bc62-bdc9-4428-a5e2-856a0f4c185e;;S-1-5-32-561)

(OA;;LCRPLORC;;4828cc14-1437-45bc-9b07-ad6f015e5f28;RU)

(OA;;LCRPLORC;;bf967aba-0de6-11d0-a285-00aa003049e2;RU)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;WD)

(OA;;CR;ab721a53-1e2f-11d0-9819-00aa0040529b;;PS)

(OA;CI;RPWPCR;91e647de-d96f-4b70-9557-d63ff4f3ccd8;;PS)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;DA)

(A;;CCDCLCSWRPWPLOCRRCWDWO;;;EA)

(A;;CCDCLCSWRPWPLOCRSDRCWDWO;;;BA)

(A;;LCRPLORC;;;AU)

```
(A;;CCDCLCSWRPWPDTLOCRSDRCWDWO;;;SY)

S:AI(AU;SA;WPWDWO;;;WD)

(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)

(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
```

In SDDL format, each ACE is contained between parenthesis and follows the format of:

```
(AceType;ACEFlags;AccessMask;ObjectType;InheritedObjectType;Trustee)
```

Note: If you want to understand SDDL better, here are two great resources:

- UW Information Technology - Understanding SDDL Syntax: https://uwconnect.uw.edu/it?id=kb_article_view&sysparm_article=KB0034194
- Splunk - Hey SDDL SDDL: Breaking Down Windows Security One ACE at a Time: https://www.splunk.com/en_us/blog/security/windows-security-sddl-guide-access-control.html

We'll take the first Pre-Win2k ACE in SDDL and break it down into human terms:

```
(OA;;RP;037088f8-0ae1-11d2-b422-00a0c968f939;bf967aba-0de6-11d0-a285-00aa003049e2;RU)
```

This is an ObjectAllow ACEType with no ACEFlags, an AccessMask of Read Property, an ObjectType specifying the RAS-Information property set by GUID, an InheritedObjectType specifying the User object class by GUID, and the Pre-Win2k trustee represented by the acronym RU.

Unfortunately, this approach is flawed. The format is grammatically correct per the rules of SDDL; however, it is not valid to specify an InheritedObjectType without the appropriate ACEFlags allowing for propagation of the ACE to child objects. In such instances, the SRM ignores the InheritedObjectType. Only when an object-specific ACE is configured to propagate to child objects, and a child object of the object class specified in the InheritedObjectType exists, will that ACE be an effective ACE per Requirement 4 in the AD Security Descriptor Requirements. More information on this is spelled out in the ACE Inheritance Rules section of SecAuthZ and in the Security Descriptor chapter of James Forshaw's Windows Security Internals.

I did a fair bit of testing in my lab around the issue of these ACEs with an InheritedObjectType specified and documented it in my GitHub here. After doing some testing, I realized that there are specific configuration scenarios where the malformed Pre-Win2k ACEs on the default AdminSDHolder security descriptor could cause an information disclosure vulnerability. I gathered more information and opened a case with Microsoft Security Research Center (MSRC) on July 1, 2025 in the interest of responsible disclosure and was assigned MSRC Case 99182. I did not expect this to be categorized as a serviceable vulnerability. I'm not even confident that this is something that Microsoft will choose to resolve.

On July 2, 2025 I provided MSRC with an additional method to replicate the issue using James Forshaw's NTObjectManager PowerShell module, specifically the Get-AccessibleDsObject cmdlet.

On July 7th, 2025 I received a response from MSRC stating: "***After careful investigation, this case has been assessed as not a vulnerability and does not meet MSRC's bar for immediate servicing.***"

Subject: RE: MSRC Case 99182 CRM:0022098720

jsykora,

Thank you again for submitting this issue to Microsoft.
Currently, MSRC prioritizes vulnerabilities that are assessed as "Important" or "Critical' severities for immediate servicing.

After careful investigation, this case has been assessed as not a vulnerability and does not meet MSRC's bar for immediate servicing.

In any modern Active Directory, you are protected by the following.

- Pre-Win2k Group only has authenticated users as members (not Everyone)
- Anonymous connections do not include the "Everyone SID" Let Everyone permissions apply to anonymous users - Windows 10 | Microsoft Learn
- Guest account is disabled. Accounts Guest account status - security policy setting - Windows 10 | Microsoft Learn

The intended permissions granted are already pretty broad in terms of what object types it can read properties from (all users, computers, inetorgpersons).
So, the information disclosure is now they can also read properties on groups protected by adminSDHolder.

Also note, that Authenticated Users already have these permissions, so this only is additional information if the attacker:

- Enables the guest account (already considered bad)
- Changes the default policy so that "Everyone" permissions can apply to anonymous users (this may be orthogonal to guest account being enabled, and you ony have to do 1 or the other, both are equally bad)
- Installs a domain In compatibility mode with WIndows NT 4.0 AND ignores all practical security advice over the past 20+ years.

Even if all of the above were true, while the attacker is able to see groups protected by adminSDHolder, they cannot make any changes.
Because of this, we are unaware of what an attacker can do with this information that would be in any way concerning.

Figure 84 - MSRC Case Response

The MSRC response is accurate and not unexpected. Unfortunately, in my time specializing in AD security I've assessed multiple environments where the protections outlined in "any modern Active Directory, you are protected by the following" are not in place for unknown reasons; the domain is in compatibility mode, and the guest account is enabled.

After sending notice that the case investigation is complete and the result is 'not a vulnerability', MSRC followed up by requesting that I share this whitepaper with them prior to public disclosure. I appreciate the curiosity even though this was deemed not a vulnerability.

Moving along from my MSRC experience, I also noted in my lab that any attempts to modify the AdminSDHolder DACL with an API-based approach (such as using the Advanced Security properties in ADUC) resulted in the SD post-processing and merge rules squashing all but one of the Pre-Win2k ACEs, leaving only that modified, properly formed ACE in the DACL.

I had originally focused on the default AdminSDHolder DACL; however, after looking at the SACL, I observed more anomalies. The schema 30 changes introduced with Windows Server 2003 RTM added two additional system-audit ACEs to the SACL which are added to the existing system-audit ACE that audits successful WriteProperty, WriteDACL, and WriteOwner grants for Everyone:

```
(OU;CIIOIDSA;WP;f30e3bbe-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
(OU;CIIOIDSA;WP;f30e3bbf-9ff0-11d1-b603-0000f80367c1;bf967aa5-0de6-11d0-a285-00aa003049e2;WD)
```

Breaking this down into more understandable language what these object audit ACEs do is monitor for successful WriteProperty grants on the gPLink and gPOptions attribute ObjectTypes restricted to organizational unit (OU) InheritedObjectTypes for the Everyone trustee. In this case, ACE Flags are configured, which is slightly more correct than the Pre-Win2k ACEs on the DACL. The ACE Flags configured are ContainerInherit, InheritOnlyACE, InheritedACE, and AuditSuccess. These flags would allow for these audit ACEs to properly propagate to any child OU objects of an AdminSDHolder protected object. The sticky bit there is that it is not possible, with any default or reasonable configuration of the AD Schema, to create an OU as a child object of a user, group, computer, or service account. This is true even if CVE-2021-34470 is applicable in the environment as the msExchStorageGroup class is not a possible superior of the organizational unit object class.

The issue with the ACEs in the SACL does not create a vulnerability; however, between the system-audit ACEs in the SACL which have an InheritedObjectType and the object-allow ACEs in the DACL that also have an InheritedObjectType, it does lead me to a conclusion that the folks responsible for creating this configuration may not have understood Windows security internals particularly well.

What were they attempting to accomplish with these InheritedObjectType ACEs? My best guess would be that they were attempting to mimic both the ACEs that are explicitly set by default on user objects to grant ReadProperty access to the General Information, Public Information, Personal Information, and Web Information property sets and the default system-audit ACEs configured on the domain NC root which audit WriteProperty on gPLink and gPOptions attributes of OUs. Replicating the system-audit ACEs on objects which, by default, have their DACL protected would make sense if it were common or even uncommon to have OUs as child objects of highly privileged admin accounts and groups. Replicating the ReadProperty property set ACEs might make more sense if the trustee were Authenticated Users, as it is on the default security descriptor of user objects; however, instead, Authenticated Users is already granted GenericRead rights on the default AdminSDHolder SD and these ACEs were targeted at the Pre-Win2k trustee instead.

Recommendation: Check the AdminSDHolder security descriptor in all the domains of all your forests. Use LDP or PowerShell. If you somehow still have the default SD after all this time, make any small change to the DACL, even if it is deleting one of the Pre-Win2k InheritedObjectType ACEs that also has an ObjectType, and apply the settings. The Windows security descriptor merge function will squash the malformed ACEs for you.

Additionally, check the membership of the Pre-Windows 2000 Compatible Access builtin group. The Anonymous and Everyone well-known SIDs have not been default members of the group since one of the versions of Windows Server 2008. Having the Anonymous and Everyone principals as members is not a secure configuration. My personal recommendation is that the Pre-Win2k group should be empty, which includes removing the Authenticated Users group as well, unless you are using AD CS in which

certificate authorities (CAs) can be members. As with any change in your environment, validate and test prior to making these changes in production.

Misconceptions: Miscellaneous

As I've been writing and editing this paper I've found several other misconceptions about AdminSDHolder that I thought warranted a quick discussion.

Misconception: Each forest has its own AdminSDHolder container and 'SDPROP' process.

The correct answer is that each domain in every forest has its own AdminSDHolder container. The ProtectAdminGroups task runs separately on the PDCe of each domain in all forests.

Misconception: *The background task stamps the DACL and removes inheritance on protected objects.*

See also: "*The PDCe operations master compares the ACL on protected users with the ACL of AdminSDHolder.*"

The ProtectAdminGroups background task (and, specifically, the SampReadAdminSecurityDescriptor() and SampUpdateSecurityDescriptor() functions in secadmin.c) compare the full binary security descriptor of the AdminSDHolder container with the full binary security descriptor of the target object. If even one bit or byte is off, SampUpdateSecurityDescriptor() will replace the entire SD on the target object, which consequently is also the only time that the system sets the adminCount attribute to "1". Replacing the entire SD replaces all SD Control Flags, the Owner and Group, the entire DACL with all of its ACEs, and the entire SACL with any of its system-audit or mandatory control ACEs. It's byte for byte, not DACL for DACL.

Misconception: fixupInheritance was the legacy rootDSE modify operation for AdminSDHolder and runProtectAdminGroupsTask is the current one.

fixupInheritance has always been associated with SDProp, which is not and has not been a part of the AdminSDHolder protection process. Prior to Windows Server 2008 R2, when the runProtectAdminGroupsTask rootDSE modify operation was introduced, there was no known method to manually start the ProtectAdminGroups background task.

Misconception: *It's not possible to delegate user class permissions on AdminSDHolder because it is a container object class.*

Certainly, Microsoft didn't make it easy to delegate least privilege access on the AdminSDHolder object. Microsoft itself has had issues with this as described in **Misconfiguration: Default ACEs With InheritedObjectType**. It's also true that other designs could offer a better experience and level of security. Sure, you can't add user class specific ACEs to AdminSDHolder from the ADUC security or

advanced security tab, but that's less of a restriction of AdminSDHolder and more of a restriction of the ADUC GUI. It's been possible since Windows 2000 to modify the AdminSDHolder ACL to add or modify object ACEs with ObjectType properties that are not applicable to container class objects via *DSACLS.exe*, *LDP.exe*, or any standards-based LDAP client. With the addition of .NET and PowerShell, that functionality has been further extended. What you can't do is attempt to use an InheritedObjectType to malform an ACE to restrict an ACE to a specific target class.

Misconception: *You can use Windows performance counters to monitor the progress of the AdminSDHolder ProtectAdminGroups task.*

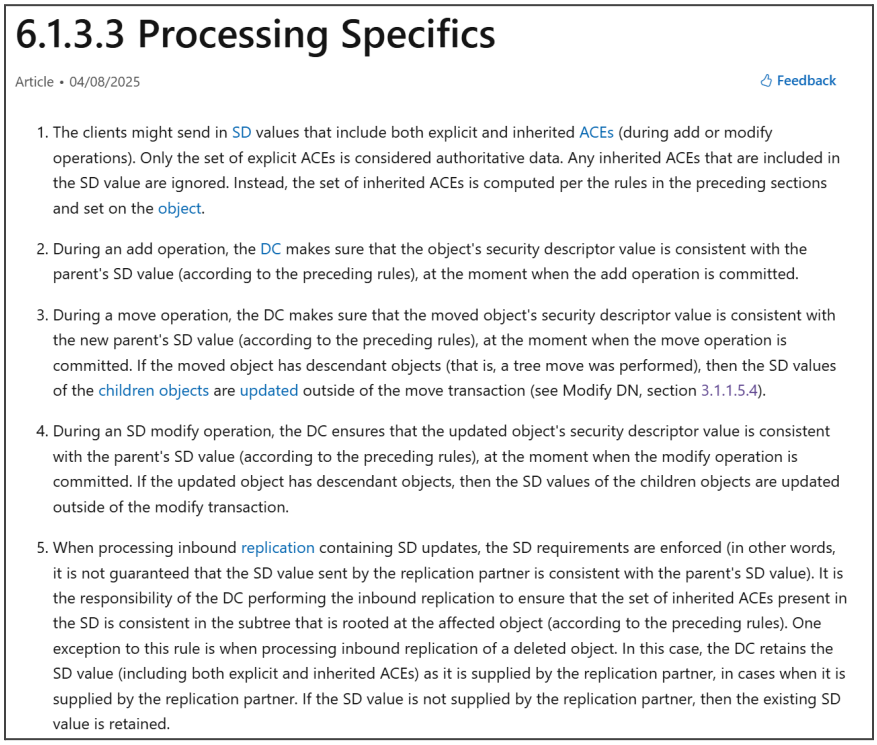
It is possible to monitor SDProp via Windows performance counters via the "\directoryservices(ntds)\ds security descriptor propagator runtime queue" counter path. There is not currently a performance counter specific to ProtectAdminGroups.

What SDProp Actually Does

The full technical details of what SDProp actually does is in [MS-ADTS] Section 3.1.1.6.3 Security Descriptor Propagator Update. SDProp handles propagation of inheritable ACEs from the parent down to the DACL of the descendent object. In other words, it fixes up inheritance. This makes lots of sense when you consider that the attribute to manually trigger SDProp is fixupInheritance.

And also, contrary to popular belief, the SDProp Background Task does run on every properly functioning DC in every domain, not just the PDC Emulator. Why does it run on every DC? Because of the way that AD replication occurs, the full inheritance on each individual AD Object isn't replicated to save bandwidth and time. That may seem silly in the times of 100GbE, but it's important to remember that when AD was originally designed, the network connections between sites were likely fractions of a T1 circuit and some DCs still replicate via telephone modems or satellite connections. Inherited ACEs are not considered authoritative data.

The Security Descriptor Requirements in MS-ADTS Section 6.1.3 also detail specifics around how DCs process security descriptors. Items 1 and 5 are particularly important to SDProp in that only the set of explicit ACEs is considered authoritative data and, when processing inbound replication security descriptor updates, the security descriptor requirements are enforced, making it the responsibility of the DC performing the inbound replication to ensure the set of inherited ACEs is consistent with the directory hierarchy as the DSA on that DC stores it.



6.1.3.3 Processing Specifics

Article • 04/08/2025 Feedback

1. The clients might send in SD values that include both explicit and inherited ACEs (during add or modify operations). Only the set of explicit ACEs is considered authoritative data. Any inherited ACEs that are included in the SD value are ignored. Instead, the set of inherited ACEs is computed per the rules in the preceding sections and set on the object.
2. During an add operation, the DC makes sure that the object's security descriptor value is consistent with the parent's SD value (according to the preceding rules), at the moment when the add operation is committed.
3. During a move operation, the DC makes sure that the moved object's security descriptor value is consistent with the new parent's SD value (according to the preceding rules), at the moment when the move operation is committed. If the moved object has descendant objects (that is, a tree move was performed), then the SD values of the children objects are updated outside of the move transaction (see Modify DN, section 3.1.1.5.4).
4. During an SD modify operation, the DC ensures that the updated object's security descriptor value is consistent with the parent's SD value (according to the preceding rules), at the moment when the modify operation is committed. If the updated object has descendant objects, then the SD values of the children objects are updated outside of the modify transaction.
5. When processing inbound replication containing SD updates, the SD requirements are enforced (in other words, it is not guaranteed that the SD value sent by the replication partner is consistent with the parent's SD value). It is the responsibility of the DC performing the inbound replication to ensure that the set of inherited ACEs present in the SD is consistent in the subtree that is rooted at the affected object (according to the preceding rules). One exception to this rule is when processing inbound replication of a deleted object. In this case, the DC retains the SD value (including both explicit and inherited ACEs) as it is supplied by the replication partner, in cases when it is supplied by the replication partner. If the SD value is not supplied by the replication partner, then the existing SD value is retained.

Figure 84 - 6.1.3.3 Processing Specifics

As I said previously, you absolutely can manually trigger SDProp to run immediately on a DC by using a LDAP operation via LDP or LDIF to set fixupInheritance to "1". That will force that specific DC to recalculate DACL inheritance on all objects it has, which is still nothing to do with AdminSDHolder. As Daniel Ulrichs notes in his blog:

> It's very common that people say that it's the SDProp that only run on the PDC Emulator DC and does all the comparison for protected objects. And they recommend to use the fixupInheritance operational attribute to trigger the AdminSDHolder propagation. *For example the Best Practices for Securing Active Directory white paper published April 2013.*
>
> Since the AdminSDHolder is a own background process that triggers the SDProp after the comparison has been made and the new SD has been enforced the fixupInheritance only triggers the SDProp to do its standard procedure, recalculate inherited permissions on all descendant objects. But protected objects doesn't have inheritance enabled and it doesn't work triggering the operational attribute.

Figure 85 - Screenshot from Daniel Ulrichs' Blog

But really, there's not much reason to manually trigger SDProp on a properly functioning DC. SDProp will trigger anytime either of these conditions is true:

- Any modification (originating or replicated) of the nTSecurityDescriptor attribute of any object, except for those modifications done by the SDProp task itself. Such an object is said to have caused a propagation event.
- Any modification of the DistinguishedName(DN) of an object that results in the object having a different parent, except for if the new parent is a Deleted Objects container. Such an object is said to have caused a propagation event.

If you change a security descriptor, the control flags could have changed enabling or disabling Inheritance. If you move an object from one OU to another, the inherited rights may change. SDProp is what handles propagating the correct inherited ACEs to the DACL. The inheritance on each individual DACL isn't replicated between DCs, so every DC in the domain runs the SDProp subprocess in LSASS to calculate those inherited ACEs. Additionally, each individual DC must guarantee that all inheritable ACEs are eventually propagated to all qualifying descendants of an object that causes a propagation event.

In the earliest versions of the Windows Server source code, there are comments in propdmon.c that explain how SDProp works and what it does:

```
72
73      // The security descriptor propagator is a single threaded daemon.  It is
74      // responsible for propagating changes in security descriptors due to ACL
75      // inheritance.  It is also responsible for propagating changes to ancestry due
76      // to moving and object in the DIT.  It makes use of four buffers, two for
77      // holding SDs, two for holding Ancestors values.  There are two buffers that
78      // hold the values that exist on the parent of the current object being
79      // fixed up, and two that hold scratch values pertaining to the current object.
80      // Since the SDP is single threaded, we make use of a set of global
81      // variables to track these four buffers.  This avoids a lot of passing of
82      // variables up and down the stack.
83      //
```

Figure 86 - Screenshot of Windows Source Code

Certainly, there are more things to learn about as it relates to SDProp. Such as:

- Understanding how the SDProp queue is tracked in the NTDS.dit database
- How SDProp ties into Performance Monitor and how its execution can be tracked there
- What the ValidateSDs DSA Heuristic does to change the behavior of SDProp
- Exactly what state an nTSecurityDescriptor is replicated in

But I did dig through the SDProp source enough to know that there is no check to ensure that SDProp is running on a PDCe FSMO role holder. I also dug through the AdminSDHolder source enough to know that SDProp is an entirely different process from the ProtectAdminGroups background task.

What AdminSDHolder Doesn't Do

There are some fairly important things that the AdminSDHolder background task doesn't do:

1. AdminSDHolder protects the Domain Controllers and Read-Only Domain Controllers security groups, but not their members. This appears to be expected behavior per: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/a0d0b4fa-2895-4c64-b182-ba64ad0f84b8 and the presence of the fExcludeMembers property. I spoke with a member of the AD product team, who confirmed this was expected behavior.

   This means that individual DC and RODC computer objects are not protected and, unfortunately, neither is the Domain Controllers OU where they reside. It's not uncommon to see DC computer objects that grant permissions they shouldn't. Fortunately, the explicit Account Operators ACE and the Creator Owner ACEs are removed from a DC computer object during the DCPromo operation on currently supported server OSes, but that wasn't always the case.

   While RODC server hosts are not Tier Zero, the computer object representing that RODC host is. For further explanation, see The Curious Case of the RODC.

   I opine that this configuration exists because, at the time these groups were made protected objects (Windows Server 2000 SP4), there were no known attacks that could abuse a computer object. However, that has changed with resource-based constrained delegation (RBCD) and other more modern attack primitives.

   The TierZeroTable community project started by some of my cohorts at SpecterOps classifies DCs as a Tier Zero asset, and each individual DC is a member of the Domain Controllers group.

2. AdminSDHolder doesn't protect these groups or their members: Access Control Assistance Operators, Certificate Service DCOM Access, Cryptographic Operators, Distributed COM Users, Event Log Readers, Hyper-V Administrators, IIS_IUSRS, Incoming Forest Trust Builders, Network Configuration Operators, OpenSSH Users, Performance Log User, Performance Monitor Users, Pre-Windows 2000 Compatible Access, RDS Endpoint Servers, RDS Management Servers, RDS Remote Access Servers, Remote Management Users, Storage Replica Administrators, Terminal Server License Servers, Windows Authorization Access Group, Allowed RODC Password Replication Group, Cert Publishers*, Cloneable Domain Controllers, Denied RODC Password Replication Group, DnsAdmins, DnsUpdateProxy, Enterprise Read-only Domain Controllers, External Trust Accounts, Group Policy Creator Owners, HelpServicesGroup, Protected Users, RAS and IAS Servers, WinRMRemoteWMIUsers_, TelnetClients, Exchange Trusted Subsystem, Exchange Windows Permissions, Organization Management

Not every single one of those groups is considered Tier Zero however, many of them are or can be in certain configurations and even the ones that aren't Tier Zero have interesting capabilities within the environment.

3. On Windows Server 2016, AdminSDHolder doesn't protect the Enterprise Key Admins group or its members. It's not until a Windows Server 2019 DC or a Windows Server SAC 1804 DC holds the PDC emulator FSMO Role that this group is protected. Oddly enough, this group and Key Admins are not currently, as of June 2025, mentioned in Section 3.1.1.6.1.2 Protected Objects of the MS-ADTS protocol specifications, which should always be correct (except for when they aren't).

   Enterprise Key Admins not being a protected object is especially interesting in light of the adprep bug that Microsoft introduced early in the lifecycle of Windows Server 2016.

   There's also the fact that Enterprise Key Admins can modify the msDS-KeyCredentialLink on any object in the AD forest that isn't otherwise protected. The msDS-KeyCredentialLink attribute was designed for Windows Hello for Business and is abusable for Shadow Credentials.

   I've collected a series of test data on how the OS level of the PDCe FSMO role holder affects AdminSDHolder on my GitHub.

   The TierZeroTable classifies Enterprise Key Admins as a Tier Zero asset.

4. Prior to Windows 2000 Server SP4, the only protected objects were Administrator, Domain Admins, Enterprise Admins, and Schema Admins.

   This means that AdminSDHolder did not protect Account Operators, Administrators, Backup Operators, Print Operators, Replicator, Server Operators, Cert Publishers, Domain Controllers, and krbtgt until SP4 for Server 2000. Windows Server 2003 was the first OS to natively protect these objects.

5. AdminSDHolder doesn't protect Cert Publishers. As stated above, it wasn't protected in Windows 2000 Server prior to SP4. Cert Publishers was also protected in the Windows Server 2003 RTM version and every newer OS version since then it went back to not being a protected object.

   The TierZeroTable classifies Cert Publishers as a Tier Zero asset.

6. AdminSDHolder doesn't protect Trusts (trustedDomain objects or their interdomain trust accounts), the MicrosoftDNS container, or any of the other objects or containers in the CN=System container. This means that any inheritable permissions applied at the root of the domain will propagate there.

7. AdminSDHolder doesn't protect foreign security principals (FSPs) whether they are representations of principals from another forest, the actual security principal in another forest, or representations of well-known security principals.

8. AdminSDHolder doesn't protect child objects of protected objects. Containers, OUs, and domainDNS are not the only objects that can contain child objects. The AD Schema defines which object class(es) can be the parent of an instance of any object class via the possSuperior and systemPossSuperior attributes. It is quite normal for a computer object to be a child of one of the aforementioned objects. And in a forest that's been extended with older versions of the Microsoft Exchange schema that are vulnerable to CVE-2021-34470, it's possible for a msExchStorageGroup object to be a child of a computer object. With the AD Schema being object-oriented and the msExchStorageGroup object class being a child class of the container class, it's possible for any object which can be a child of a container object to be a child of an msExchStorageGroup object.

   In the *houdini.lab.lan* forest in my lab, I created a computer object named 'AdminPC' and added it to the Print Operators group so AdminSDHolder would protect it. I then used a modified version of James Forshaw's poc_exchange_schema.ps1 script to create a msExchStorageGroup object named 'ASDHTest' with a user object named 'ASDHTestUser' as a child of that. No child objects of 'AdminPC' received protections from AdminSDHolder, as can be seen in the data on GitHub. Thank you to my friend Jake Hildreth for posing the question if any child objects of protected objects would also be protected.

9. AdminSDHolder doesn't protect msDS-DelegatedManagedServiceAccounts (dMSA) that were migrated from a 'service account' that is a protected object.

   Hopefully, you do not have any 'service accounts' which are members of highly privileged AD groups and thus eligible for AdminSDHolder protection as any host where a privileged 'service account' or managed service account is present is also a privileged host. If your 'service account' is Tier Zero, then so is any host that service runs on. If you do have 'service accounts' in Tier Zero, that is an attack path exposure that AdminSDHolder cannot help you with.

   Even if you don't have any 'service accounts' in privileged AD groups, Yuval Gordon's BadSuccessor research demonstrates that the ability for any low-privileged user to create a dMSA object in AD can most likely lead to compromise of any security principal in the domain, including members of highly privileged groups like Domain Admins. However, if an attacker is abusing BadSuccessor in your environment, protecting the attacker's dMSA account is perhaps the least of your concerns.

10. AdminSDHolder doesn't protect every Tier Zero object. It can't protect OUs where Tier Zero objects are located and it can't protect any GPOs that may be linked or applied to those OUs. It doesn't protect AD CS objects like the NTAuth store, CA computer objects, certificate templates,

or AIA objects. It doesn't protect arguably the most important AD object of them all: the Domain root object of objectClass domainDNS.

The TierZeroTable classifies the Domain root object, GPOs linked to a Tier Zero container, the Users container, AIA certificate authority (CA) objects, Certificate templates, Enterprise CA computer objects, Root CA computer objects, and the NTAuth store to all be Tier Zero assets.

11. Security principals and custom groups that are delegated effective Tier Zero permissions or rights within the forest are not protected by AdminSDHolder unless they are also a designated protected object. This includes principals like the MSOL_ directory sync accounts Entra ID Connect uses or the gMSA accounts used Entra ID Connect cloud sync uses to perform Password Hash Synchronization (PHS). In more recent versions of Entra ID Connect, the MSOL_ account will have a protected DACL with a severely restricted set of ACEs, but AdminSDHolder does not protect it. Another common example are PKI admins which may be local administrators or hold specific privileged rights in the PKI ecosystem, but are not members of default privileged groups.

    In nearly every AD environment I've had the opportunity to review, there is almost always at least one custom group that has been delegated permissions or assigned rights in the forest that either grant it effective AD administration capabilities or the ability to gain those capabilities. More often than not, these custom groups and their members are not protected objects.

What ProtectAdminGroups Actually Does

This flowchart and pseudocode outline are based upon my review of the Windows 2003 RC leaked source code and inferring how changes made to the AdminSDHolder process are likely to have been implemented since then.

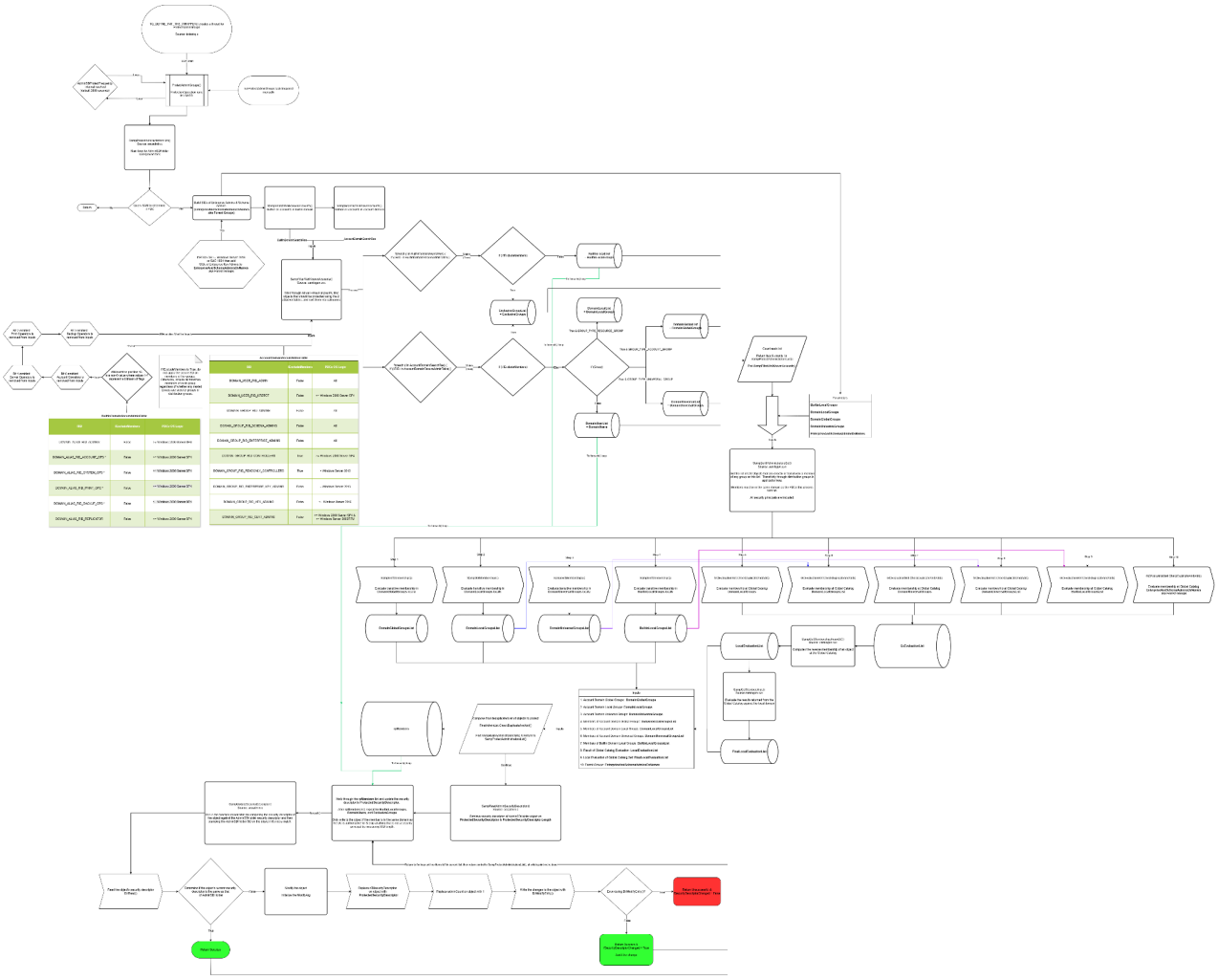


Figure 87 - Flowchart of AdminSDHolder Process

1. Only run on PDCe FSMO role holder

2. Define set of objects to be protected from BuiltinDomainSecureAdminTable and AccountDomainSecureAdminTable, filtering out accounts from BuiltinDomainSecureAdminTable if dSHeuristics is in play.

3. Built set of all accounts (security principals including users, groups, computers, gMSA, etc.) in both the Builtin domain (S-1-5-32-) BuiltinDomainSearchRes and the local account domain (S-1-5-21-x-y-z-) AccountDomainSearchRes.

4. Loop through each item in BuiltinDomainSearchRes:

    a. If item in BuiltinDomainSearchRes is in the BuiltinDomainSecureAdminTable AND fExcludeMembers = False, then include it in the BuiltinLocalList which becomes BuiltinLocalGroups

    b. If item in BuiltinDomainSearchRes is in the BuiltinDomainSecureAdminTable AND fExcludeMembers = True, then include it in the ExclusiveGroupList which becomes ExclusiveGroups

5. Loop through each item in AccountDomainSearchRes:

    a. If item in AccountDomainSearchRes is in the AccountDomainSecureAdminTable, we have a match and continue with indented logic:

        i. If fExcludeMembers = True, then include it in the ExclusiveGroupList which becomes ExclusiveGroups.

        ii. ElseIf the item is a security group AND

            1. If item GROUP_TYPE_RESOURCE_GROUP, then include it in the DomainLocalListGroup which becomes DomainLocalGroups

            2. If item GROUP_TYPE_ACCOUNT_GROUP, then include it in the DomainGlobalList which becomes DomainGlobalGroups

            3. If item GROUP_TYPE_UNIVERSAL_GROUP, then include it in the DomainUniversalList which becomes DomainUniversalGroups

        iii. Else the item is not a security group, it's considered a user and included in the DomainUserList which becomes DomainUsers

6. BuiltinLocalGroups, DomainLocalGroups, DomainGlobalGroups, DomainUniversalGroups, and the EnterpriseAndSchemaAdmins are input to the SampBuildAdministratorsSet() function, which will return the list of accounts which in the same domain that the process is running in and are directly or transitively members of any group on this list. Transitive membership includes membership through both distribution groups and security groups. Transitive members through a distribution group will not have the secure group's SID placed in its access token. Note that

DomainGlobalGroups and BuiltinLocalGroups are not evaluated against the Global Catalog here. SampBuildAdministratorsSet() performs the following steps:

a. Evaluate transitive membership in DomainGlobalGroups against the local domain using SampGetMemberships() function and include in DomainGlobalGroupsList

b. Evaluate transitive membership in DomainLocalGroups against the local domain using SampGetMemberships() function and include in DomainLocalGroupsList

c. Evaluate transitive membership in DomainUniversalGroups against the local domain using SampGetMemberships() function and include in DomainUniversalGroupsList

d. Evaluate transitive membership in BuiltinLocalGroups against the local domain using SampGetMemberships() function and include in BuiltinLocalGroupsList

e. Evaluate membership in DomainLocalGroups against Global Catalog using GCEvaluationSet.CheckDuplicateAndAdd() and include in GcEvaluationList

f. Evaluate membership in DomainLocalGroupsList against Global Catalog using GCEvaluationSet.CheckDuplicateAndAdd() and include in GcEvaluationList

g. Evaluate membership in DomainUniversalGroups against Global Catalog using GCEvaluationSet.CheckDuplicateAndAdd() and include in GcEvaluationList

h. Evaluate membership in DomainUniversalGroupsList against Global Catalog using GCEvaluationSet.CheckDuplicateAndAdd() and include in GcEvaluationList

i. Evaluate membership in BuiltinLocalGroupsList against Global Catalog and using GCEvaluationSet.CheckDuplicateAndAdd() include in GcEvaluationList

j. Evaluate membership in Enterprise Admins, Schema Admins, and if the PDCe is Server 2019 or higher Enterprise Key Admins against Global Catalog using GCEvaluationSet.CheckDuplicateAndAdd() and include in GcEvaluationList

k. GcEvaluationList is an input for SampGetMembershipsFromGC() which computes the reverse membership of an object at the Global Catalog and includes the output in LocalEvaluationList

l. LocalEvaluationList is the input to SampGetMemberships(), again, to evaluate all the results returned from the Global Catalog against the local domain to arrive at the FinalLocalEvaluationList.

m. The 10 arrays: DomainGlobalGroups, DomainLocalGroups, DomainUniversalGroups, DomainGlobalGroupsList, DomainLocalGroupsList, DomainUniversalGroupsList, BuiltinLocalGroupsList, LocalevaluationList, FinalLocalEvaluationList, and EnterpriseandSchemaAdmins are all inputs to FinalAdminList.CheckDuplicateAndAdd(), where the final list of accounts to protect is a deduplicated list named rpMembers.

n. Return control back to the main logic, ending the SampBuildAdministratorsSet() function.

7. The function SampReadAdminSecurityDescriptor() retrieves the security descriptor of the AdminSDHolder container as ProtectedSecurityDescriptor

8. Loop through rpMembers list, calling SampUpdateSecurityDescriptor() function for each item:

    a. Read the item's security descriptor

    b. If the item's security descriptor already matches the ProtectedSecurityDescriptor, the function returns success and continues with the next item.

    c. Else:

        i. Replace the nTSecurityDescriptor on the item with ProtectedSecurityDescriptor

        ii. Replace adminCount with 1

        iii. Commit the changes with DirModifyEntry()

        iv. If DirModifyEntry() is successful, return fSecurityDescriptorChanged = True

        v. Else, return fSecurityDescriptorChanged = False

    d. If Auditing is enabled and fSecurityDescriptorChanged = True then create an audit event.

9. Perform step 8 again for BuiltinLocalGroups, DomainUsers, and ExclusiveGroups, leaving off the auditing in step 8d

10. When all four loops are done, the process is done until it is triggered again.

Tier Zero and Attack Path Management

In the Attack Path Management Manifesto, Andy Robbins lays out a definition and vision for Attack Path Management (APM):

> **'Attack Path Management is the continuous discovery, mapping, and risk assessment of AD (on-prem and Azure) Attack Path choke points. Organizations can use APM to eliminate, mitigate, and manage Attack Paths, finally achieve effective Tiered Administration and Least Privilege, and significantly reduce the attack surface presented to the adversary by AD. APM does not require fundamental architectural changes and helps organizations achieve the above benefits without endlessly chasing down misconfigurations, vulnerabilities, and dangerous user behaviors.'**

Let's take that definition along with the three fundamental pillars of APM and view AdminSDHolder through the lens of Attack Path Management.

Pillar 1: Continuous, Comprehensive Attack Path Mapping

We can, and should, use available resources to comprehensively map attack paths. hen that functionality is built into the system we're addressing, we absolutely should use it. AdminSDHolder is an engineered bit of code that has been mapping and safely remediating the most basic, yet brutal, attack paths in AD for 25+ years.

The AdminSDHolder ProtectAdminGroups background task is continuous, running every hour by default, but it is not comprehensive.

Pillar 2: Empirical Impact Assessment of Attack Path Choke Points

Attack Path Choke Points are connections in the graph that connect the rest of the environment to a target asset or collection of assets. As Andy notes, "any connection into the collection of Tier Zero assets is a Tier Zero Choke Point – a privilege or user behavior the adversary must abuse in order to compromise a Tier Zero asset."

With the way that AdminSDHolder is designed and functions, it can serve as a significant Tier Zero Choke Point, at least for the set of Tier Zero objects that AdminSDHolder protects from common DACL abuse. Unfortunately, it is not comprehensive.

Pillar 3: Practical, Precise, and Safe Remediation Guidance

Let's see how AdminSDHolder stacks up against the four criteria that practical remediation guidance must adhere to:

- Remediation actions should not require drastic changes to the environment's directory services architecture
- Remediation actions should not require the organization to migrate from one directory services platform to another
- Remediation actions should not require expert-level knowledge to implement
- Remediations should have expected outcomes, and both the remediation and the expected outcomes should be verifiable

AdminSDHolder is built into the directory services. It's enabled by default. You don't need expert-level knowledge to implement it because it's fairly automatic. This paper maps out the expected outcomes and how to verify them.

Proposal: Attackers Abuse Functionality, and So Can Defenders

Attackers abuse functionality built into systems all the time, so why can't defenders abuse functionality as well? What if we abuse the functionality of the AdminSDHolder system and its mapping of transient members, even through distribution group membership, to our advantage?

1. Create a new Distribution Group in a secure OU with a Global group scope. For example:

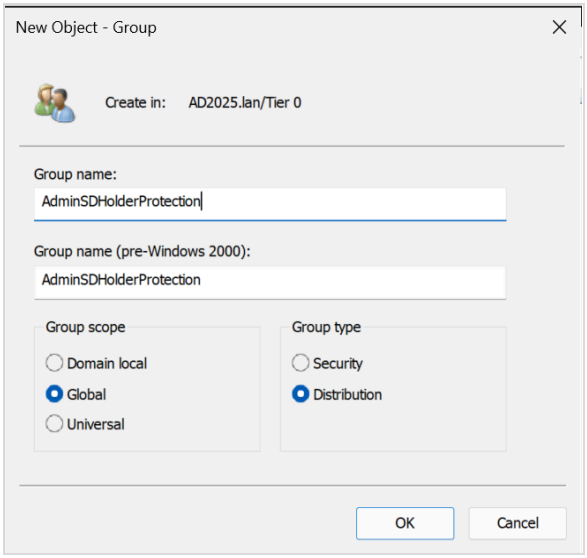


Figure 88 - Screenshot of Creating New AdminSDHolderProtection Global Distribution Group

2. Make sure to give the new group a Description and applicable information in the Notes:

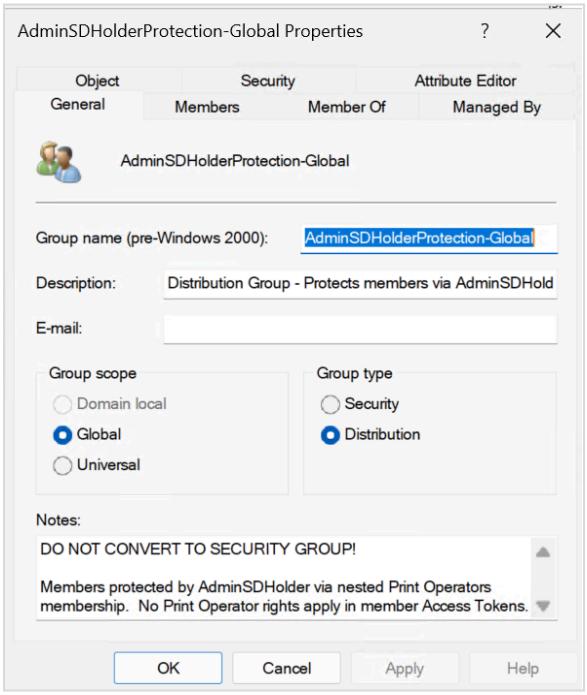


Figure 89 - AdminSDHolderProtection-Global Properties

3. Add the new AdminSDHolderProtection Global distribution group as a member of the Print Operators builtin local group for the same domain:

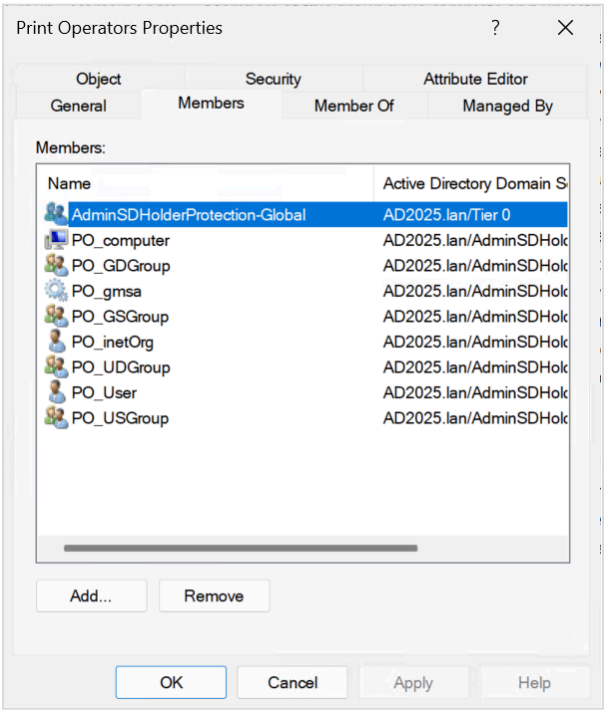


Figure 90 - Print Operator Members

4. Before any members are added to the AdminSDHolderProtection group, it's important to ensure that the Print Operators group for the local domain is as harmless as it can be. I strongly recommend "defanging" Print Operators even though members nested through distribution group membership will not have the Print Operators SID in the security context of their Access Token, which means that members of this group will receive no extra privileges, just protections. Still, Microsoft purposefully decided to include distribution groups in the protections for AdminSDHolder because they could be converted to security groups, so we'll do the same.

   By default, Print Operators are assigned rights to Allow log on locally (SeInteractiveLogonRight), Load and unload device drivers (SeLoadDriverPrivilege), and Shut down the system (SeShutdownPrivilege). These rights are assigned on Domain Controllers by default, even if no User Rights Assignments (URA) group policy setting is defined. To remove them, we need to configure User Rights Assignments via a group policy linked to the Domain Controllers OU:

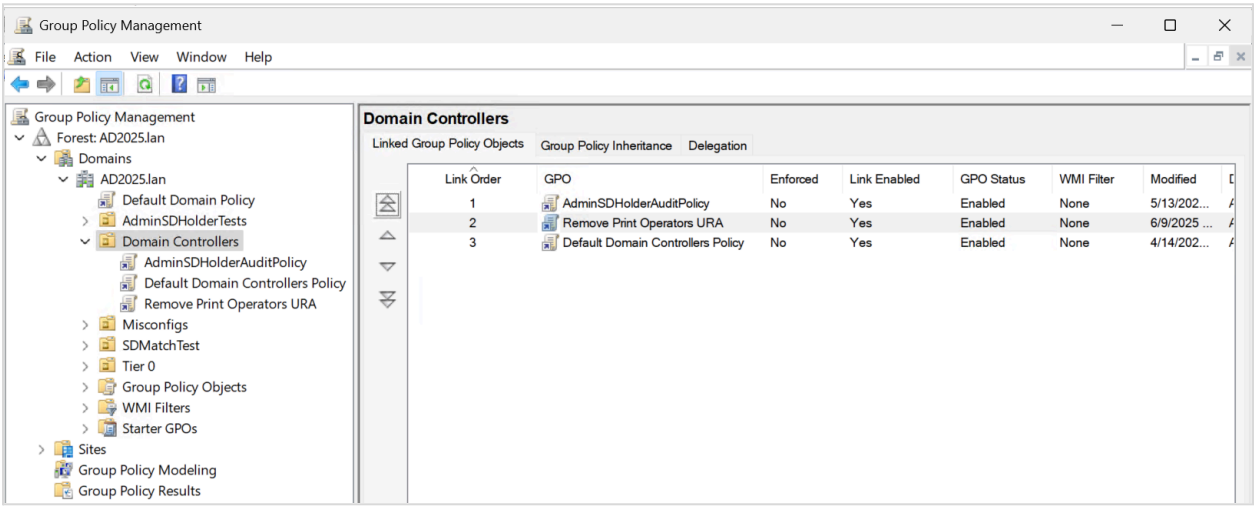


Figure 91 - Screenshot of Group Policy Management

5. With GPOs, it is challenging to provide an exact example for how they need to be configured as highly depends on the existing configuration of the environment. Ideally, URA will not be configured in the Default Domain Controllers Policy and will instead be configured in a URA-specific policy that has a higher link order than the Default Domain Controllers Policy so that its settings will be applied in the scope of management. Here I've created a policy named "Remove Print Operators URA"; however, a monolithic policy that configures URA at the Domain Controllers OU scope is preferable. A monolithic policy that covers multiple categories of settings is not preferable and difficult to manage.

6. Now that we have a DC policy for User Rights Assignments, we need to add the settings. These are located in Computer Configuration\Policies\Windows Settings\Security Settings\User Rights Assignments.

7. The first setting to configure is 'Allow log on locally', which determines which principals have the right to perform a local console login on a DC. Reviewing the documentation under the Explain tab, on DC the default URA is Account Operators, Administrators, Backup Operators, and Print Operators. This isn't quite accurate because Server Operators and Enterprise DCs are also assigned this right by default.

   My preference for the 'Allow log on locally' URA is to configure it so only Administrators and Enterprise DCs have this right. Account Operators should not be used, and if it is it shouldn't be able to log onto DCs. We are repurposing the Print Operators AdminSDHolder protections so it shouldn't have the right. Server Operators should not be used. The Domain Local Backup Operators is only for backing up AD and it also should not need the ability to log on locally. If there are other principals in your environment, such as Infrastructure Admins, which must be able to log on locally to DCs make sure they are added here. Use your best judgement, but it should look something like this:

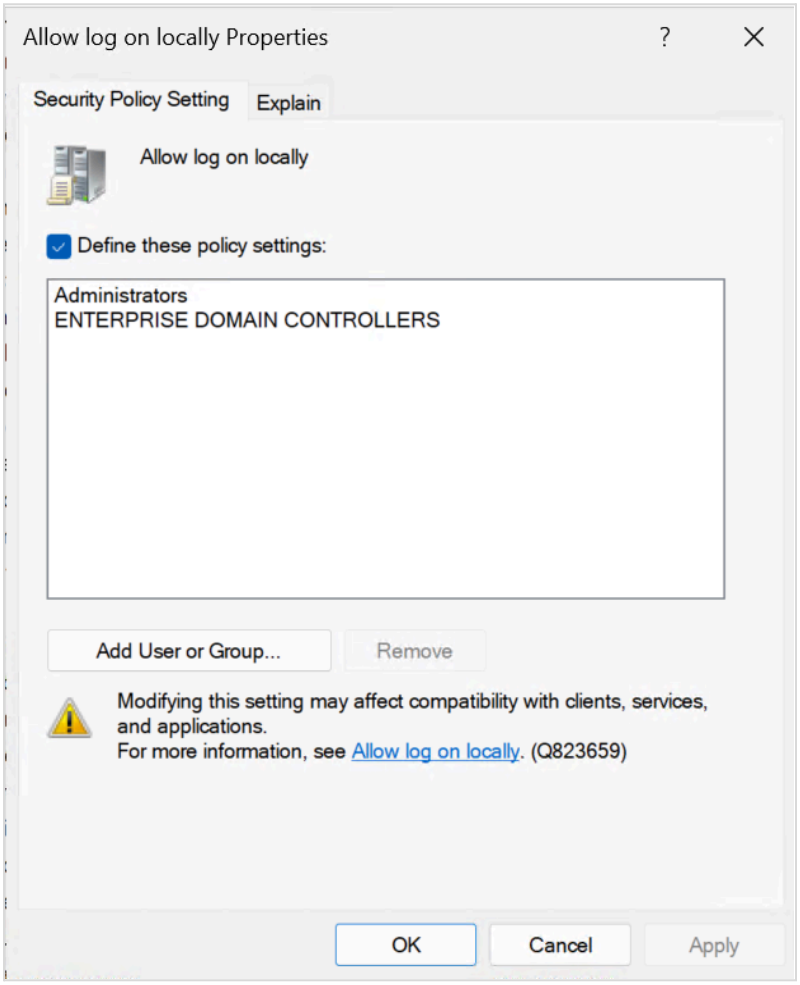


Figure 92 - Screenshot of Allow Log On Locally Properties

8. The next setting to configure is '[Load and unload device drivers]', which determines who can manage drivers on DCs. It's important to remember that drivers may have kernel access, so allowing anyone other than an administrator to load drivers on a DC is dangerous. Reviewing the documentation on the Explain tab we note that by default on DCs both Administrators and Print Operators are granted this right. Only Administrators should have it:

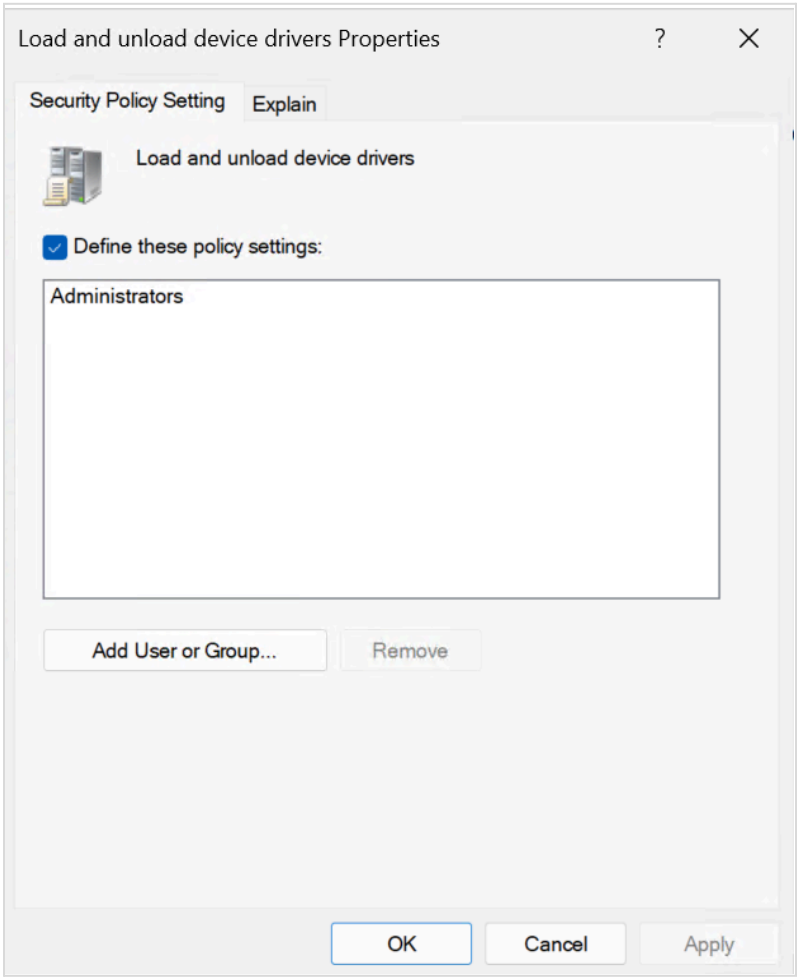


Figure 93 - Screenshot of Load and Unload Device Drivers Properties

9. The third URA setting we need to correct is the '[Shut down the system]' privilege. To use this privilege, an account needs to be logged on to the system. By default this right is granted to Administrators, Backup Operators, Server Operators, and Print Operators on DCs. This should privilege should only be granted to Administrators:

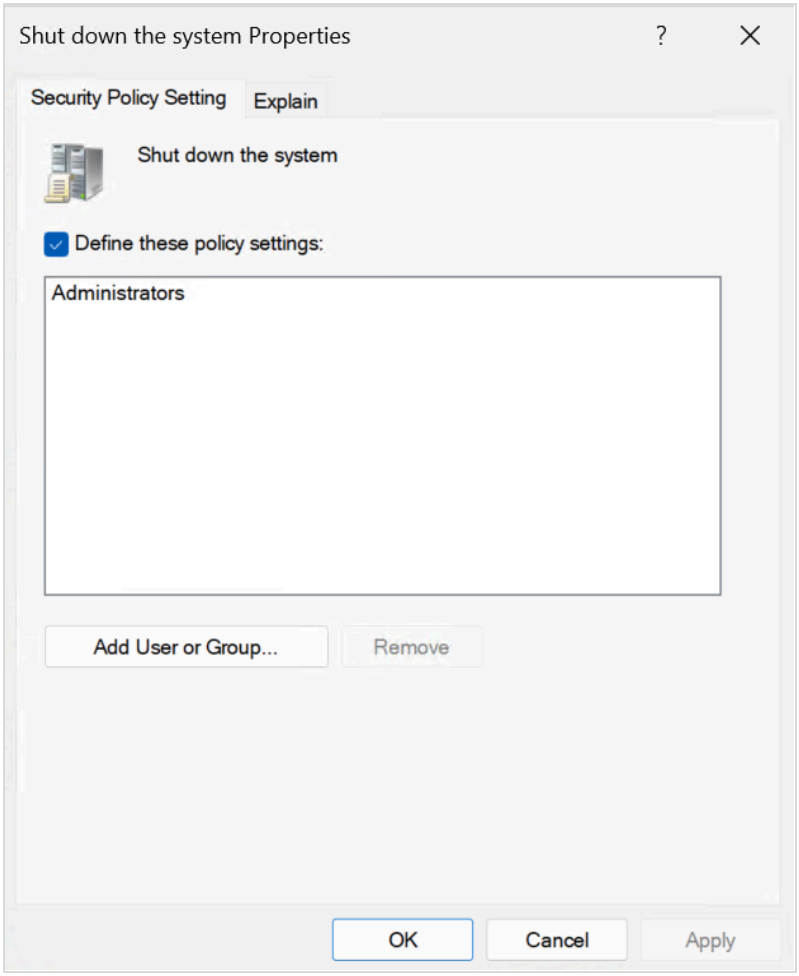


Figure 94 - Screenshot of Shut Down the System Properties

10. That covers the URA, but the Print Spooler service should not be running on DCs; not even for print queue pruning. Group Policy can be used to disable the Print Spooler service as well. Ideally this should go in a separate GPO which handles services on DCs. A Group Policy Preference (GPP) is the best way to handle this. The setting can be found at Computer Configuration\Preferences\Control Panel Settings\Services:

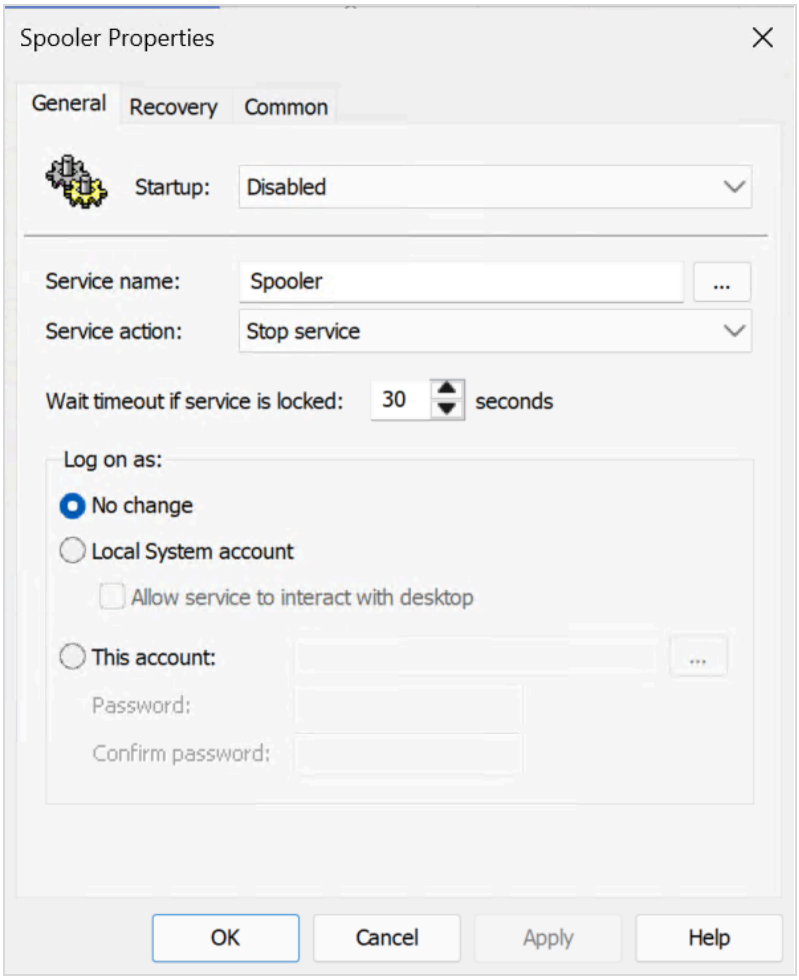


Figure 95 - Screenshot of Spooler Properties

11. There are other rights Windows implicitly grants to Print Operators. One of these is the ability to enumerate user sessions via the NetWkstaUserEnum API. If you would like to collect logon sessions without requiring membership in the local Administrators group, adding the SharpHound service account to Print Operators (combined with these 'defanging' steps) as a direct member instead of as a nested member of the AdminSDHolderProtection distribution group is a great least-privilege option. The NetWkstaUserEnum privilege is a known quantity. There may be unknown implicit rights the OS assigned to Print Operators. If this concerns your organization, do not follow through with this example.
12. Add a security principal to the AdminSDHolderProtection distribution group and either wait about an hour or manually run the ProtectAdminGroups task. This will test to ensure the protection works as expected, that the test account indeed is stamped with the secure AdminSDHolder security descriptor, and that it receives an adminCount of 1. It may also be prudent to log onto a system and check the privileges assigned to the account with whoami /all. See if your Security Operations Center (SOC) is paying attention.

As an example, I'll test this out with Kim Smith's PKI Admin account:

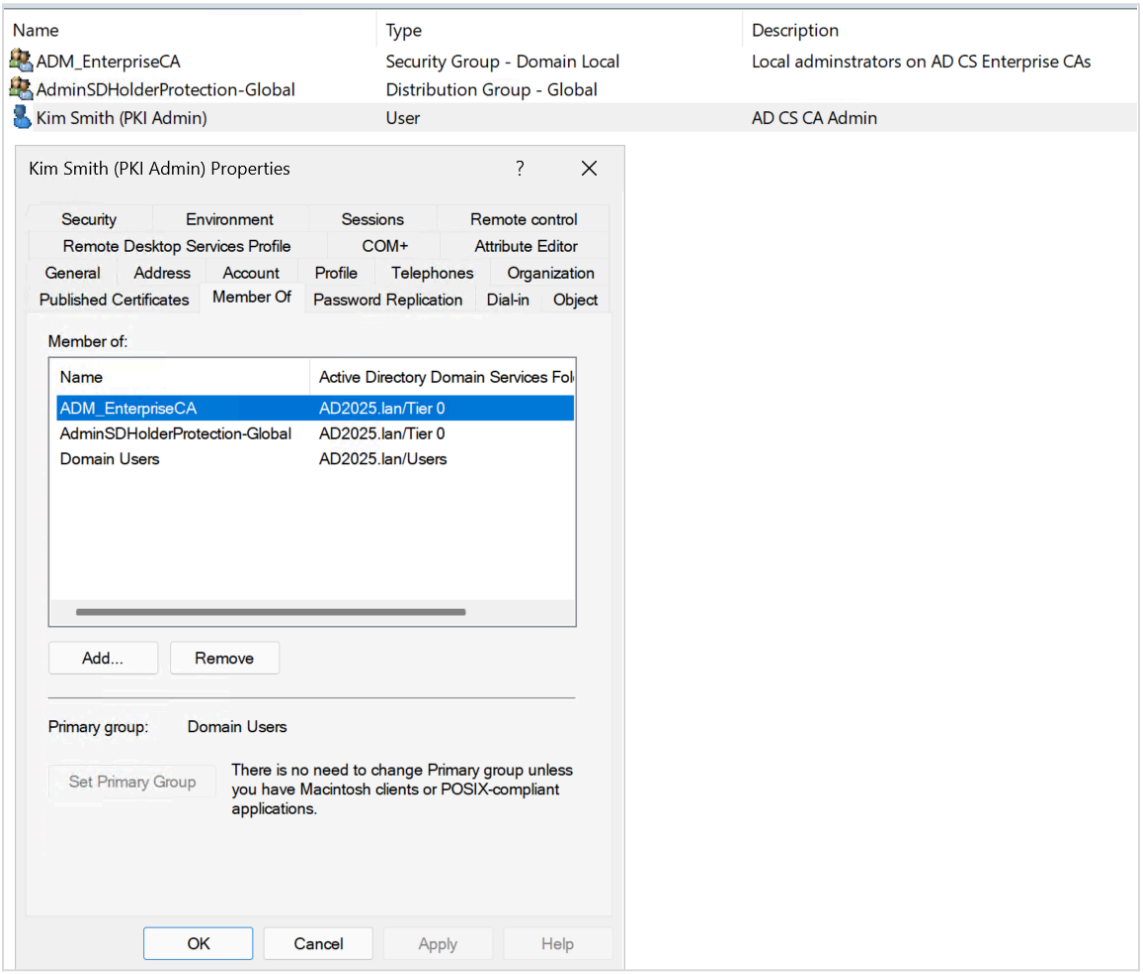


Figure 96 - ADUC Screenshot of Kim Smith (PKI Admin) Member Of Properties

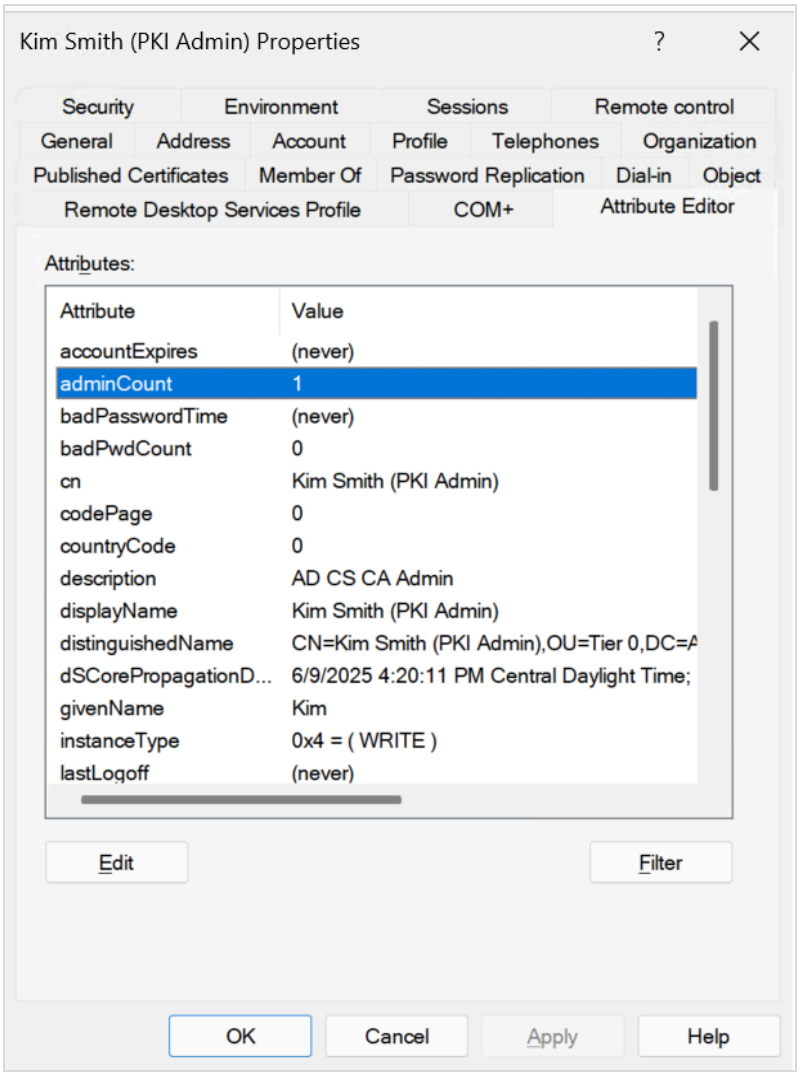


Figure 97 - Screenshot of Kim Smith (PKI Admin) Properties

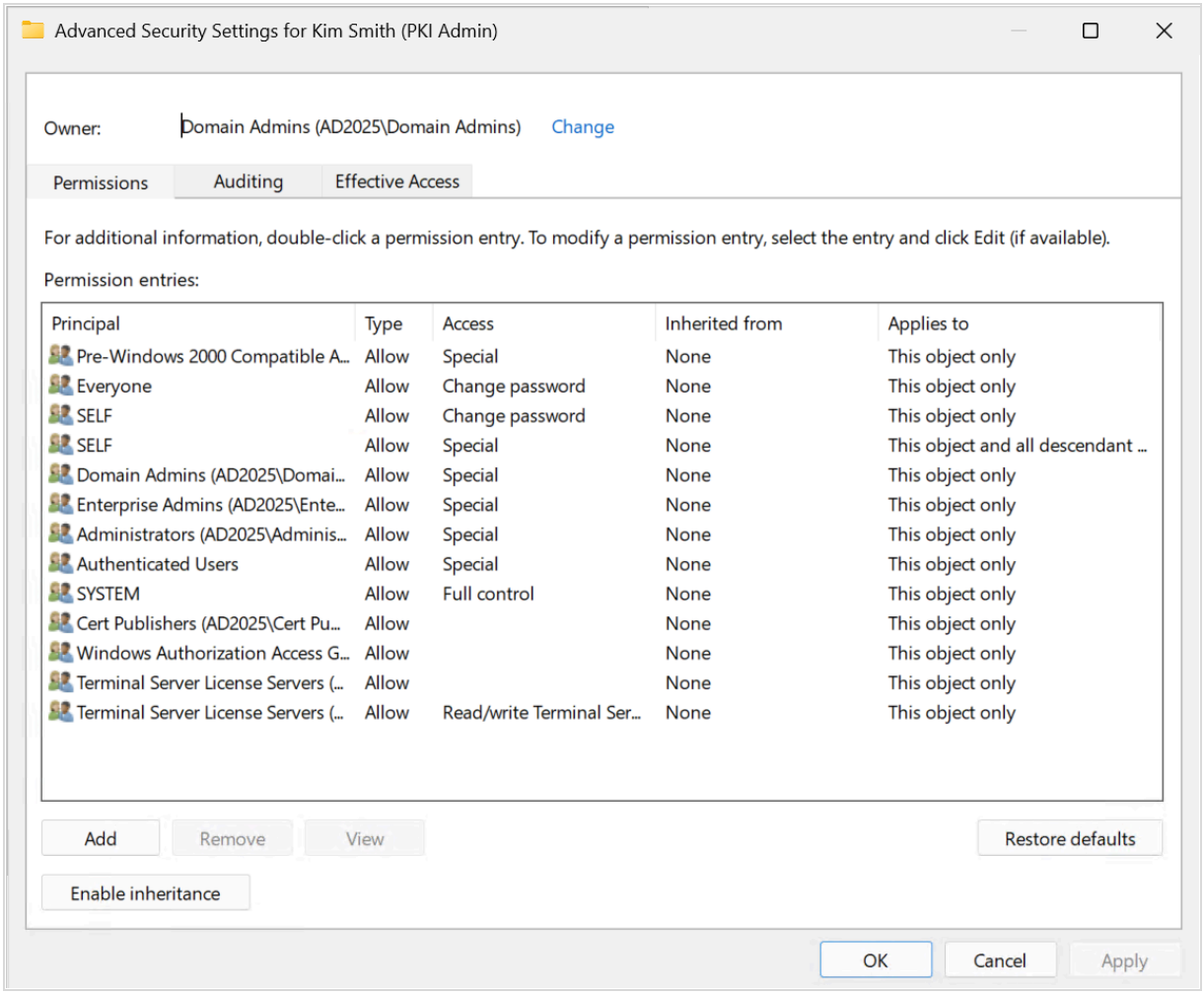


Figure 98 - AdminSDHolder Protects Kim's Account

13. Begin adding other privileged security principals to the AdminSDHolderProtection distribution group. It may be tempting to nest entire groups in the distribution group and if your goal is to protect not just the members, but also the group itself, that can be a viable path. Just remember that group scopes matter. You won't be able to add Universal groups or built-in Domain Local groups as members. You can, however, add individual users, computers, and service accounts from this domain. This could be handled via orchestration or scripting.

14. Eventually, you might end up with a fairly long list of members of your AdminSDHolderProtection distribution group.

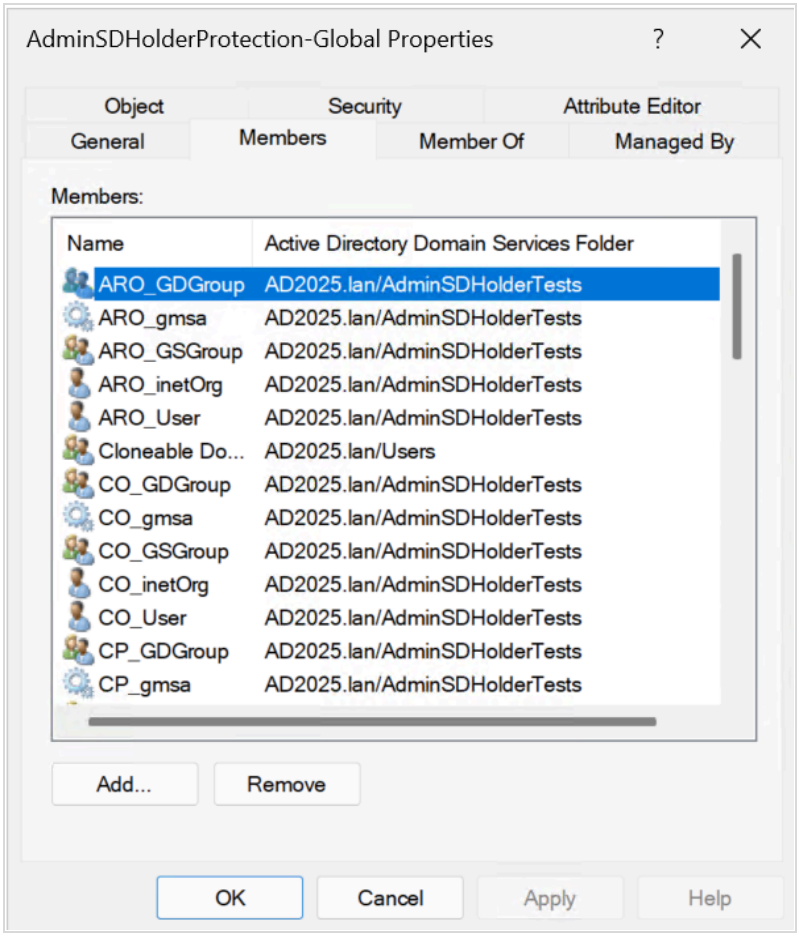


Figure 99 - Screenshot of AdminSDHolderProtection-Global Group Members

The other benefit of this is that, after ensuring the ProtectAdminGroups task ran as intended, I collected another dataset with Test-AdminSDHolder.ps1. Prior to creating and populating the AdminSDHolderProtection distribution group, AdminSDHolder protected 128 of 489 principals (i.e., 26%) in this domain. After implementation of this example, it protected 211 of 620 principals (i.e., 34%).

I'm using a distribution group with a Global scope because AdminSDHolder can only protect principals in its own domain. Global groups can only contain members from the same domain; however, this introduces a challenge to this concept. The AdminSDHolderProtection distribution group cannot be utilized to protect builtin local groups as 'Builtin groups cannot be added to other groups'. This means that AdminSDHolder cannot protect groups like Cryptographic Operators or Remote Management Users; however, if they had any members, you could add those principals directly to the AdminSDHolderProtection distribution group so AdminSDHolder will protect them.

A potential mitigation around the limitation of an AdminSDHolder-protected distribution group not being able to protect builtin local groups would be to disable security inheritance on the Builtin container and configure a restrictive DACL similar to that of the AdminSDHolder container. This would need to be

combined with ensuring that membership in the Account Operators group is always empty to mitigate the implicit Full Control ACE that Account Operators is granted on every group by default.

Another possible limitation of this abuse of the AdminSDHolder functionality is that Microsoft doesn't recommend it and may not support it. There is no guarantee that this functionality will continue working exactly as stated in this paper. Microsoft generally recommends that the Print Operators group remain empty. I agree, with perhaps the exception of this method or granting least-privilege access to the SharpHound collector for logon session enumeration.

Alternative: Be and Stay Aware

If you're not ready to jump on the proactive approach of abusing AdminSDHolder functionality for defense, then I urge you to use AdminSDHolder's default functionality for defense by ensuring that the AdminSDHolder security descriptor is in a secure state which does not allow for violations of the Clean Source Principal and that it stays in a secure state. Review the AdminSDHolder security descriptor routinely. Check the AdminSDHolder node in BloodHound to determine if the inbound paths create tiering violations. Make sure your audit policies are configured properly and that you are collecting the correct logs from your DCs.

Attack Paths Managed

Regardless of whether you choose to take a proactive or reactive approach, AdminSDHolder cannot identify or protect everything that is Tier Zero, provide protection from any abuse primitive which is not DACL related, nor provide protection to Builtin Local Group objects which are not already protected objects. AdminSDHolder can't protect anything which isn't a security principal.

Allow AdminSDHolder to do its job; don't hinder it with misconfigurations which make it less able to do its job. Utilize perhaps the only APM tool built into AD to the best of its limited capabilities. To manage all of the attack paths in your environment, utilize FOSS BloodHound Community Edition with some elbow grease or look into the only commercial Attack Path Management product: BloodHound Enterprise.

BloodHound and AdminSDHolder

AdminSDHolder has been a part of BloodHound since 4.1; however, the modeling of AdminSDHolder in past versions of BloodHound was certainly not as detailed as this whitepaper. We can see from this BloodHound Enterprise screenshot an example of how AdminSDHolder has been modeled:

Figure 100 - Screenshot of AdminSDHolder@ad2025.lan From BHE 7.4RC2

The AdminSDHolder is modeled as a container node, which is correct. However, object ACEs in the AdminSDHolder DACL are not intended to apply to a container object, but rather users, groups, and other security principals. AD handles this nuance with few issues because the AdminSDHolder security descriptor is directly applied to the various privileged security principals, at which point the object ACEs are handled natively. BloodHound hasn't been able to do this, so the AdminSDHolder node only contains Inbound Object Control at this point and there is no Outbound Object Control, with a brief one-hour default waiting period, to any of the protected objects.

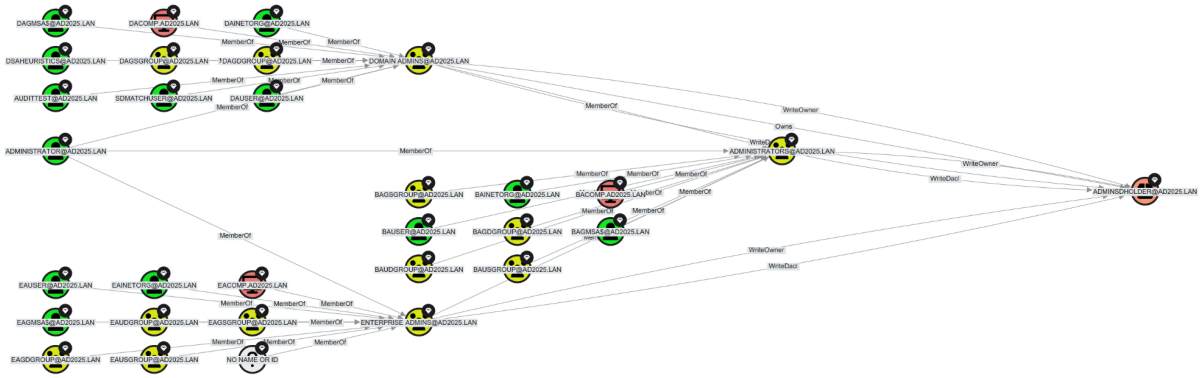


Figure 101 - AdminSDHolder Inbound Object Control Screenshot

The AdminSDHolder node is a default Tier Zero node in BloodHound, so BloodHound Enterprise findings can apply to it and it does have an entity panel which shows the “ACL Inheritance Denied” property. To be pedantic, this could be a little confusing because there are two separate security descriptor flags: DACL_Protected and SACL_Protected. By default, AdminSDHolder’s DACL_Protected flag is enabled and the SACL_Protected flag is not. This means that DACL inheritance is blocked on AdminSDHolder and SACL inheritance is allowed. DACL and SACL are both ACLs. Hopefully, we can eventually update this entity property to DACL Protected or DACL Inheritance Denied without causing issues.

When the Contains edge was originally introduced, it had the possibility of creating a false positive for ACL abuse edges across the Contains traversal due to objects in the container or OU potentially having DACL inheritance disabled. This is most common for objects AdminSDHolder protects, but can occur otherwise. BloodHound 4.2 introduced updates to how the ACL Inheritance Denied property is accounted for in regards to Contains edges. This may have resolved those false positives, but it still doesn’t give insight into which objects AdminSDHolder protects. When a privileged security principal, such as a member of Domain Admins, has an abusable ACL edge, it is currently not intuitive to determine the root cause and where the remediation must take place. Modifying the DACL on that privileged security principal is ultimately futile as the AdminSDHolder ProtectAdminGroups task will overwrite the change. Knowing which objects AdminSDHolder protects allows for understanding the root cause of that issue to be a weak DACL on the AdminSDHolder itself, and thus the AdminSDHolder object’s DACL being the correct location for remediation.

With the upcoming changes to AdminSDHolder in BloodHound, the first objective is that BloodHound needs to be able to determine when an edge is created due to the impact of the AdminSDHolder object and process so that remediation can be properly targeted at the root cause.

To achieve that objective, the code I added to BloodHound will analyze the security descriptor flags on the AdminSDHolder object along with the explicit ACEs in the DACL. Because DACL inheritance can unfortunately be enabled on AdminSDHolder and approximately a fifth of BHE environments have it enabled, BloodHound cannot consider any inherited ACEs. This is ideal because, when Windows adds or modifies a security descriptor, any inherited ACEs are not considered authoritative and are ignored. This process may look something like this:

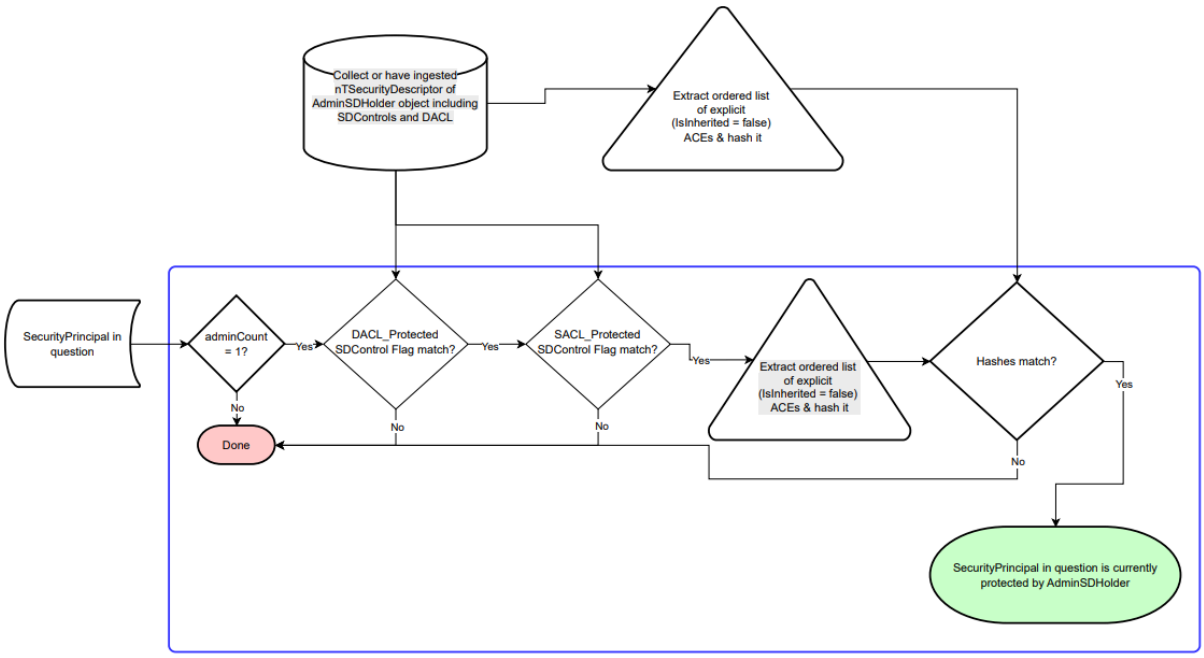


Figure 102 - Decision Flow for Determining AdminSDHolder Protection

The second objective for the AdminSDHolder changes is to introduce a non-traversable edge that flows from the AdminSDHolder object for a domain to the objects it protects. Changes made to that AdminSDHolder node for that domain via WriteDacl, Owns, or GenericAll edge will be replicated to all of the target nodes on the other end of the ProtectAdminGroups edge. This will show opportunities for persistence and will show in the attack graph where the permissions on the AdminSDHolderProtected objects come from.

The third objective is that BloodHound should be able to determine when a node is in a configuration state such that the AdminSDHolder process will continually protect it. This is beneficial information for both defensive and offensive operators.

This is a little more complex to accomplish due to the way that the ProtectAdminGroups task handles determining the transitive membership of protected objects. For this, group scope and group type come into play. SharpHound is already being modified to collect group scope and it can be as easily modified to capture group scope and type as one property. The next issue is that SharpHound currently ignores

the distinction between security and distribution groups. This will be resolved in a future update as well. It's important to make a distinction when calculating transitive membership for security groups vs calculating transitive membership for AdminSDHolder. Transitive membership in security groups causes SIDs of those groups to be added to a security principal's access token, and thus changes the rights and privileges associated with that logon session or thread. Transitive membership for AdminSDHolder must take distribution groups into account, which conveys to rights or privileges in an access token. Therefore, the transitive membership for AdminSDHolder protected objects will likely need to be handled separately from the standard transitive security group parsing. As some of the scaffolding for this objective is still in progress, I'll revisit this at a future date as it is not a core feature.

The fourth objective is to provide higher quality findings in BloodHound Enterprise such that the remediation language in the finding makes it clear whether the change needs to be made on the AdminSDHolder container or individual principals.

BloodHound v8.3.0 and the corresponding SharpHound v2.8.0 include new features to address these objectives. To help resolve **Misconception: AdminSDHolder Doesn't Protect Computer Objects**, the Admin Count property will now display on computer nodes. Additionally:

- SharpHoundCommon includes methods to hash authoritative security descriptors and compare hashed SDs.
- SharpHound FOSS and SharpHound Enterprise will now collect and hash the security descriptor of the AdminSDHolder container for a domain before continuing to collect the rest of the domain. When processing ACLs for user, computer, and group nodes the SD is hashed and compared with the AdminSDHolder hash for that domain. If there's a match, the node is `adminsdholderprotected` in the JSON.
- BloodHound CE and BloodHound Enterprise display the AdminSDHolder Protected property on user, group, and computer nodes based on the `adminsdholderprotected` property in the JSON. That includes managed service account nodes also. A new ProtectAdminGroups non-traversable edge is created between the AdminSDHolder node for each domain and the object nodes it protects. Documentation and help files are updated to match the information in this paper.
- A pre-built cypher search is added to find Tier Zero principals without AdminSDHolder protection.
- BloodHound Enterprise DACL-based findings are updated to include the AdminSDHolder Protected property.

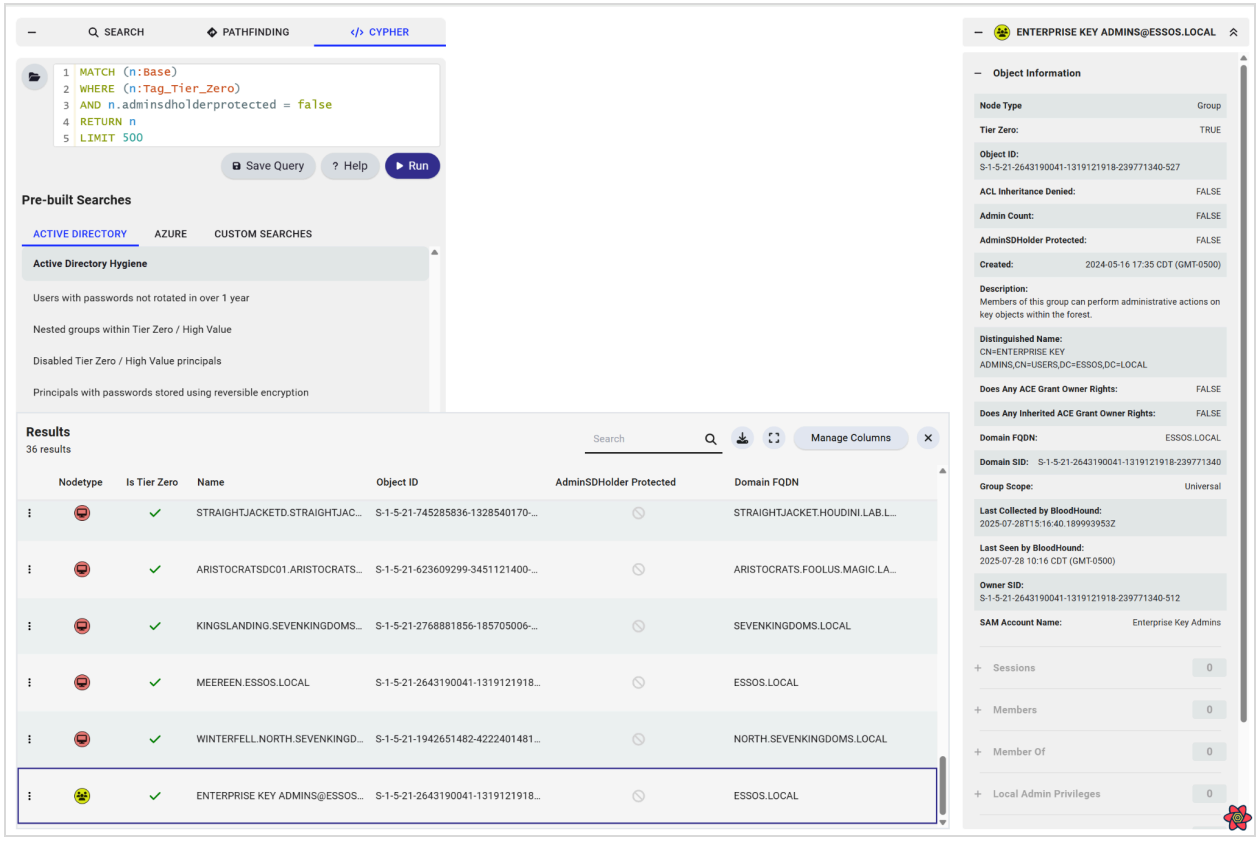


Figure 103 - New BloodHound Features

Proactive Remediations

- Review the security descriptor of the AdminSDHolder object regularly across all domains in all forests.
    - Ensure that no principals or groups that are not Tier Zero are not granted rights that would violate the Clean Source Principle resulting in the introduction of a control edge, resulting in an attack path.
    - Ensure that in a multi-domain forest, the security descriptor of each domain's AdminSDHolder object is equally restrictive.
- Ensure auditing is properly configured and review EID 4780 on PDCe.
- Consider creating a designated distribution group for the purposes of applying AdminSDHolder protections to otherwise unprotected security principals.

Recap

- AdminSDHolder is both an object and a process that Microsoft designed to protect highly privileged security principals from direct DACL manipulation by lower privileged principals.
- SDProp does not have any direct relationship to AdminSDHolder.
- The majority of the defaults around AdminSDHolder are reasonable.
- Once an object is privileged, it should always be treated as privileged.
- Direct association with a protected object is the trigger for applying AdminSDHolder protection, not adminCount.
- AdminCount is not a reliable way to determine privileged status; adminCount doesn't always count.
- The AdminSDHolder process can only protect objects in its own domain.
- AdminSDHolder protections are not permanent if an attacker has control over a principal that can modify AdminSDHolder.
- The members of the Domain Controllers and Read-Only Domain Controllers groups are purposefully excluded from being protected via the fExcludeMembers property.
- Modify dSHeuristics at your own peril.
- The default DACL_Protected flag on AdminSDHolder is there for good reason.
- Be wary of any changes to AdminSDHolder's DACL that decrease security for privileged accounts, even if Microsoft software tries to make that change.
- AdminSDHolder can generally protect computer objects and computer-derived objects like (g)MSAs, but not if they are members of a fExcludeMembers group.
- Inherited ACEs on security descriptors are not authoritative for add, modify, or replication purposes. SDProp's job is to run on every DC to handle the propagation of inheritable ACEs any time an object's security descriptor is modified or the object is moved.
- AdminSDHolder does not protect all Tier Zero security principals and even if it could, non-DACL-based attack paths would likely still exist. BloodHound can assist with finding attack paths to Tier Zero for instances where AdminSDHolder cannot or is not designed to protect.
- Not everything on the Internet is correct.

References

I'd like to credit Daniel Ulrichs (@DanielUlrichs) for once again inspiring me to dig deeper into how something in AD works. Your blog is full of great technical content that's more technically correct than even Microsoft documentation.

For the purposes of correlating each reference website with any misconceptions or misconfigurations from the paper, this table of abbreviations will be used:

Abbreviation	Section
MISC1	Misconception: SDProp
MISC2	Misconception: Changing the SDProp Interval
MISC3	Misconception: Running Manually
MISC4	Misconception: Just Change AdminSDHolder's DACL
MISC5	Misconception: adminCount
MISC6	Misconception: Permanence and Persistence
MISC7	Misconception: Protected ~~Groups~~ Objects
MISC8	Misconfiguration: dSHeuristics
MISC9	Misconfiguration: Change AdminSDHolder Interval
MISC10	Misconfiguration: Enabling Inheritance on AdminSDHolder
MISC11	Misconception: It's OK if Microsoft Changes the AdminSDHolder DACL, Right?
MISC12	Misconception: AdminSDHolder Doesn't Protect Computer Objects
MISC13	Misconfiguration: Default ACEs with InheritedObjectType
MISC14	Misconception: Miscellaneous

- Microsoft Appendix C: Protected Accounts and Groups in Active Directory: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/appendix-c--protected-accounts-and-groups-in-active-directory
    - As of June 2025: MISC1, MISC2, MISC3, MISC7, MISC9
    - See Issue 7240: https://github.com/MicrosoftDocs/windowsserverdocs/issues/7240 (for Microsoft)
    - Internet Archive: https://web.archive.org/web/20250602180745/https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/plan/security-best-practices/appendix-c--protected-accounts-and-groups-in-active-directory
    - PDF download of Best Practices for Securing Active Directory, which is the precursor to Appendix C: https://github.com/JimSycurity/AdminSDHolder/blob/main/Misc/Archive/Best%20Practices%20for%20Securing%20Active%20Directory.pdf
- Microsoft TechNet AdminSDHolder, Protected Groups, and Security Descriptor Propagator: https://social.technet.microsoft.com/wiki/contents/articles/22331.adminsdholder-protected-groups-and-security-descriptor-propagator.aspx
    - As of June 2025: MISC1, MISC2, MISC5, MISC7, MISC8, MISC9, MISC10
    - Internet Archive: https://web.archive.org/web/20250602181159/https://learn.microsoft.com/en-us/archive/technet-wiki/22331.adminsdholder-protected-groups-and-security-descriptor-propagator
- Microsoft TechNet Magazine – AdminSDHolder, Protected Groups, and SDPROP: https://learn.microsoft.com/en-us/previous-versions/technet-magazine/ee361593(v=msdn.10)?redirectedfrom=MSDN
    - As of June 2025: MISC1, MISC3, MISC5, MISC7, MISC8, MISC9,
    - Internet Archive: https://web.archive.org/web/20250602083833/https://learn.microsoft.com/en-us/previous-versions/technet-magazine/ee361593(v=msdn.10)?redirectedfrom=MSDN
- Microsoft Tech Community Five Common Questions About AdminSdHolder and SDProp: https://techcommunity.microsoft.com/t5/ask-the-directory-services-team/five-common-questions-about-adminsdholder-and-sdprop/ba-p/396293
    - As of June 2025, this is the most correct Microsoft document that isn't technical specifications. Thanks Ned! But we still have some MISC5 🙂
    - Internet Archive: https://web.archive.org/web/20250602181902/https://techcommunity.microsoft.com/blog/askds/five-common-questions-about-adminsdholder-and-sdprop/396293
    - 2008 Version: https://learn.microsoft.com/en-us/archive/blogs/askds/five-common-questions-about-adminsdholder-and-sdprop
        - Internet Archive: https://web.archive.org/web/20250617151637/https://learn.microsoft.com/en-us/archive/blogs/askds/five-common-questions-about-adminsdholder-and-sdprop

- Microsoft Active Directory Access Control List – Attacks and Defense: https://techcommunity.microsoft.com/t5/security-compliance-and-identity/active-directory-access-control-list-8211-attacks-and-defense/ba-p/250315
    - As of June 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250602201319/https://techcommunity.microsoft.com/blog/microsoft-security-blog/active-directory-access-control-list-8211-attacks-and-defense/250315
- Microsoft Support – Delegated Permissions are Not Available and Inheritance is Automatically Disabled: https://support.microsoft.com/en-us/topic/delegated-permissions-are-not-available-and-inheritance-is-automatically-disabled-56a70fa8-6d17-35ac-0f2c-87ec14b61980
    - As of June 2025: MISC1, MISC4, MISC5, MISC8, MISC9, MISC10
    - Internet Archive: https://web.archive.org/web/20250602202015/https://support.microsoft.com/en-us/topic/delegated-permissions-are-not-available-and-inheritance-is-automatically-disabled-56a70fa8-6d17-35ac-0f2c-87ec14b61980
- Microsoft Active Directory – FSMO Roles: https://learn.microsoft.com/en-us/troubleshoot/windows-server/identity/fsmo-roles
    - As of June 2025, this document doesn't reference the AdminSDHolder ProtectAdminGroups Background Task that runs on it every 60 minutes in the list of PDCe role functions.
    - Internet Archive: https://web.archive.org/web/20250602202436/https://learn.microsoft.com/en-us/troubleshoot/windows-server/active-directory/fsmo-roles
- Microsoft - Here to Stay: Gaining Persistence by Abusing Advanced Authentication Mechanisms: https://media.defcon.org/DEF%20CON%2025/DEF%20CON%2025%20presentations/DEF%20CON%2025%20-%20Marina-Simakov-and-Igal-Gofman-Here-to-stay-Gaining-persistence-by-abusing-auth-mechanisms.pdf
    - As of June 2025: MISC1, MISC4, MISC8
- Microsoft MS-ADTS OpenSpecs (Mostly Correct):
    - 3.1.1.3.3 rootDSE Modify Operations: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/fc74972f-b267-4c1a-8716-0f5b48cf52b9
        - Internet Archive: https://web.archive.org/web/20250602203107/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/fc74972f-b267-4c1a-8716-0f5b48cf52b9
    - 3.1.1.3.3.10 fixupInheritance: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/ddc8da4a-6ac8-4193-b51c-205cebbf483b
        - Internet Archive: https://web.archive.org/web/20250602203717/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/ddc8da4a-6ac8-4193-b51c-205cebbf483b
    - 3.1.1.3.3.26 runProtectAdminGroupsTask: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/f9dbd527-5594-4d27-be4b-ec16d23136e6

        - Internet Archive: https://web.archive.org/web/20250602203944/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/f9dbd527-5594-4d27-be4b-ec16d23136e6
    - 3.1.1.6 Background Tasks: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/29e5a169-9426-4374-870e-7f764e04db5e
        - Internet Archive: https://web.archive.org/web/20250602182010/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/29e5a169-9426-4374-870e-7f764e04db5e
    - 3.1.1.6.1 AdminSDHolder: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dd3d29f3-8e1e-4e8c-a210-9eaef3abd628
        - Missing Key Admins and Enterprise Key Admins
        - Internet Archive: https://web.archive.org/web/20250602204001/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dd3d29f3-8e1e-4e8c-a210-9eaef3abd628
    - 3.1.1.6.1.1 Authoritative Security Descriptor: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/233b5493-4246-4849-8eff-5fb578ce2642
        - Internet Archive: https://web.archive.org/web/20250602205300/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/233b5493-4246-4849-8eff-5fb578ce2642
    - 3.1.1.6.1.2 Protected Objects: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/a0d0b4fa-2895-4c64-b182-ba64ad0f84b8
        - As of June 2025: Missing Key Admins and Enterprise Key Admins
        - Internet Archive: https://web.archive.org/web/20250602222425/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/a0d0b4fa-2895-4c64-b182-ba64ad0f84b8
    - 3.1.6.1.3 Protection Operation: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/0a8a143b-d00b-425e-b43f-91df23779828
        - Internet Archive: https://web.archive.org/web/20250602223611/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/0a8a143b-d00b-425e-b43f-91df23779828
    - 3.1.16.1.4 Configurable State: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/50097362-ede5-40fa-973e-8d65e782e384
        - Internet Archive: https://web.archive.org/web/20250602223814/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/50097362-ede5-40fa-973e-8d65e782e384
    - 3.1.1.6.3 Security Descriptor Propagator Update: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/05c8a4b6-43aa-49f7-8c31-df3ac72230f3

        - This is what SDProp actually does.
        - Internet Archive: https://web.archive.org/web/20250602224243/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/05c8a4b6-43aa-49f7-8c31-df3ac72230f3
    - 5.1.3.3 Checking Access: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dc1b0dbc-cee1-4661-aa87-810b3c6759ab
        - Internet Archive: https://web.archive.org/web/20250603190751/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dc1b0dbc-cee1-4661-aa87-810b3c6759ab
    - 5.1.3.3.1 Null vs Empty DACLs: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/ee18efbf-e8d4-4eb0-8b27-79773e96d61f
        - Internet Archive: https://web.archive.org/web/20250603225413/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/ee18efbf-e8d4-4eb0-8b27-79773e96d61f
    - 5.1.3.3.2 Checking Simple Access: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/c55e113e-25a2-45eb-abba-bf5a9e521a8f
        - Internet Archive: https://web.archive.org/web/20250616182831/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/c55e113e-25a2-45eb-abba-bf5a9e521a8f
    - 5.1.3.3.3 Checking Object-Specific Access: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/3da5080d-de25-4ac8-9f2b-982709253dfb
        - Internet Archive: https://web.archive.org/web/20250616183100/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/3da5080d-de25-4ac8-9f2b-982709253dfb
    - 5.1.3.3.4 Checking Control Access Right-Based Access: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/d7e4cf8d-a771-486d-ba8e-6c83e4c6b1b2
        - Internet Archive: https://web.archive.org/web/20250616183232/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/d7e4cf8d-a771-486d-ba8e-6c83e4c6b1b2
    - 5.1.3.3.5 Checking Validated Write-Based Access: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/aa2c17ed-7a7e-45f8-83b6-0682b9d95e25
        - Internet Archive: https://web.archive.org/web/20250616184436/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/aa2c17ed-7a7e-45f8-83b6-0682b9d95e25
    - 5.1.3.3.6 Checking Object Visibility: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/4a7705f7-c61e-4020-86a7-41a44fb233e5

        - Internet Archive: https://web.archive.org/web/20250616184551/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/4a7705f7-c61e-4020-86a7-41a44fb233e5
    - 6.1.1.5.1 AdminSDHolder Object: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/9909ca0d-caf4-44b3-b089-b25aae13e601
        - Internet Archive: https://web.archive.org/web/20250603001539/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/9909ca0d-caf4-44b3-b089-b25aae13e601
    - 6.1.3 Security Descriptor Requirements: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/081c41f0-4c8d-4ab0-971d-77ec2504375a
        - Internet Archive: https://web.archive.org/web/20250604022623/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/081c41f0-4c8d-4ab0-971d-77ec2504375a
    - 6.1.3.1 ACE Ordering Rules: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dbfdc00c-1e4b-4165-939b-974e8ea23733
        - Internet Archive: https://web.archive.org/web/20250616185109/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dbfdc00c-1e4b-4165-939b-974e8ea23733
    - 6.1.3.3 Processing Specifics: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/e42f988c-72a0-4f8d-a705-7235eac175d9
        - Internet Archive: https://web.archive.org/web/20250616185425/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/e42f988c-72a0-4f8d-a705-7235eac175d9
- ADSecurity – Sneaky Active Directory Persistence #15: Leverage AdminSDHolder and SDProp to (Re)Gain Domain Admin Rights: https://adsecurity.org/?p=1906
    - As of June 2025: MISC1, MISC3
    - Internet Archive: https://web.archive.org/web/20250603004605/https://adsecurity.org/?p=1906
- Cayosoft - Active Directory Permissions Attack: Hackers Gain Persistence with AdminSDHolder: https://www.cayosoft.com/defending-active-directory-against-adminsdholder-attacks/
    - As of July 2025: MISC1, MISC5
    - Internet Archive: https://web.archive.org/web/20250703150009/https://www.cayosoft.com/defending-active-directory-against-adminsdholder-attacks/
- Cayosoft - AdminSDHolder: A Critical Active Directory Security Guide: https://www.cayosoft.com/adminsdholder/
    - As of July 2025: MISC1, MISC6, MISC7
    - Internet Archive: https://web.archive.org/web/20250703150936/https://www.cayosoft.com/adminsdholder/

- Daniel Ulrichs Secure Identity – AdminSDHolder – Pitfalls and Misunderstandings: https://secureidentity.se/adminsdholder-pitfalls-and-misunderstandings/
    - Correct
    - Internet Archive: https://web.archive.org/web/20250603125623/https://secureidentity.se/adminsdholder-pitfalls-and-misunderstandings/
- Daniel Ulrichs Secure Identity – Where the adminCount Doesn't Count and the SD Isn't What You Thought: https://secureidentity.se/adminsdholder-pt2/
    - Internet Archive: https://web.archive.org/web/20250603125623/https://secureidentity.se/adminsdholder-pitfalls-and-misunderstandings/
- Elastic - AdminSDHolder Backdoor: https://www.elastic.co/docs/reference/security/prebuilt-rules/rules/windows/persistence_ad_adminsdholder
    - As of June 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250616182633/https://www.elastic.co/docs/reference/security/prebuilt-rules/rules/windows/persistence_ad_adminsdholder
- eXploit - PowerView A New Hope: https://exploit.ph/powerview.html
    - As of June 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250616185944/https://exploit.ph/powerview.html
- GIAC - Protecting Administrative User Objects: How Microsoft Got a Good Idea Completely Wrong: https://www.giac.org/paper/gsec/3965/protecting-administrative-user-objects-how-microsoft-good-ideacompletely-wrong/106366
    - As of July 2025: MISC1, MISC4, MISC5, MISC10
    - Internet Archive: https://web.archive.org/web/20250701234912/https://www.giac.org/paper/gsec/3965/protecting-administrative-user-objects-how-microsoft-good-ideacompletely-wrong/106366
- harmj0y - Abusing Active Directory Permissions With PowerView: https://blog.harmj0y.net/redteaming/abusing-active-directory-permissions-with-powerview/
    - As of July 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250616190338/https://blog.harmj0y.net/redteaming/abusing-active-directory-permissions-with-powerview/
- Lepide - What is an AdminSDHolder Attack and How to Defend Against it?: https://www.lepide.com/blog/what-is-an-adminsdholder-attack-and-how-to-defend-against-it/
    - As of July 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250616190740/https://www.lepide.com/blog/what-is-an-adminsdholder-attack-and-how-to-defend-against-it/
- Netwrix - How Adversaries Achieve Persistence Using AdminSDHolder and SDProp: https://blog.netwrix.com/2023/06/16/adminsdholder/

    - As of July 2025: MISC1, MISC5, MISC7
    - Internet Archive: https://web.archive.org/web/20250604023549/https://blog.netwrix.com/2023/06/16/adminsdholder/
- Penetration Testing Lab - Domain Persistence AdminSDHolder: https://pentestlab.blog/2022/01/04/domain-persistence-adminsdholder/
    - As of July 2025: MISC1, MISC2, MISC5
    - Internet Archive: https://web.archive.org/web/20250616191146/https://pentestlab.blog/2022/01/04/domain-persistence-adminsdholder/
- Petri - Active Directory Security Understanding the AdminSDHolder Object: https://petri.com/active-directory-security-understanding-adminsdholder-object/
    - As of July 2025: MISC1, MISC2, MISC4, MISC5, MISC8
    - Internet Archive: https://web.archive.org/web/20250604023838/https://petri.com/active-directory-security-understanding-adminsdholder-object/
- PwnDefend - Abusing AdminSDHolder to Enable a Domain Backdoor: https://www.pwndefend.com/2021/09/11/abusing-adminsdholder-to-enable-a-domain-backdoor/
    - As of July 2025: MISC1, MISC2, MISC3
    - Internet Archive: https://web.archive.org/web/20250604024104/https://www.pwndefend.com/2021/09/11/abusing-adminsdholder-to-enable-a-domain-backdoor/
- Semperis - Improving your Active Directory Security Posture AdminSDHolder to the Rescue: https://www.semperis.com/resources/improving-your-active-directory-security-posture-adminsdholderto-the-rescue/
    - As of July 2025: MISC1, MISC4, MISC5
    - Internet Archive: https://web.archive.org/web/20250604135330/https://www.semperis.com/resources/improving-your-active-directory-security-posture-adminsdholderto-the-rescue/
- SentinelOne - Protecting Your Active Directory From AdminSDHolder Attacks: https://www.sentinelone.com/blog/protecting-your-active-directory-from-adminsdholder-attacks/
    - As of July 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250604135728/https://www.sentinelone.com/blog/protecting-your-active-directory-from-adminsdholder-attacks/
- Specops - Understanding Privileged Accounts and the AdminSDHolder: https://specopssoft.com/support/en/password-reset/understanding-privileged-accounts-and-adminsdholder.htm
    - As of July 2025: MISC1, MISC2, MISC5, MISC7, MISC8

    - Internet Archive: https://web.archive.org/web/20250616192217/https://specopssoft.com/support/en/password-reset/understanding-privileged-accounts-and-adminsdholder.htm

- Tenable - Securing Active Directory: How to Prevent the SDProp and adminSDHolder Attack: https://www.tenable.com/blog/securing-active-directory-how-to-prevent-the-sdprop-and-adminsdholder-attack
    - As of July 2025: MISC1, MISC5
    - Internet Archive: https://web.archive.org/web/20250603182215/https://www.tenable.com/blog/securing-active-directory-how-to-prevent-the-sdprop-and-adminsdholder-attack
- Tenable - Active Directory Security Deep-dive Master Class: https://static.carahsoft.com/concrete/files/9416/6154/4033/AD_Security_Training_Master.pdf
    - As of July 2025: MISC1
    - Internet Archive: https://web.archive.org/web/20250703151345/https://static.carahsoft.com/concrete/files/9416/6154/4033/AD_Security_Training_Master.pdf
- The Hacker Recipes - AdminSDHolder: https://www.thehacker.recipes/ad/persistence/adminsdholder
    - As of July 2025: MISC1, MISC5
    - Internet Archive: https://web.archive.org/web/20250609215409/https://www.thehacker.recipes/ad/persistence/adminsdholder
- Microsoft - Delegated Permissions are Not Available and Inheritance is Automatically Disabled: https://support.microsoft.com/en-us/topic/delegated-permissions-are-not-available-and-inheritance-is-automatically-disabled-56a70fa8-6d17-35ac-0f2c-87ec14b61980
    - As of July 2025: MISC1, MISC5, MISC8, MISC10
    - Internet Archive: https://web.archive.org/web/20250616194910/https://support.microsoft.com/en-us/topic/delegated-permissions-are-not-available-and-inheritance-is-automatically-disabled-56a70fa8-6d17-35ac-0f2c-87ec14b61980
- ITPro - Demystifying the AdminSDHolder Object: https://www.itprotoday.com/it-security/demystifying-the-adminsdholder-object
    - Correct
    - Internet Archive: https://web.archive.org/web/20250616200932/https://www.itprotoday.com/it-security/demystifying-the-adminsdholder-object
- TechTarget - Learn to adjust the AdminCount attribute in protected accounts: https://www.techtarget.com/searchwindowsserver/tutorial/Learn-to-adjust-the-AdminCount-attribute-in-protected-accounts
    - As of June 2025: MISC1, MISC4, MISC5
    - Internet Archive: https://web.archive.org/web/20250616202235/https://www.techtarget.com/searchwindowsserver/tutorial/Learn-to-adjust-the-AdminCount-attribute-in-protected-accounts

- Netwrix - Using Active Directory's AdminCount Attribute to Find Privileged Accounts: https://blog.netwrix.com/2022/09/30/admincount_attribute/
    - As of June 2025: MISC1, MISC2, MISC9
    - Internet Archive: https://web.archive.org/web/20250616190004/https://blog.netwrix.com/2022/09/30/admincount_attribute/
- Petri - Active Directory Security: Understanding the AdminSDHolder Object: https://petri.com/active-directory-security-understanding-adminsdholder-object/
    - As of June 2025: MISC1, MISC2, MISC8, MISC9
    - Internet Archive: https://web.archive.org/web/20250616203951/https://petri.com/active-directory-security-understanding-adminsdholder-object/
- SpecterOps - Pwned by the Mail Carrier: https://posts.specterops.io/pwned-by-the-mail-carrier-0750edfad43b
    - As of June 2025: MISC12
- Microsoft KB232199 - Description and Update of the Active Directory AdminSDHolder Object: https://www.betaarchive.com/wiki/index.php/Microsoft_KB_Archive/232199
    - As of July 2025: MISC4, MISC11
- Microsoft KB251343 - Manually initializing the SD propagator thread to evaluate inherited permissions for objects in Active Directory: https://www.betaarchive.com/wiki/index.php/Microsoft_KB_Archive/251343
    - As of July 2025: MISC1, MISC3
- Microsoft KB318180 - AdminSDHolder Thread Affects Transitive Members of Distribution Groups: https://www.betaarchive.com/wiki/index.php/Microsoft_KB_Archive/318180
    - As of July 2025: MISC5, MISC7
- Microsoft KB817433 - Delegated permissions are not available and inheritance is automatically disabled: https://www.betaarchive.com/wiki/index.php?title=Microsoft_KB_Archive/817433
    - As of July 2025: MISC4, MISC5, MISC7, MISC8, MISC9, MISC10
- Microsoft KB907434 - The "Send As" right is removed from a user object after you configure the "Send As" right in the Active Directory Users and Computers snap-in in Exchange Server: https://www.betaarchive.com/wiki/index.php/Microsoft_KB_Archive/907434
    - As of July 2025: MISC4
- Microsoft - Exchange 2010 and Resolution of the AdminSDHolder Elevation Issue: https://techcommunity.microsoft.com/blog/exchange/exchange-2010-and-resolution-of-the-adminsdholder-elevation-issue/598497
    - Internet Archive: https://web.archive.org/web/20250616204354/https://techcommunity.microsoft.com/blog/exchange/exchange-2010-and-resolution-of-the-adminsdholder-elevation-issue/598497
- Musings of the Proletariat - of Exchange 2010, Mobile Phones, and the AdminSDHolder or Why Doesn't My Phone Work Anymore?: https://musingsoftheproletariat.com/exchange-2010-phones-and-adminsdholder/

- As of June 2025: MISC4, MISC5
- Internet Archive: https://web.archive.org/web/20250603182831/https://musingsoftheproletariat.com/exchange-2010-phones-and-adminsdholder/

- C7Solutions - Administrators, AADConnect and AdminSDHolder Issues: https://c7solutions.com/2017/03/administrators-aadconnect-and-adminsdholder-issues
  - As of June 2025: MISC4, MISC5
  - Internet Archive: https://web.archive.org/web/20250701234600/https://c7solutions.com/2017/03/administrators-aadconnect-and-adminsdholder-issues
- Security Descriptors:
  - Microsoft Authorization – Security Descriptors: https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptors
    - Internet Archive: https://web.archive.org/web/20250603141010/https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptors
  - Microsoft Authorization – Securable Objects: https://learn.microsoft.com/en-us/windows/win32/secauthz/securable-objects
    - Internet Archive: https://web.archive.org/web/20250603141034/https://learn.microsoft.com/en-us/windows/win32/secauthz/securable-objects
  - Microsoft Authorization - DACLs and ACEs: https://learn.microsoft.com/en-us/windows/win32/secauthz/dacls-and-aces
    - Internet Archive: https://web.archive.org/web/20250603150256/https://learn.microsoft.com/en-us/windows/win32/secauthz/dacls-and-aces
  - Microsoft Authorization - Security Descriptor Controls: https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-control
    - Internet Archive: https://web.archive.org/web/20250603150530/https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-control
  - Microsoft Authorization – Access Masks: https://learn.microsoft.com/en-us/windows/win32/secauthz/access-mask
    - Internet Archive: https://web.archive.org/web/20250603151714/https://learn.microsoft.com/en-us/windows/win32/secauthz/access-mask
  - Microsoft Authorization – Security Information: https://learn.microsoft.com/en-us/windows/win32/secauthz/security-information
    - Internet Archive: https://web.archive.org/web/20250603153200/https://learn.microsoft.com/en-us/windows/win32/secauthz/security-information

- Microsoft Authorization - Security Descriptor Definition Language: https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-definition-language
  - Internet Archive: https://web.archive.org/web/20250603153400/https://learn.microsoft.com/en-us/windows/win32/secauthz/security-descriptor-definition-language
- Microsoft Authorization - Order of ACEs in a DACL: https://learn.microsoft.com/en-us/windows/win32/secauthz/order-of-aces-in-a-dacl
  - Internet Archive: https://web.archive.org/web/20250603150347/https://learn.microsoft.com/en-us/windows/win32/secauthz/order-of-aces-in-a-dacl
- Microsoft Authorization - Access Tokens: https://learn.microsoft.com/en-us/windows/win32/secauthz/access-tokens
  - Internet Archive: https://web.archive.org/web/20250603183126/https://learn.microsoft.com/en-us/windows/win32/secauthz/access-tokens
- Microsoft Authorization - SID Attributes in an Access Token: https://web.archive.org/web/20250603183331/https://learn.microsoft.com/en-us/windows/win32/secauthz/sid-attributes-in-an-access-token
  - Internet Archive: https://web.archive.org/web/20250603183331/https://learn.microsoft.com/en-us/windows/win32/secauthz/sid-attributes-in-an-access-token
- Microsoft AD-DS – How Security Principals Work: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-principals
  - Internet Archive: https://web.archive.org/web/20250603153857/https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-principals
- Microsoft AD-DS - Understand Security Identifiers: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers
  - Internet Archive: https://web.archive.org/web/20250603154930/https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/understand-security-identifiers
- Microsoft AD-DS – How Security Descriptors are Set on New Directory Objects: https://learn.microsoft.com/en-us/windows/win32/ad/how-security-descriptors-are-set-on-new-directory-objects
  - Internet Archive: https://web.archive.org/web/20250603155101/https://learn.microsoft.com/en-us/windows/win32/ad/how-security-descriptors-are-set-on-new-directory-objects
- Microsoft MS-ADTS OpenSpecs – 5.1.3.2 Access Rights: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584

        - Internet Archive: https://web.archive.org/web/20250603155400/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/990fb975-ab31-4bc1-8b75-5da132cd4584
    - Microsoft MS-ADTS OpenSpecs – 6.1.3 Security Descriptor Requirements: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/081c41f0-4c8d-4ab0-971d-77ec2504375a
        - Internet Archive: https://web.archive.org/web/20250602203858/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/081c41f0-4c8d-4ab0-971d-77ec2504375a
    - Microsoft MS-ADTS OpenSpecs – 6.1.6.7.5 nTSecurityDescriptor: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/0bf38ee3-016f-442e-9ceb-7b82348639cf
        - Internet Archive: https://web.archive.org/web/20250603155755/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/0bf38ee3-016f-442e-9ceb-7b82348639cf
    - Microsoft MS-DTYP OpenSpecs – 2.4.6 SECURITY_DESCRIPTOR: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/7d4dac05-9cef-4563-a058-f108abecce1d
        - Internet Archive: https://web.archive.org/web/20250602203910/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/7d4dac05-9cef-4563-a058-f108abecce1d
    - Microsoft MS-DTYP OpenSpects – 2.5.1 Security Descriptor Description Language: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/4f4251cc-23b6-44b6-93ba-69688422cb06
        - Internet Archive: https://web.archive.org/web/20250603001725/https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-dtyp/4f4251cc-23b6-44b6-93ba-69688422cb06
    - Microsoft PowerShell Community – Understanding Get-ACL and AD Drive Output: https://devblogs.microsoft.com/powershell-community/understanding-get-acl-and-ad-drive-output/
        - Internet Archive: https://web.archive.org/web/20250603160446/https://devblogs.microsoft.com/powershell-community/understanding-get-acl-and-ad-drive-output/
    - Microsoft SECURITY_DESCRIPTOR structure: https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-security_descriptor
        - Internet Archive: https://web.archive.org/web/20250603141128/https://learn.microsoft.com/en-us/windows/win32/api/winnt/ns-winnt-security_descriptor
    - Microsoft ADSI – IADsSecurityUtility interface: https://learn.microsoft.com/en-us/windows/win32/api/iads/nn-iads-iadssecurityutility
        - Internet Archive: https://web.archive.org/save/https://learn.microsoft.com/en-us/windows/win32/api/iads/nn-iads-iadssecurityutility

- PowerShell Gallery – SDDLParser: https://www.powershellgallery.com/packages/SDDLParser/0.5.0/Content/SDDLParserADObjects.ps1
  - Internet Archive: https://web.archive.org/web/20250603164256/https://www.powershellgallery.com/packages/SDDLParser/0.5.0/Content/SDDLParserADObjects.ps1
- The Old New Thing - Why Does Canonical Order for ACEs Put Deny ACEs Ahead of Allow ACEs?: https://devblogs.microsoft.com/oldnewthing/20070608-00/?p=26503
  - Internet Archive: https://web.archive.org/web/20250603225616/https://devblogs.microsoft.com/oldnewthing/20070608-00/?p=26503
- IsCanonical: https://learn.microsoft.com/en-us/dotnet/api/system.security.accesscontrol.commonacl.iscanonical?view=net-9.0#system-security-accesscontrol-commonacl-iscanonical
  - Internet Archive: https://web.archive.org/web/20250616210120/https://learn.microsoft.com/en-us/dotnet/api/system.security.accesscontrol.commonacl.iscanonical?view=net-9.0#system-security-accesscontrol-commonacl-iscanonical
- Secure Ideas – ACL, DACL, SACL and the ACE: https://secureidentity.se/acl-dacl-sacl-and-the-ace/
  - Internet Archive: https://web.archive.org/web/20250603181104/https://secureidentity.se/acl-dacl-sacl-and-the-ace/

- DACL Abuse:
  - Microsoft - Active Directory Access Control List – Attacks and Defense (2017): https://techcommunity.microsoft.com/t5/security-compliance-and-identity/active-directory-access-control-list-8211-attacks-and-defense/ba-p/250315
    - Internet Archive: https://web.archive.org/web/20250603181439/https://techcommunity.microsoft.com/blog/microsoft-security-blog/active-directory-access-control-list-8211-attacks-and-defense/250315
  - Microsoft Research - Heat-ray - Combating Identity Snowball Attacks (2009): https://www.microsoft.com/en-us/research/publication/heat-ray-combating-identity-snowball-attacks-using-machine-learning-combinatorial-optimization-and-attack-graphs/
    - Internet Archive: https://web.archive.org/web/20250603181732/https://www.microsoft.com/en-us/research/publication/heat-ray-combating-identity-snowball-attacks-using-machine-learning-combinatorial-optimization-and-attack-graphs/
    - Internet Archive (PDF): https://web.archive.org/web/20250603181746/https://www.microsoft.com/en-us/research/wp-content/uploads/2009/01/sosp2009-heatray-10pt.pdf
  - Control Paths in an Active Directory Environment (FR 2014): https://www.sstic.org/2014/presentation/chemins_de_controle_active_directory/

        - Internet Archive: https://web.archive.org/web/20250616210313/https://www.sstic.org/2014/presentation/chemins_de_controle_active_directory/
    - SpecterOps – An ACE Up the Sleeve (pdf): https://specterops.io/wp-content/uploads/sites/3/2022/06/an_ace_up_the_sleeve.pdf
        - Internet Archive: https://web.archive.org/web/20250612052627/https://specterops.io/wp-content/uploads/sites/3/2022/06/an_ace_up_the_sleeve.pdf
    - SpecterOps – BloodHound 1.2 The Attack Path Update: https://wald0.com/?p=112
        - Internet Archive: https://web.archive.org/web/20250616230337/https://wald0.com/?p=112
    - The Hacker Recipes – DACL Abuse: https://www.thehacker.recipes/ad/movement/dacl
        - Internet Archive: https://web.archive.org/web/20250616230848/https://www.thehacker.recipes/ad/movement/dacl/
    - Secure Ideas – Watching yOUr Permissions: https://www.secureideas.com/blog/2018/07/watching-your-permissions.html
        - Internet Archive: https://web.archive.org/web/20211019020457/https://www.secureideas.com/blog/2018/07/watching-your-permissions.html
    - SpecterOps - BloodHound Edges: https://bloodhound.specterops.io/resources/edges/overview
- Auditing:
    - 4780(S): The ACL was set on accounts which are members of administrators groups: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4780
        - Internet Archive: https://web.archive.org/web/20250603182524/https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-10/security/threat-protection/auditing/event-4780
    - A batch of Event ID 4780 are logged in the primary domain controller: https://learn.microsoft.com/en-us/troubleshoot/windows-server/active-directory/a-batch-of-event-4780-logged-pdc
        - Internet Archive: https://web.archive.org/web/20250616231517/https://learn.microsoft.com/en-us/troubleshoot/windows-server/active-directory/a-batch-of-event-4780-logged-pdc
- Other:
    - SpecterOps - The Security Principle Every Attacker Needs to Follow: https://posts.specterops.io/the-security-principle-every-attacker-needs-to-follow-905cc94ddfc6
    - Microsoft - Appendix I: Creating Management Accounts for Protected Accounts and Groups in Active Directory: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/component-updates/appendix-i--creating-management-accounts-for-protected-accounts-and-groups-in-active-directory

        - Internet Archive: https://web.archive.org/web/20250617004253/https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/manage/component-updates/appendix-i--creating-management-accounts-for-protected-accounts-and-groups-in-active-directory
    - SpecterOps – What is Tier Zero: https://posts.specterops.io/what-is-tier-zero-part-1-e0da9b7cdfca
    - Trimarc – Owner or Pwned?: https://www.hub.trimarcsecurity.com/post/trimarc-whitepaper-owner-or-pwnd
        - Internet Archive: https://web.archive.org/web/20250603140701/https://www.hub.trimarcsecurity.com/post/trimarc-whitepaper-owner-or-pwnd
        - PDF Direct Link: https://adminsdholder.com/files/Owner_or_Pwned_v1.3_JimSykora.pdf

Glossary

Access Control Entry (ACE) A rule that defines the permissions granted or denied to a specific security principal for a specific resource. See also ACE

ACE Header A structure that defines the type and size of an Access Control Entry. See also ACE_HEADER

Access Control List (ACL) A list of Access Control Entries that collectively determines the access to a specific resource. See also ACL structure

Access Mask A 32-bit value that defines the access rights defined in an ACE or an Access Request. See also 2.4.3 ACCESS_MASK or ACCESS_MASK

Access Token A data structure containing the SID for a security principal, SIDs for the groups that the security principal belongs to, and a list of the security principal's privileges (also known as user rights) on the local computer. An access token is created for every security principal who logs on locally, at the computer's keyboard, or remotely, through a network connection. The access token provides a security context for the security principal's actions on the computer. See also Access Tokens

Control Access Rights An extension of the standard access rights which allow for controlling access to a resource in ways the standard access rights do not support. Control Access Rights are used in three ways: Extended Rights, Property Sets, and Validated Writes. See also Control Access Rights

Discretionary Access Control List (DACL) A portion of a security descriptor that specifies the access rights allowed or denied to particular users or groups. See also DACLs and ACEs

Generic Access Rights In an Access Mask, the four high-order bits specify generic access rights. Each type of securable object maps these bits to a set of its standard and object-specific access rights. You can use generic access rights to specify the type of access you need when opening a handle to an object. This is typically simpler than specifying all the corresponding standard and specific rights. These high-order bits are 28: GenericAll, 29: GenericExecute, 30: GenericWrite, and 31: GenericRead. See also Generic Access Rights and Generic Access Mappings

Object Access Rights In an Access Mask, the 16 low-order bits are a set of rights specific to the object type being secured. In AD, these object access rights are known as Directory Services Access Rights including Open, CreateChild, DeleteChild, List, ReadProperties, WriteProperties, Self, DeleteTree, ListObject, and ControlAccess. Each object type or resource manager will have different object-specific access rights. The object access rights for the NTFS file system are different from registry keys, etc. See also Access Rights and Access Masks, Directory Services Access Rights, and ADS_RIGHTS_ENUM

Object-Specific ACEs Object-specific ACEs are supported for directory service (DS) objects. An object-specific ACE contains a pair of GUIDs that expand the ways in which the ACE can protect an object. The ObjectType GUID identifies either a type of child object, a property set or property, an

extended right, or a validated write. The InheritedObjectType GUID indicates the type of child object that can inherit the ACE. See also Object-Specific ACEs

Privilege The right of a user to perform various system-related operations, such as shutting down the system, loading device drivers, or changing the system time. A user's access token contains a list of the privileges held by either the user or the user's groups.

Relative Identifier (RID) Portion of a Security Identifier (SID) that identifies the unique security principal in relation to the authority that issued the SID. This variable length number is assigned to objects at creation and becomes part of the object's security identifier. The RID is generally the last set of digits in a SID after the last dash.

Security Descriptor A structure and associated data that contains the security information for a Securable Object. A security descriptor identifies the object's owner and primary group. It can also contain a DACL that controls access to the object, and a System ACL (SACL) that controls the logging of attempts to access the object. See also Security Descriptors

Security Descriptor Control A set of bit control flags on the security descriptor that qualifies the security descriptor and its components. See also SECURITY_DESCRIPTOR_CONTROL

Security Information Security information includes the owner of an object, the primary group of an object, the DACL of an object, and the SACL of an object. See also SECURITY_INFORMATION

System Access Control List (SACL) A SACL allows administrators to log attempts to access a secured object. Each ACE in a SACL specifies the types of access attempts by a specific trustee that cause the system to generate a record in the security event log.

Security Identifier (SID) Data structure of variable length that identifies users, groups, computer accounts, states of authentication, and more. Every account on a network is issued a unique SID when the account is first created. Internal processes in Windows refer to an account's SID rather than name. See also Security Identifiers Technical Overview

Securable Object A Securable Object is any object that can have a security descriptor. All named Windows objects are securable. Some unnamed objects, like thread and process objects, can have security descriptors also. Most Securable Objects define their security descriptor when created and some allow for a specified security descriptor instead of a default security descriptor. Each type of Securable Object defines its own set of access rights and mappings of generic access rights. Here are some examples of Securable Objects: Files and directories in NTFS, Named pipes, Processes, Access tokens, Registry keys, Windows services, Printers, Network shares, and Directory service objects (AD). See also Securable Objects

Security Principal A user, group, computer, or service. Security principals have accounts. Local accounts are managed by the Security Account Manager (SAM) on the computer. AD manages domain accounts. See also Security Principals Technical Overview

Standard Access Rights Each type of securable object has a set of access rights that correspond to operations specific to that type of object. In addition to these object-specific access rights, there is a set of standard access rights that correspond to operations common to most types of securable objects. These rights are mapped in bits 16-23 of an access mask and include 16: Delete, 17: ReadControl, 20: Synchronize, 18: WriteDACL, and 19: WriteOwner. Standard access rights also include mappings of combinations of standard access rights such as StandardRightsAll, StandardRightsExecute, StandardRightsRead, StandardRightsRequired, and StandardRightsWrite. See also Standard Access Rights

Trustee A user account, group account, computer account, or logon session to which an Access Control Entry (ACE) applies. See also TRUSTEE

Universal Well-Known Security Identifier Well-Known SIDs identify generic groups and users and have values that remain constant across all operating systems. For example, the Everyone and World SIDs. Universal Well-Known SIDs are known on all systems that adhere to the security model based on authenticated users and discretionary access control. See also WELL_KNOWN_SID_TYPE enumeration and Well-Known SIDs