TRIMARC

Owner or Pwned?

Discovering and Remediating Active Directory Object Ownership Issues

Abstract

The default behavior in Active Directory (AD) allows the Owner of an AD Object to fully control that Object. Do you know who owns objects in your AD Forest? Do you know which AD Object Owners could compromise your AD Forest? Do you know who could own your AD Objects and Pwn your AD?

Jim Sykora

Version 1.3

Foreword

"Owner or Pwned?" is an in-depth journey into the intricacies of ownership in Active Directory (AD). Yes, I had to look up how to spell intricacies. Trimarc's own Jim Sykora smashes a year's worth of research into 54 short pages. Complete with code snips, screenshots, examples and of course Kenny Loggins references. This whitepaper touches on all aspects of AD ownership: Organizational Units (OUs), Computers, Groups, Users, AD Certificate Services (ADCS), Group Policy Objects (GPOs), and even Active Directory Integrated DNS (ADI DNS).

Jim identifies reactive approaches to fix what's already vulnerable as well as proactive options to reconfigure AD to be more secure in the future. These fundamental shifts in design strategy remove the necessity for monthly or quarterly scripts to scan and remediate misconfigurations that persist as a default as new computers, users, and other objects are provisioned into AD.

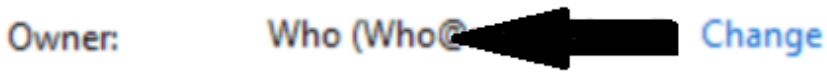
This paper drops plenty of best practices along the way but reads more like a journey. Jim walks through his thought process from start to finish showcasing the difference between doing it your way and doing it the right way. He concludes by investigating several recent patches and published research that highlight the necessity for understanding ownership. Failure to do so, as Kenny Loggins puts it, is a "highway to the danger zone".

- Brandon Colley, Identity Security Consultant at Trimarc Security

Who's the Owner?

When an Object is created in Active Directory (AD), an Owner will be assigned according to the group membership of the Object's Creator.

Note: Everything in Active Directory is an Object.



There are three different scenarios describing who will be granted Ownership of an Object upon creation in any standard Active Directory domain naming context (domain NC).

DEFINITION: A DOMAIN NC IS A SPECIFIC SET OF OBJECTS ORGANIZED AS A TREE WHICH REPRESENTS AN AD DOMAIN. A DOMAIN NC IS THE ONLY TYPE OF NC THAT CAN CONTAIN SECURITY PRINCIPAL OBJECTS. THIS IS WHAT YOU MIGHT SEE WHEN YOU OPEN ACTIVE DIRECTORY USERS AND COMPUTERS.

Note: Different rules apply for Active Directory Schema and Configuration partitions. More on that later…

Scenario 1: When an Object is created by a member of the Domain Admins group (DA), Domain Admins will be granted Ownership of the Object. Ditto for the Enterprise Admins group (EA). Oddly enough, if an Object's Creator is a member of both Domain Admins and Enterprise Admins, Domain Admins will be granted Ownership of the Object despite Enterprise Admins implied superiority. This is because the order of operations for a domain NC process DA before EA when evaluating for the "default administrators group" (DAG), which is different than the BUILTIN\Administrators aka RID500 group.

Scenario 2: When a member of the BUILTIN\Administrators group for the domain that is NOT a member of DA or EA creates an Object, it will be Owned by the user that created it – not the Administrators group.

Scenario 3: When a standard user creates an Object through delegated permissions, that user becomes the Owner of the Object.

Objects created in Scenario 1 are said to have "Standard Ownership". Additionally, Objects Owned by the BUILTIN\Administrators are considered to have standard ownership *because the Administrators group is ultimately functionally identical to the other AD Admin groups*: Domain Admins and Enterprise Admins, at least for a domain NC.

Objects created in Scenarios 2 or 3 are said to have "Non-standard Ownership", *even if* the Owner is an AD Admin!

While performing Active Directory Security Assessments (ADSA), Trimarc continues to regularly discover AD Objects with non-standard ownership on the following object types:

- Organizational Units (OU)
- Computer Objects
- Security Groups
- Users
- Active Directory Certificate Services (AD CS)/Public Key Services (PKS) objects
- Group Policy Objects
- Active Directory Integrated DNS zones

In each of these cases, delegated rights result in objects having non-standard Ownership.

Note: There is a common misconception that the default ability of Authenticated Users to add up to 10 workstations to the domain (as configured by the "Add workstations to domain" SeMachineAccountPrivilege user right & the ms-DS-MachineAccountQuota attribute) would result in non-standard ownership. However, computer accounts added via this ability result in a computer Object that is Owned by the Domain Admins group. The computer's creator is captured in the msDS-CreatorSID attribute.

To reiterate, there's a difference in Ownership on an AD Object in a domain NC when:

- A privileged user which is a member of Domain Admins creates an object: **Domain Admins group** becomes the Owner.
- A privileged user which is a member of Enterprise Admins creates an object: **Enterprise Admins group** becomes the Owner.
- A privileged user which is a member of both Domain Admins AND Enterprise Admins creates an object: **Domain Admins group** becomes the Owner.
- A privileged user which isn't a member of either Domain Admins or Enterprise Admins creates an object: the **Creator** becomes the Owner.
- An Authenticated User creates a computer object via the "Add workstations to domain" user right: **Domain Admins group** becomes the Owner.
- An Authenticated User creates a DNS node: the **Creator** becomes the Owner.
- An Authenticated User with delegated rights creates an Object: the **Creator** becomes the Owner.

The scope and scale of the issue often depends on whether the non-standard Owner is an individual delegated account, or a service account. It's not uncommon to find service accounts for computer provisioning solutions as being owners on tens of thousands of computer objects.

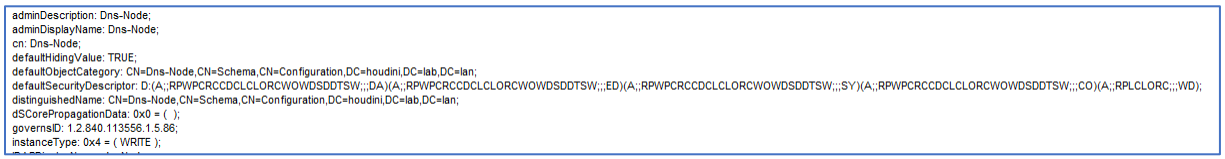
The Creator of an AD Object isn't always the Owner. Ownership can be modified in a few different ways:

- Delegated WriteOwner permissions
- Full Control (GenericAll) permissions, which includes WriteOwner permissions

- User Rights Assignment to Take ownership of files or other objects
- User Rights Assignment to Restore files and directories

This is covered in more detail in a later section "Who Else Can Be the Owner?".

The Microsoft documentation on who the Owner will be isn't always perfectly clear. It's not so much that the rules aren't clear, if you're good at jumping around in the documentation, but that the rules in this link are what happen if the new object's security descriptor isn't specified when the object is created. In most cases, from what I can tell, the default security descriptor is provided by the AD Schema, or it is provided at object creation (when allowed, which is discussed more in the LDAP Add & Modify section). However, the default security descriptor in the AD Schema doesn't specify Owner or Group information. An object class's default security descriptor only provides the Discretionary Access Control List (DACL) portion of the security descriptor, in SDDL format.



Microsoft Open specifications for Active Directory Technical Specification (MS-ADTS) provide further explanation of how the Owner and Group are defaulted on Security Descriptors. Sections 6.1.3.7 Owner and Group Defaulting Rules and 6.1.3.8 Default Administrators Group (*I told you we'd get back to this*) provide detailed information on how Ownership is defaulted.

The Owner and Group are defaulted on object creation when:

1. The Security Descriptor Flags do not include the Owner bit
2. The Security Descriptor Flags include the Owner bit, but the Owner field value is null

In either of those cases, the Owner field value is defaulted as such:

1. If the user creating the object is a member of the "default administrator group" DAG for that object, the security identifier (SID) of the DAG's group is written in the Owner field
2. Failing that, if the creator of the object's security context (or access token) contains the TokenOwner field, then the SID in that field is written to the Owner field
3. Otherwise, the creator's SID is written into the Owner field.

Depending on the functional level of the Domain Controller that services the object creation request, these rules will also determine how the Group field is populated in the Security Descriptor.

The 1st scenario when the Owner field is defaulted brings us back to the concept of a default administrators group (DAG). The DAG is used for Owner defaulting and access checks dealing with the Owner. The DAG depends on the object creator or requestor's access token composition (which groups the user is a member of and which privileges they have) and the location of the target object.

We have already discussed how these rules are applied to a domain NC. The order is Domain Admins, Enterprise Admins, and undefined. For the Configuration NC the order is Enterprise Admins, Domain Admins (of the current domain controller), undefined. For the Schema NC the order is Schema Admins, Enterprise Admins, Domain Admins (of the current domain controller), undefined. For any application NC the order is Domain Admins (of the sdReferenceDomain), Enterprise Admins, undefined.

When I originally wrote this paper, I didn't yet understand how the "default administrators group" functionality worked, but had observed it in my lab. Finding 6.1.3.8 Default Administrators Group in the MS-ADTS open specs tied it all together. If you're serious about understanding how Active Directory works, I highly recommend aiming to understand the Microsoft MS-ADTS open specs. The only thing better is hooking a debugger or working for Microsoft and having the source code.

The Danger Zone

Active Directory permissions abuses, also known as ACE/ACL or DACL abuse, have existed since the first organizations upgraded their NT 4.0 Domains to Windows 2000 AD DS domains. These abuses have become more common in recent years as knowledge of the techniques have become more mainstream and tooling to discover and abuse the permissions model more readily available. A sufficiently advanced attacker may rely on these abuses to blend in more closely with routine behavior or to work around other baseline defenses. The ability to modify permissions on an object can provide the ultimate in DACL abuse: create your own DACL.

Non-standard Owners can be dangerous because the default behavior in Active Directory is to grant Owners rights to Read and Modify Permissions (WRITE_DAC) on the Object. The ability to Modify Permissions can be abused by an attacker with control of the Owner security principal to assign any arbitrary Access Control Entry (ACE) to the object that the attacker desires. That could be granting the attacker-controlled Owner security principal Full Control (GenericAll) over the object, or the attacker could grant any other security principal rights on the object.

Note: WRITE_DAC may also allow a principal with that right to change the order of the ACEs in the DACL, which can also affect the access check process in unexpected ways.

In other words, by default, the Owner of an AD Object can easily gain Full Control on that object. This means AD Object Owners are one step away from:

- Reading the plaintext `ms-mcs-AdmPwd` LAPS credential on computers
- Modifying the `servicePrincipalName` or `ms-DS-AllowedToActOnBehalfOfOtherIdentity` attributes for Kerberos abuse
- Modifying the `msDS-KeyCredentialLink` for Shadow Credentials abuse
- Reading the `msfve-recoverypassword` attribute to access the BitLocker recovery key
- Modifying a GPO linked to domain root, Domain Controllers, or OUs with Admin accounts
- Modifying an AD CS certificate template, object, or CA host for PKI abuse
- Force-resetting a user account password
- Add a malicious logon script
- Adding members to a privileged group
- Manipulating a DNS record

In addition to the WRITE_DAC issue on all AD Objects, Computer Objects in AD have a default CREATOR OWNER ACE that allows for "Validated write to computer attributes" by the creator of the Computer Object. This allows an attacker with the capability to create new Computer Objects to modify the SPN, DNS host name, UAC value, and every other property or attribute on that Computer Object via extensive WriteProperty, Self, and owner permissions. The attacker-controlled Computer Object can then be used to pivot to other attack types such as Resource-Based Constrained Delegation abuse, SPN-jacking, or just reading the confidential attributes. By default, every Authenticated User can join up to 10 computers to the domain via the Machine Account Quota and default User Rights Assignments (ms-DS-MachineAccountQuota + SeMachineAccountPrivilege).

When the Owner is a privileged and protected security principal (other than DA/EA) the situation is only slightly more risky than standard ownership (owned by an AD Admins group), unless or until that user's privileged access is revoked. If an Owner that was a

delegated admin or member of BUILTIN\Administrators is removed from those privileged assignments, they don't automatically lose their Ownership privileges on objects.

Note: Once a user or security principal has been privileged, it should always be considered privileged for this very reason, and more. Don't remove an account from privileged groups, reset the adminCount attribute, and re-enable security descriptor inheritance on privileged accounts in an attempt to convert it to a standard user account. Always create a new standard user account and disable the former privileged account.

Another dangerous scenario is when a standard user account is an object owner. This can happen due to delegations. An attacker who compromises this standard user's credentials or workstation could then escalate themselves to local administrator on that device via LAPS, when configured in the environment.

Compromise of a service account that is an owner on many objects is another path towards lateral movement and privilege escalation. There are multiple ways an attacker can compromise a service account for the computer management solutions that often own a disproportionate amount of AD Objects in many AD Forests that Trimarc assesses. With control of the service account comes control of the objects that service account owns. This situation is common with many workstation provisioning solutions like SCCM or MDT that use a service account to join the computer to the domain.

Ownership (or the ability to write ownership) can also be abused by threat actors for AD persistence that often flies under the radar (PDF). Once AD is compromised it's possible for an attacker to set ownership on AdminSDHolder or domain root to an attacker-controlled security principal and if partially evicted, use that ownership to regain control of AD.

If all that wasn't dangerous enough:

"Additionally, the owner of an object has complete control (GenericAll equivalent) of the object, regardless of any explicit deny ACEs." – Andy Robbins (@wald0)

This means that even if there's an explicit Access Control Entry to Deny the Owner's security principal (or groups they're a member of) on an AD Object, the Owner can still Write_DAC to remove that Deny ACE and then gain access.

Non-standard Ownership isn't a new issue. It's been around as long as Active Directory has existed (25+ years). There are instances of it being discussed on social media within the past few years:

Reactive Approaches

Discovering Non-standard Owners

Discovering Non-standard Owners can be done with PowerShell. The amount of time it takes to perform a query and process the information depends heavily on the size and scope of the Active Directory environment.

For smaller environments, something like this PowerShell one-liner can be a good starting point:

```
Get-ADObject -Filter * -properties ntSecurityDescriptor | Select-Object -
Property Name, @{Name='ntSecurityDescriptorOwner';
Expression={$_.ntSecurityDescriptor.Owner }}, DistinguishedName | where {
$_.ntSecurityDescriptorOwner -notlike "*\Domain Admins" -and
$_.ntSecurityDescriptorOwner -notlike "*\Enterprise Admins" -and
$_.ntSecurityDescriptorOwner -notlike "NT AUTHORITY\SYSTEM" -and
$_.ntSecurityDescriptorOwner -notlike "BUILTIN\Administrators" }
```

Note: This query could take a long time to run in larger environments and I'm not claiming this is a well-written or optimized query. GPOs with non-standard ownership will be displayed via GUID instead of name.

This snippet will output the AD Object Name, the Owner, and the Object's Distinguished Name. The output may require a little filtering to remove Read-only Domain Controllers (RODCs) that have ownership on RODC-related objects. But largely the non-standard ownership should be accurate and ready to remediate.

For larger Active Directory environments, it may be necessary to break up queries by object type, filter by OU, perform queries domain-by-domain, and output results to CSV for further filtering. Or you could engage Trimarc to perform an Active Directory Security Assessment for your AD Forest(s) and non-standard object ownership will be part of the assessment and report.

Remediating Non-Standard Ownership

When a domain has only a handful of non-standard owners it's not overly challenging to use Active Directory Users and Computers (ADUC) to change an object's owner to Domain Admins. When it comes to setting standard ownership on many objects, it's time to look at PowerShell:

```
# Build the AD Object Path based on DistinguishedName
$ObjectPath = ("AD:" + (Get-ADUser <UserName>).DistinguishedName)
# Retrieve the current object ACL
$ACL = Get-ACL -Path $ObjectPath
# Specify the Domain Admins group for the current domain
$NewOwner = New-Object System.Security.Principal.NTAccount((Get-
ADDomain).NetBIOSName, "Domain Admins")
# Apply the NewOwner to the $ACL object in memory
$ACL.SetOwner($NewOwner)
# Apply the updated ACL to AD
Set-ACL -Path $ObjectPath $ACL
```

This snippet will need to be run as an AD Admin in the correct domain in order to apply standard ownership to <UserName>. This could be modified to take input from the previous snippet and iteratively set the correct standard ownership on all items.

This snippet will not iterate through computer objects to remove permissions that were assigned to the CREATOR OWNER or resolve any modifications to the DACL by the owner that may have already occurred.

Proactive Approaches

There are a couple of approaches to proactively lessen the impact of non-standard object ownership in Active Directory: Owner Rights and dSHeuristics.

Owner Rights

Owner Rights is a well-known security principal (SID S-1-3-4) that was introduced with Windows Server 2008 AD DS. It's well-known in that it's a special entity known by the Windows security subsystem and available in all Active Directory domains running on at least Windows Server 2008 domain controllers. It's seemingly not well-known yet among AD operations and security teams despite being around for about 15 years. To the best of my knowledge Trimarc hasn't yet seen Owner Rights configured on an ACE in an ADSA engagement. I became aware of the Owner Rights group after reading a great blog post by Daniel Ulrichs.

Per Microsoft, "*Owner Rights is a well-known security principal that you can add to the DACL of an object to specify the permissions that are assigned to owners of objects in the directory service. This added security feature overrides the default behavior of owners of objects in the system. Because owners of objects (as specified in the security descriptor of the object) have*

WRITE_DAC permission, they can give rights to themselves and to other security principals as they see fit."

The Owner Rights principal can be found in AD at: CN=Owner Rights,CN=WellKnown Security Principals,CN=Configuration,<localdomain>

Owner Rights aren't applied by default. The Owner Rights principal needs to be specifically applied to object DACLs with an ACE. Preferably the Owner Rights principal is inherited from the DACLs of a well-designed OU structure. If Owner Rights isn't applied to a DACL, it won't take effect and the default pre-Windows 2008 behavior of Owners having WRITE_DAC will be in place regardless of the current Domain Functional Level or Domain Controller OS version.

When Owner Rights security principal is applied to objects it is possible to specify custom permissions for the Owner. Read Permissions or even an empty (being careful not to create a NULL DACL) permissions would be a good option.

I recommend reviewing the scenarios in the Owner Rights link, but I've used a lab environment (Windows Server 2019 DC, fully patched, Windows2016 domain & forest functional level) showing how it works. I've done everything through PowerShell to make the tests repeatable and accurate across multiple lab environments. There are probably more efficient ways to do these PowerShell bits, but it gets the job done here. The PowerShell snippets I use are available on GitHub.

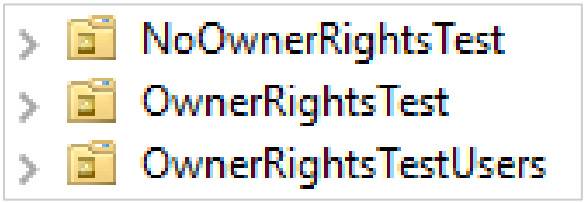
1. Set up an OU structure

This OU structure includes 3 separate OUs. A placeholder OU to contain our test users and groups, a Test OU to determine the effects of the Owner Rights security principal, and a Control OU to demonstrate the Active Directory defaults.

Note: PowerShell snippets for steps 1-4 are located on GitHub as Create-ObjectOwnerFoundation.ps1

Figure 1 PowerShell snippet to create a lab OU structure and screenshot of resulting OU structure:

```powershell
<# This snippet lays down the foundation of the lab environment. It creates OUs, groups, delegations, users, and
group membership. Run This First#>

#Password is utilized later in the snippets
$Password = Read-Host "Enter a password for test users:"  -AsSecureString

$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Create-ObjectOwnerFoundation.txt -append

$Domain = Get-ADDomain
$DomainRoot = $Domain.DistinguishedName
$TargetOU = (Get-ADDomain).DistinguishedName
$TestUserOU = "OU=OwnerRightsTestUsers,"+$TargetOU
$TargetOU1 = "OU=OwnerRightsTest,"+$TargetOU
$TargetOU2 = "OU=NoOwnerRightsTest,"+$TargetOU

# Create OUs for Testing AD Object Ownership
    # OU for the test users and delegation groups
    New-ADOrganizationalUnit "OwnerRightsTestUsers" -Path $DomainRoot
    #Test OU where Owner Rights will have an ACE
    New-ADOrganizationalUnit "OwnerRightsTest" -Path $DomainRoot
    #Control OU
    New-ADOrganizationalUnit "NoOwnerRightsTest" -Path $DomainRoot
    New-ADGroup -Name "URASeRestorePrivilege" -SamAccountName "URASeRestorePrivilege" -GroupCategory Security -
GroupScope Global -Path $TestUserOU
```



2. Create Security Groups

These security groups will be used for custom delegation in this AD Object Ownership testing and will have test users added to them in a future step.

Figure 2 PowerShell snippet to create groups for delegation purposes and screenshot of results:

```powershell
# Create Security Groups for delegation for AD Object Ownership Testing
    # This group will be delegated Full Control at the domain root.  Note: This is horribly insecure and nobody should do this!
    New-ADGroup -Name "DelegatedFullControlDomain" -SamAccountName "DelegatedFullControlDomain" -GroupCategory Security -GroupScope Global -Path $TestUserOU
    # This group will be delegated Full Control on both OUs
    New-ADGroup -Name "DelegatedFullControlOU" -SamAccountName "DelegatedFullControlOU" -GroupCategory Security -GroupScope Global -Path $TestUserOU
    # This group will be delegated the right to join workstations to the domain
    New-ADGroup -Name "DelegatedJoinWorkstationDomain" -SamAccountName "DelegatedJoinWorkstationDomain" -GroupCategory Security -GroupScope Global -Path $TestUserOU
    # This group will be delegated the right to creat computer objects in both OUs
    New-ADGroup -Name "DelegatedOUCreateComputer" -SamAccountName "DelegatedOUCreateComputer" -GroupCategory Security -GroupScope Global -Path $TestUserOU
    # This group will be added to a GPO linked to the Domain Controllers OU with the User Rights Assignment to join workstations to the domain
    New-ADGroup -Name "URAJoinWorkstationDomain" -SamAccountName "URAJoinWorkstationDomain" -GroupCategory Security -GroupScope Global -Path $TestUserOU
    # This group will be added to a GPO linked to the Domain Controllers OU with the User Rights Assignment to SeTakeOwnershipPrivilege:  Take ownership of files or other objects
    New-ADGroup -Name "URASeTakeOwnershipPriv" -SamAccountName "URASeTakeOwnershipPriv" -GroupCategory Security -GroupScope Global -Path $TestUserOU
    # This group will be added to a GPO linked to the Domain Controllers OU with the User Rights Assignment to SeRestorePrivilege:  Restore files and directories
    New-ADGroup -Name "URASeRestorePrivilege" -SamAccountName "URASeRestorePrivilege" -GroupCategory Security -GroupScope Global -Path $TestUserOU
```

Name	Type	Description
DelegatedFullControlDomain	Security Group - Global	
DelegatedFullControlOU	Security Group - Global	
DelegatedJoinWorkstationDomain	Security Group - Global	
DelegatedOUCreateComputer	Security Group - Global	
URAJoinWorkstationDomain	Security Group - Global	
URASeRestorePrivilege	Security Group - Global	
URASeTakeOwnershipPriv	Security Group - Global	

3. Delegate Permissions

This step includes adding the well-known security principal Owner Rights with a very limited permission set to the Test OU, but not the control OU. The rest of the delegations allow the test users to perform privileged actions in AD without being AD Admins in the domain, similar to how Trimarc often sees Help Desk or other lower-privileged roles delegated. (*Note: Some of these delegations are dangerous and are only done here for the purposes of the lab.*)

Figure 3 PowerShell snippet to delegate permissions to newly created groups:

```powershell
# Delegate Permissions for AD Object Ownership Testing (NOTE: Many of these delegations are dangerous and I'm only doing them here in a lab for testing purposes)
    # Delegate Owner Rights Read Permissions on TargetOU1
    $OrganizationalUnit = $TargetOU1
    $GroupName = "Owner Rights"

    Set-Location AD:
    # Need to manually add the SID here due to Owner Rights being a well-known identity.
    $GroupSID =  New-Object System.Security.Principal.SecurityIdentifier("S-1-3-4")
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "ListChildren"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

    # Delegate Full Control on Domain
    $OrganizationalUnit = $DomainRoot
    $GroupName = "DelegatedFullControlDomain"

    Set-Location AD:
    $Group = Get-ADGroup -Identity $GroupName
    $GroupSID = [System.Security.Principal.SecurityIdentifier] $Group.SID
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "GenericAll"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

    # Delegate Full Control on TargetOU1
    $OrganizationalUnit = $TargetOU1
    $GroupName = "DelegatedFullControlOU"

    Set-Location AD:
    $Group = Get-ADGroup -Identity $GroupName
    $GroupSID = [System.Security.Principal.SecurityIdentifier] $Group.SID
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "GenericAll"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

    # Delegate Full Control on TargetOU2
    $OrganizationalUnit = $TargetOU2
    $GroupName = "DelegatedFullControlOU"

    Set-Location AD:
    $Group = Get-ADGroup -Identity $GroupName
    $GroupSID = [System.Security.Principal.SecurityIdentifier] $Group.SID
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "GenericAll"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

    # Delegate Join Workstation to Domain
    $OrganizationalUnit = $DomainRoot
    $GroupName = "DelegatedJoinWorkstationDomain"

    Set-Location AD:
    $Group = Get-ADGroup -Identity $GroupName
    $GroupSID = [System.Security.Principal.SecurityIdentifier] $Group.SID
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Computers = [GUID]"bf967a86-0de6-11d0-a285-00aa003049e2"
    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "CreateChild"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $Computers, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

    # Delegate Create Computer on TargetOU1
    $OrganizationalUnit = $TargetOU1
    $GroupName = "DelegatedOUCreateComputer"

    Set-Location AD:
    $Group = Get-ADGroup -Identity $GroupName
    $GroupSID = [System.Security.Principal.SecurityIdentifier] $Group.SID
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Computers = [GUID]"bf967a86-0de6-11d0-a285-00aa003049e2"
    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "GenericAll"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $Computers, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

    # Delegate Create Computer on TargetOU2
    $OrganizationalUnit = $TargetOU2
    $GroupName = "DelegatedOUCreateComputer"

    Set-Location AD:
    $Group = Get-ADGroup -Identity $GroupName
    $GroupSID = [System.Security.Principal.SecurityIdentifier] $Group.SID
    $ACL = Get-Acl -Path $OrganizationalUnit

    $Computers = [GUID]"bf967a86-0de6-11d0-a285-00aa003049e2"
    $Identity = [System.Security.Principal.IdentityReference] $GroupSID
    $ADRight = [System.DirectoryServices.ActiveDirectoryRights] "GenericAll"
    $Type = [System.Security.AccessControl.AccessControlType] "Allow"
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
    $Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type, $Computers, $InheritanceType)

    $ACL.AddAccessRule($Rule)
    Set-Acl -Path $OrganizationalUnit -AclObject $ACL

```

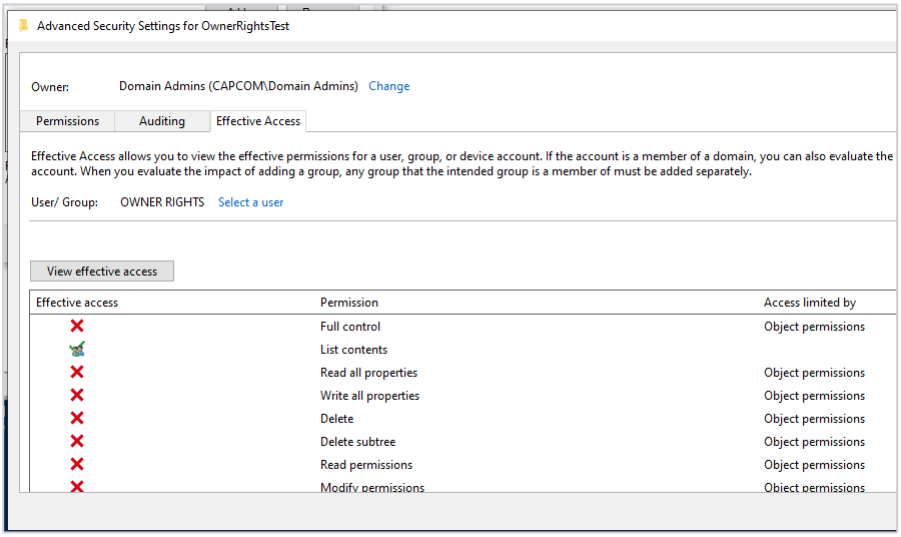
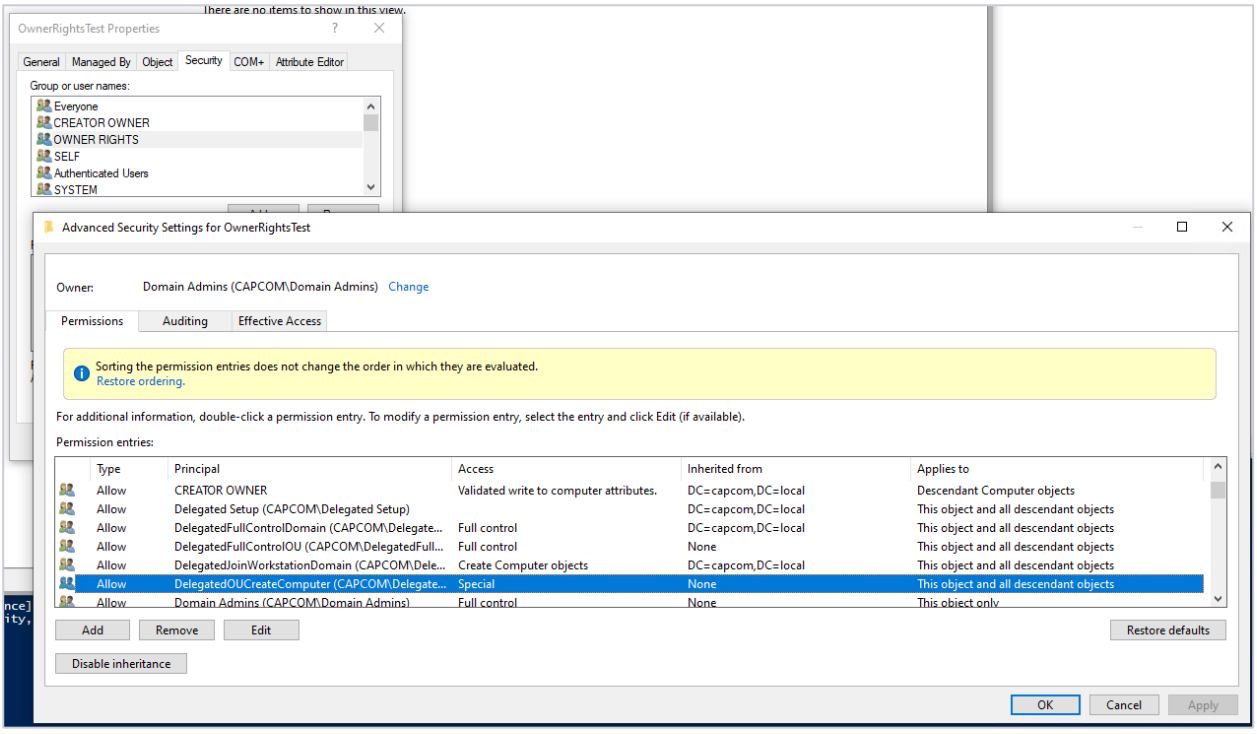
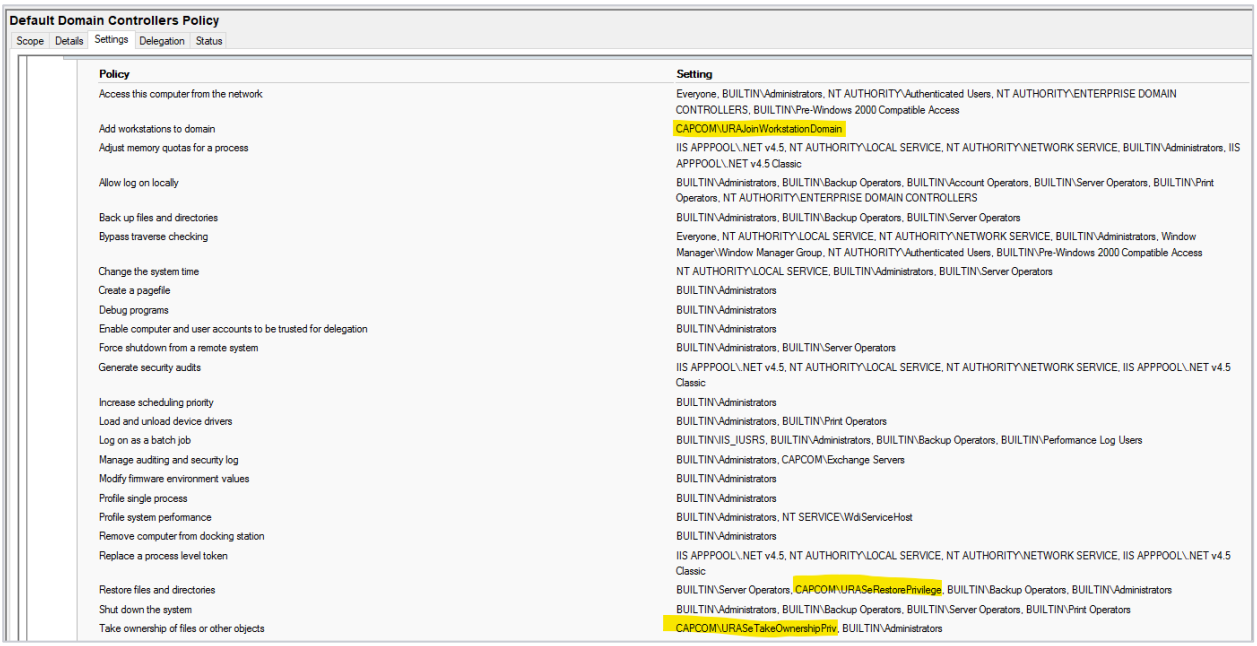


Figure 4: Owner Rights has List contents permissions on the OwnerRightsTest OU



Note that CREATOR OWNER has Validated write to computer attributes on Descendent Computer objects. This will show up later and it was discussed in The Danger Zone section above.

Figure 5 Screenshot of User Rights Assignment delegation with groups highlighted:



4. Create Test User Accounts

Here we create all our test users and add them to their corresponding security groups to gain administrative or delegated privileges. I've also added the non-admin users to the Remote Management Users group so that future PSRemote commands will work properly. This is only being done for the purposes of this lab and, as noted, is not safe to do in a production environment. (*Note: Creating multiple accounts with the same password is not a good idea and is only done here for lab purposes. There are also other ways this could have been scripted out without using PSRemote, such as runas.exe /netonly.*)

Figure 6 Table of users with their initial permissions:

Name	SAMAccountName	Privilege(s)
OwnershipTest Admin	OwnershipTestAdmin	MemberOf BUILTIN\Administrators for Domain
OwnershipTest AO	OwnershipTestAO	MemberOf Account Operators
OwnershipTest AuthUsers	OwnershipTestAU	None (Domain Users)
OwnershipTest DA	OwnershipTestDA	MemberOf Domain Admins
OwnershipTest DFCD	OwnershipTestDFCD	Delegated Full Control rights at Domain root
OwnershipTest DFCO	OwnershipTestDFCO	Delegated Full Control rights at OU level
OwnershipTest DJWD	OwnershipTestDJWD	Delegated Join Workstation to Domain at Domain root
OwnershipTest DOCC	OwnershipTestDOCC	Delegated Create Computer at OU level
OwnershipTest EA	OwnershipTestEA	MemberOf Enterprise Admins
OwnershipTest SA1	OwnershipTestSA1	MemberOf Server Operators
OwnershipTest URAJWD	OwnershipTestURAJWD	GPO User Rights Assignment to Join Workstation to Domain (same as default Authenticated Users)
OwnershipTest URARP	OwnershipTestURARP	GPO User Rights Assignment to SeRestorePrivilege
OwnershipTest URATO	OwnershipTestURATO	GPO User Rights Assignment to SeTakeOwnershipPrivilege
Attacker Controlled	AttackerControlled	None (Domain Users)

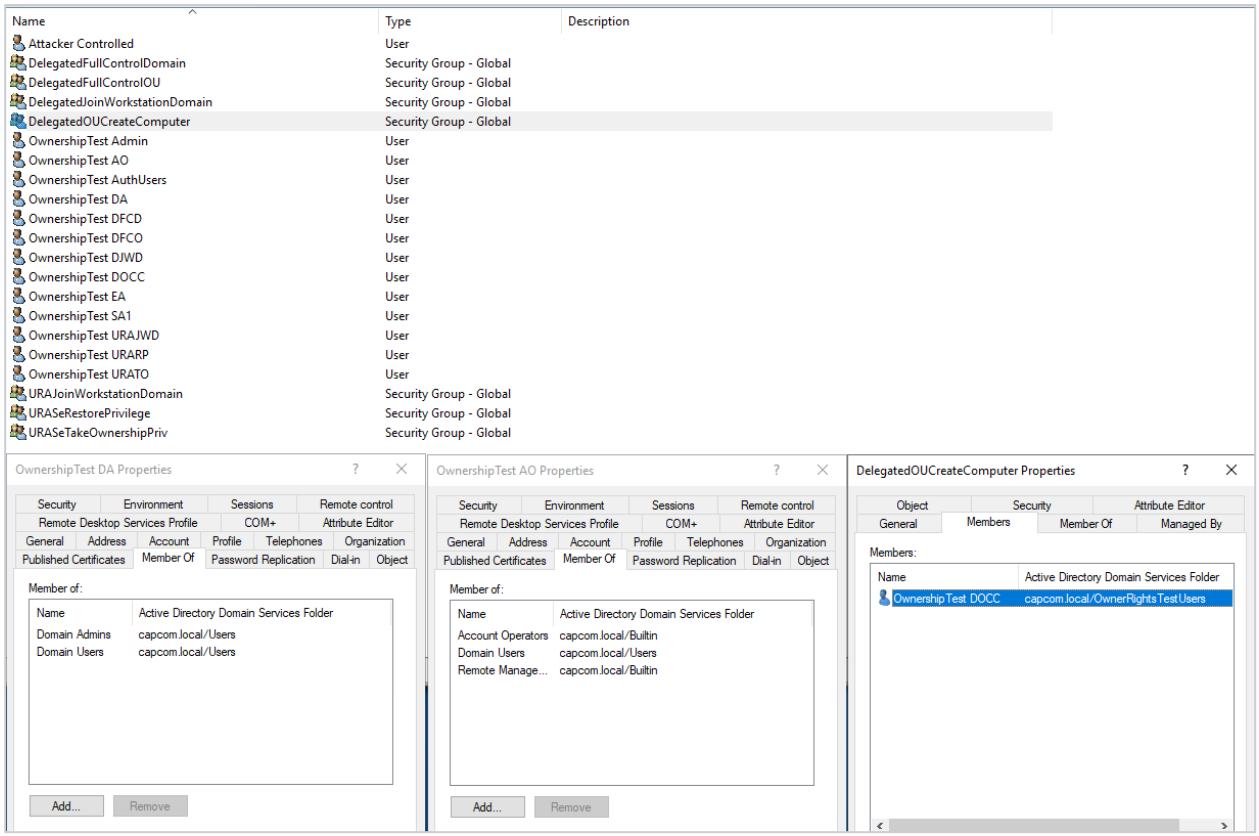
Figure 7 PowerShell snippet to create test users:

```powershell
# Create User accounts AD Object Ownership testing  (NOTE: It's an awful idea to set users with the same credential. This is for lab purposes only!!)
New-ADUser -Name "OwnershipTest DA" -SamAccountName "OwnershipTestDA" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest EA" -SamAccountName "OwnershipTestEA" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest Admin" -SamAccountName "OwnershipTestAdmin" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest AO" -SamAccountName "OwnershipTestAO" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest DFCD" -SamAccountName "OwnershipTestDFCD" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest DFCO" -SamAccountName "OwnershipTestDFCO" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest DJWD" -SamAccountName "OwnershipTestDJWD" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest DOCC" -SamAccountName "OwnershipTestDOCC" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest URAJWD" -SamAccountName "OwnershipTestURAJWD" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest AuthUsers" -SamAccountName "OwnershipTestAU" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest URATO" -SamAccountName "OwnershipTestURATO" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest URARP" -SamAccountName "OwnershipTestURARP" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "OwnershipTest SA1" -SamAccountName "OwnershipTestSA1" -AccountPassword $Password -Enabled $true -Path $TestUserOU
New-ADUser -Name "Attacker Controlled" -SamAccountName "AttackerControlled" -AccountPassword $Password -Enabled $true -Path $TestUserOU
```

Figure 8 PowerShell snippet to add users to groups:

```powershell
# Add Users to Groups for AD Object Ownership Testing
Add-ADGroupMember -Identity "Domain Admins" -Members OwnershipTestDA
Add-ADGroupMember -Identity "Enterprise Admins" -Members OwnershipTestEA
Add-ADGroupMember -Identity "Administrators" -Members OwnershipTestAdmin
Add-ADGroupMember -Identity "Account Operators" -Members OwnershipTestAO
Add-ADGroupMember -Identity "DelegatedFullControlDomain" -Members OwnershipTestDFCD
Add-ADGroupMember -Identity "DelegatedFullControlOU" -Members OwnershipTestDFCO
Add-ADGroupMember -Identity "DelegatedJoinWorkstationDomain" -Members OwnershipTestDJWD
Add-ADGroupMember -Identity "DelegatedOUCreateComputer" -Members OwnershipTestDOCC
Add-ADGroupMember -Identity "URAJoinWorkstationDomain" -Members OwnershipTestURAJWD
Add-ADGroupMember -Identity "URASeTakeOwnershipPriv" -Members OwnershipTestURATO
Add-ADGroupMember -Identity "URASeRestorePrivilege" -Members OwnershipTestURARP
Add-ADGroupMember -Identity "Server Operators" -Members OwnershipTestSA1
# Allow the newly created users to Invoke-Command on DCs without being AD Admins  NOTE: This is not a good idea and I'm only doing this for lab purposes!!!!
Add-ADGroupMember -Identity "Remote Management Users" -Members OwnershipTestDFCD, OwnershipTestDFCO, OwnershipTestDJWD, OwnershipTestDFCD, OwnershipTestDOCC, OwnershipTestURAJWD, OwnershipTestAU, OwnershipTestURATO, OwnershipTestURARP, OwnershipTestSA1, OwnershipTestAO

Stop-Transcript
```

Here we see the resulting test users as members of their respective groups:

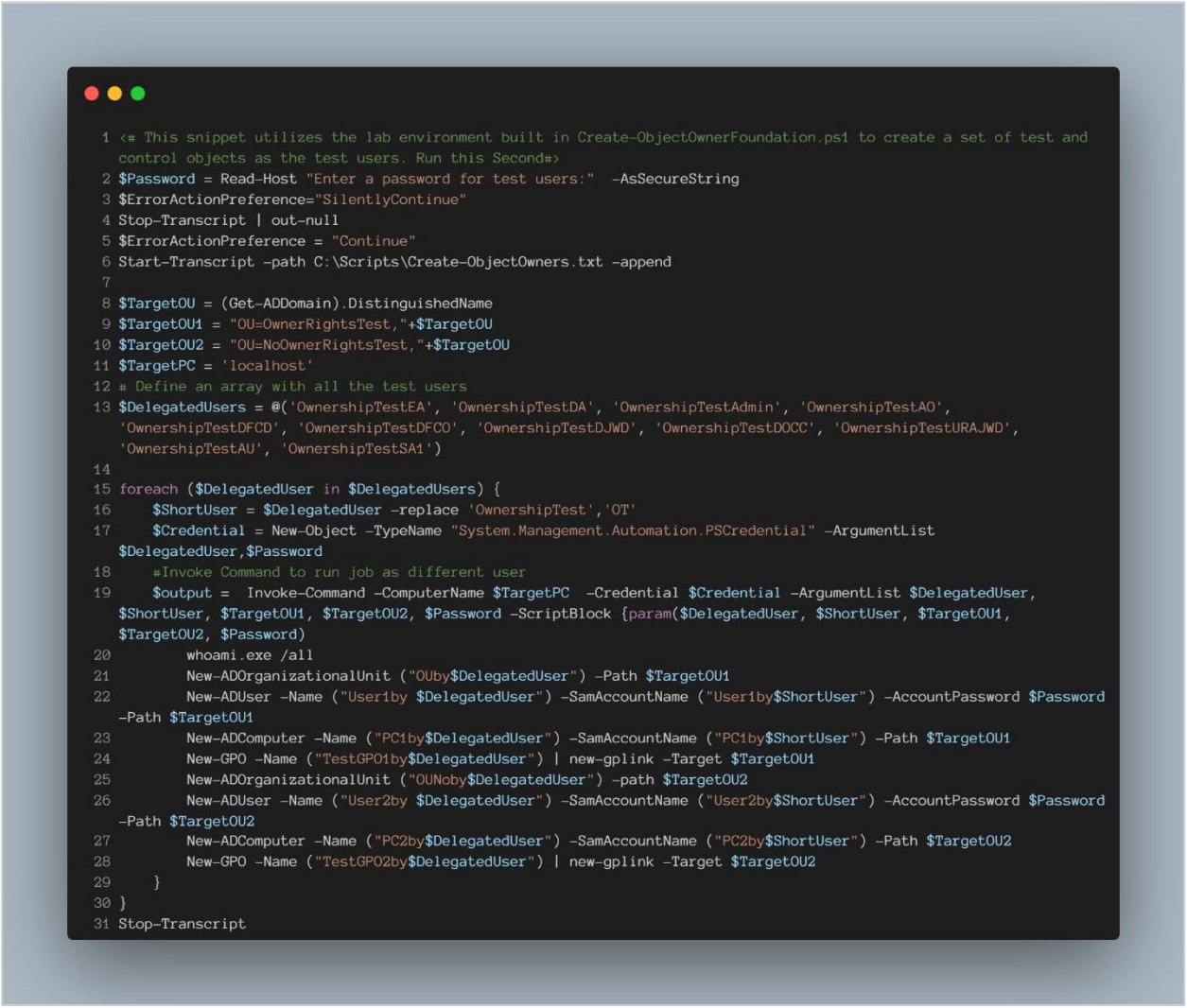


5. Create AD Objects

Here we utilize Invoke-Command as the test users created above to attempt to create multiple objects as that user possessing delegated or admin privileges in both target OUs. This will throw some errors as not every account is delegated the permissions it needs to perform every activity. All the steps prior to this are just setting up the environment. Here's where the Ownership magic starts.

Note: This PowerShell snippet is located on GitHub as Create-ObjectOwners.ps1

Figure 9 PowerShell snippet to create various AD objects using the context of all the test user accounts:



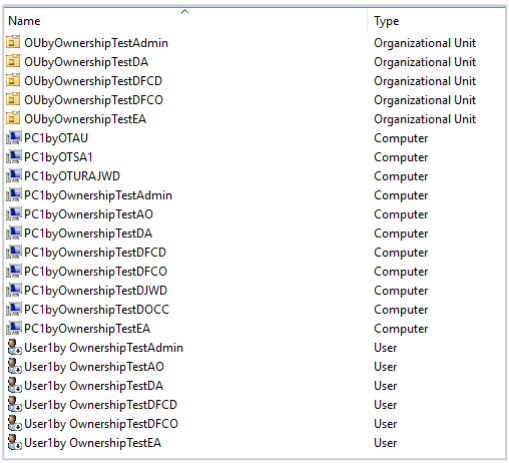
```powershell
<# This snippet utilizes the lab environment built in Create-ObjectOwnerFoundation.ps1 to create a set of test and control objects as the test users. Run this Second#>
$Password = Read-Host "Enter a password for test users:"  -AsSecureString
$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Create-ObjectOwners.txt -append

$TargetOU = (Get-ADDomain).DistinguishedName
$TargetOU1 = "OU=OwnerRightsTest,"+$TargetOU
$TargetOU2 = "OU=NoOwnerRightsTest,"+$TargetOU
$TargetPC = 'localhost'
# Define an array with all the test users
$DelegatedUsers = @('OwnershipTestEA', 'OwnershipTestDA', 'OwnershipTestAdmin', 'OwnershipTestAO', 'OwnershipTestDFCD', 'OwnershipTestDFCO', 'OwnershipTestDJWD', 'OwnershipTestDOCC', 'OwnershipTestURAJWD', 'OwnershipTestAU', 'OwnershipTestSA1')

foreach ($DelegatedUser in $DelegatedUsers) {
    $ShortUser = $DelegatedUser -replace 'OwnershipTest','OT'
    $Credential = New-Object -TypeName "System.Management.Automation.PSCredential" -ArgumentList $DelegatedUser,$Password
    #Invoke Command to run job as different user
    $output =  Invoke-Command -ComputerName $TargetPC  -Credential $Credential -ArgumentList $DelegatedUser, $ShortUser, $TargetOU1, $TargetOU2, $Password -ScriptBlock {param($DelegatedUser, $ShortUser, $TargetOU1, $TargetOU2, $Password)
        whoami.exe /all
        New-ADOrganizationalUnit ("OUby$DelegatedUser") -Path $TargetOU1
        New-ADUser -Name ("User1by $DelegatedUser") -SamAccountName ("User1by$ShortUser") -AccountPassword $Password -Path $TargetOU1
        New-ADComputer -Name ("PC1by$DelegatedUser") -SamAccountName ("PC1by$ShortUser") -Path $TargetOU1
        New-GPO -Name ("TestGPO1by$DelegatedUser") | new-gplink -Target $TargetOU1
        New-ADOrganizationalUnit ("OUNoby$DelegatedUser") -path $TargetOU2
        New-ADUser -Name ("User2by $DelegatedUser") -SamAccountName ("User2by$ShortUser") -AccountPassword $Password -Path $TargetOU2
        New-ADComputer -Name ("PC2by$DelegatedUser") -SamAccountName ("PC2by$ShortUser") -Path $TargetOU2
        New-GPO -Name ("TestGPO2by$DelegatedUser") | new-gplink -Target $TargetOU2
    }
}
Stop-Transcript
```

I also utilized Powermad's New-MachineAccount function to create several computer accounts similar to how an attacker might accomplish this. Here I also used Invoke-Command to run under the context of test users which had only permissions under the default User Rights Assignment for Add workstation to domain (SeMachineAccountPrivilege + ms-DS-MachineAccountQuota). Since this function adds the new computer objects to the default Computers container (unless redircmp is in place), I moved them to their respective OUs for the next phases of the test.

Note: Windows Defender and most other AV will trigger on the full Powermad module but may not trigger on individual functions.

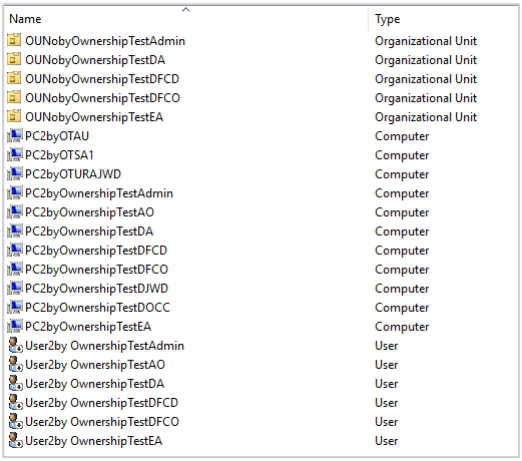
Now we have our Test OU (`OU=OwnershipTest`) and Control OU (`OU=NoOwnershipTest`) with child OUs and AD Objects created by the test users:

Figure 10 Contents of OwnershipTest OU, where Owner Rights ACE is assigned.



Name	Type
OUbyOwnershipTestAdmin	Organizational Unit
OUbyOwnershipTestDA	Organizational Unit
OUbyOwnershipTestDFCD	Organizational Unit
OUbyOwnershipTestDFCO	Organizational Unit
OUbyOwnershipTestEA	Organizational Unit
PC1byOTAU	Computer
PC1byOTSA1	Computer
PC1byOTURAJWD	Computer
PC1byOwnershipTestAdmin	Computer
PC1byOwnershipTestAO	Computer
PC1byOwnershipTestDA	Computer
PC1byOwnershipTestDFCD	Computer
PC1byOwnershipTestDFCO	Computer
PC1byOwnershipTestDJWD	Computer
PC1byOwnershipTestDOCC	Computer
PC1byOwnershipTestEA	Computer
User1by OwnershipTestAdmin	User
User1by OwnershipTestAO	User
User1by OwnershipTestDA	User
User1by OwnershipTestDFCD	User
User1by OwnershipTestDFCO	User
User1by OwnershipTestEA	User

Figure 11 Contents of NoOwnershipTest OU, where no special ACE is assigned.



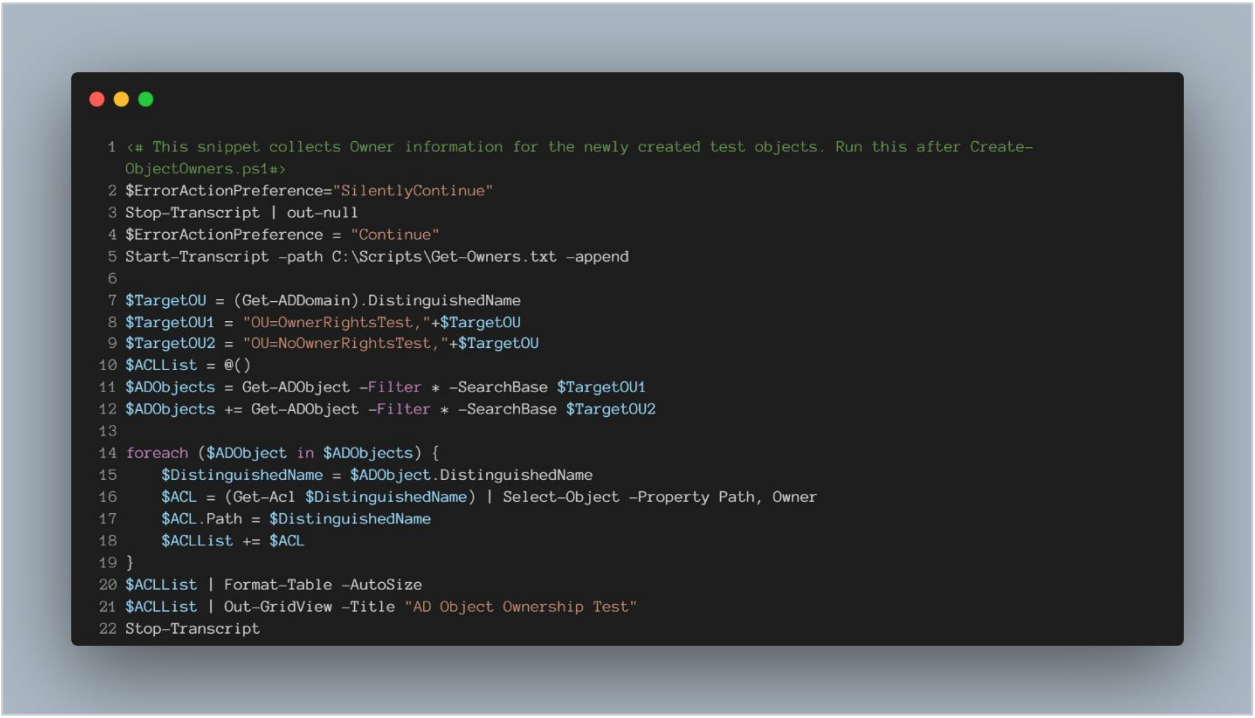
Name	Type
OUNobyOwnershipTestAdmin	Organizational Unit
OUNobyOwnershipTestDA	Organizational Unit
OUNobyOwnershipTestDFCD	Organizational Unit
OUNobyOwnershipTestDFCO	Organizational Unit
OUNobyOwnershipTestEA	Organizational Unit
PC2byOTAU	Computer
PC2byOTSA1	Computer
PC2byOTURAJWD	Computer
PC2byOwnershipTestAdmin	Computer
PC2byOwnershipTestAO	Computer
PC2byOwnershipTestDA	Computer
PC2byOwnershipTestDFCD	Computer
PC2byOwnershipTestDFCO	Computer
PC2byOwnershipTestDJWD	Computer
PC2byOwnershipTestDOCC	Computer
PC2byOwnershipTestEA	Computer
User2by OwnershipTestAdmin	User
User2by OwnershipTestAO	User
User2by OwnershipTestDA	User
User2by OwnershipTestDFCD	User
User2by OwnershipTestDFCO	User
User2by OwnershipTestEA	User

6. Review the Newly Created AD Objects

Here we'll verify our statements from earlier sections and see who the Owners are on the objects that the test users were created with Invoke-Commands above. I'll put the data into a table to make it more legible.

Note: The PowerShell snippet for this is located in GitHub as [Get-ObjectOwnerInfo.ps1]

Figure 12 PowerShell snippet to pull ownership data from the test OUs:



```powershell
<# This snippet collects Owner information for the newly created test objects. Run this after Create-ObjectOwners.ps1#>
$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Get-Owners.txt -append

$TargetOU = (Get-ADDomain).DistinguishedName
$TargetOU1 = "OU=OwnerRightsTest,"+$TargetOU
$TargetOU2 = "OU=NoOwnerRightsTest,"+$TargetOU
$ACLList = @()
$ADObjects = Get-ADObject -Filter * -SearchBase $TargetOU1
$ADObjects += Get-ADObject -Filter * -SearchBase $TargetOU2

foreach ($ADObject in $ADObjects) {
    $DistinguishedName = $ADObject.DistinguishedName
    $ACL = (Get-Acl $DistinguishedName) | Select-Object -Property Path, Owner
    $ACL.Path = $DistinguishedName
    $ACLList += $ACL
}
$ACLList | Format-Table -AutoSize
$ACLList | Out-GridView -Title "AD Object Ownership Test"
Stop-Transcript
```

Raw text output from Get-ObjectOwnerInfo.ps1

Path	Owner
OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC1byOwnershipTestDJWD,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDJWD**
CN=PC1byOwnershipTestDOCC,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDOCC**
CN=PC1byOTURAJWD,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC1byOTAU,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC1byOTSA1,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
OU=OUbyOwnershipTestEA,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Enterprise Admins
CN=User1by OwnershipTestEA,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Enterprise Admins
CN=PC1byOwnershipTestEA,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Enterprise Admins
OU=OUbyOwnershipTestDA,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=User1by OwnershipTestDA,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC1byOwnershipTestDA,OU=OwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
OU=OUbyOwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAdmin**
CN=User1by OwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAdmin**
CN=PC1byOwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAdmin**
CN=User1by OwnershipTestAO,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAO**
CN=PC1byOwnershipTestAO,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAO**
OU=OUbyOwnershipTestDFCD,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCD**
CN=User1by OwnershipTestDFCD,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCD**
CN=PC1byOwnershipTestDFCD,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCD**
OU=OUbyOwnershipTestDFCO,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCO**

CN=User1by OwnershipTestDFCO,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCO**
CN=PC1byOwnershipTestDFCO,OU=OwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCO**
OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=User2by OwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCO**
CN=PC2byOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCO**
CN=PC2byOwnershipTestDJWD,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDJWD**
CN=PC2byOwnershipTestDOCC,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDOCC**
CN=PC2byOTURAJWD,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC2byOTAU,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC2byOTSA1,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
OU=OUNobyOwnershipTestEA,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Enterprise Admins
CN=User2by OwnershipTestEA,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Enterprise Admins
CN=PC2byOwnershipTestEA,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Enterprise Admins
OU=OUNobyOwnershipTestDA,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=User2by OwnershipTestDA,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
CN=PC2byOwnershipTestDA,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\Domain Admins
OU=OUNobyOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAdmin**
CN=User2by OwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAdmin**
CN=PC2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAdmin**
CN=User2by OwnershipTestAO,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAO**
CN=PC2byOwnershipTestAO,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestAO**
OU=OUNobyOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCD**
CN=User2by OwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCD**
CN=PC2byOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCD**
OU=OUNobyOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local	**CAPCOM\OwnershipTestDFCO**

Note that the OUs and AD Objects created by members of Domain Admins or Enterprise Admins have those security principals as Owner, whereas AD Objects created with delegated privileges or membership in the Administrators or Account Operators groups have the creator as the sole Owner (bolded rows). This validates what we stated in the Who's the Owner? section above.

We also see here that not every test user was able to create all of their objects. This is due to the way we set up group memberships and privilege delegation and is expected.

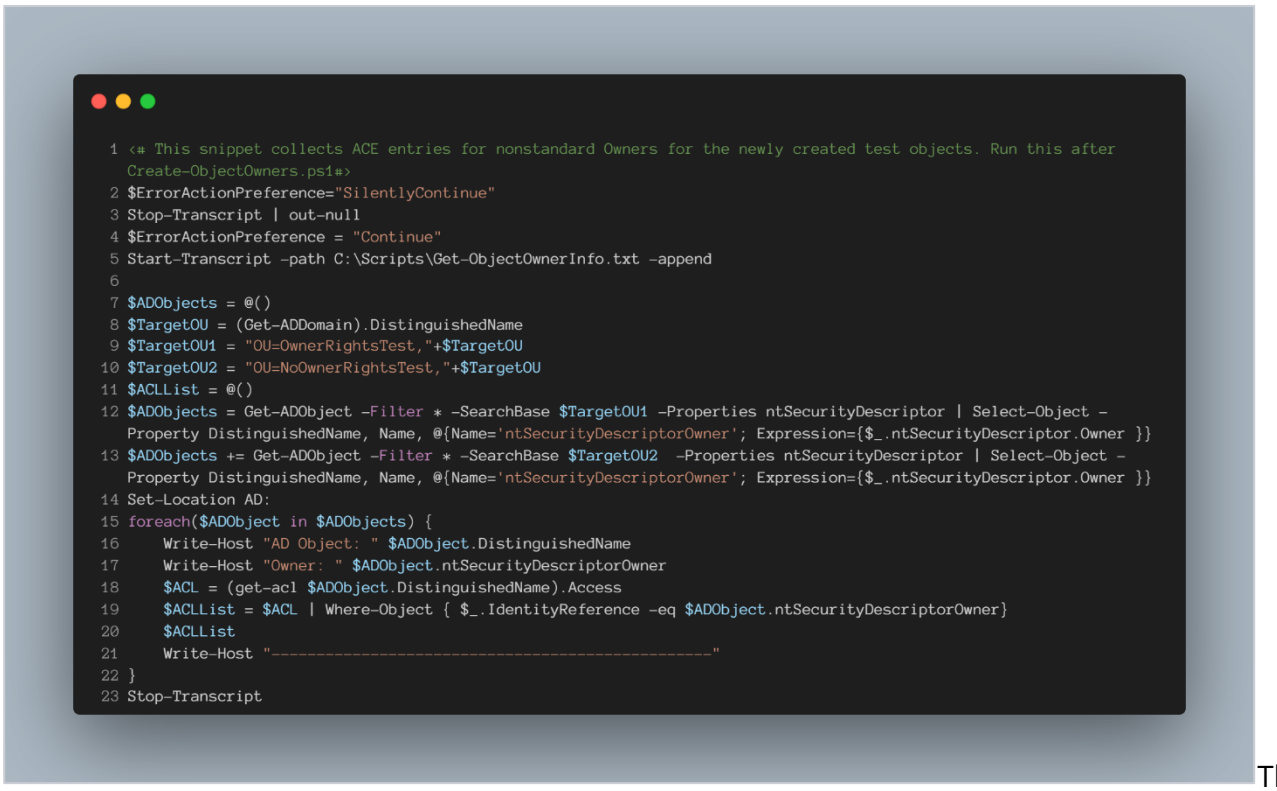
But remember that by default, every Authenticated User has rights to Add workstations to the domain, which is what the rows in red font show. Note that these computer objects do not have the creator as the owner. This is expected per the "Add workstations to domain" documentation:

> *"Furthermore, machine accounts that are created through the Add workstations to domain user right have Domain Administrators as the owner of the machine account. Machine accounts that are created through permissions on the computer's container use the creator as the owner of the machine account. If a user has permissions on the container and also has the Add workstation to domain user right, the device is added based on the computer container permissions rather than the user right."*

7. Review ACLs

Here we'll gather some information about the ACLs on the newly created objects, paying special attention to those with non-standard ownership, as demonstrated in the previous step.

This PowerShell snippet will iterate through the AD Objects created by the test users and look for ACEs where the IdentityReference is the Owner:



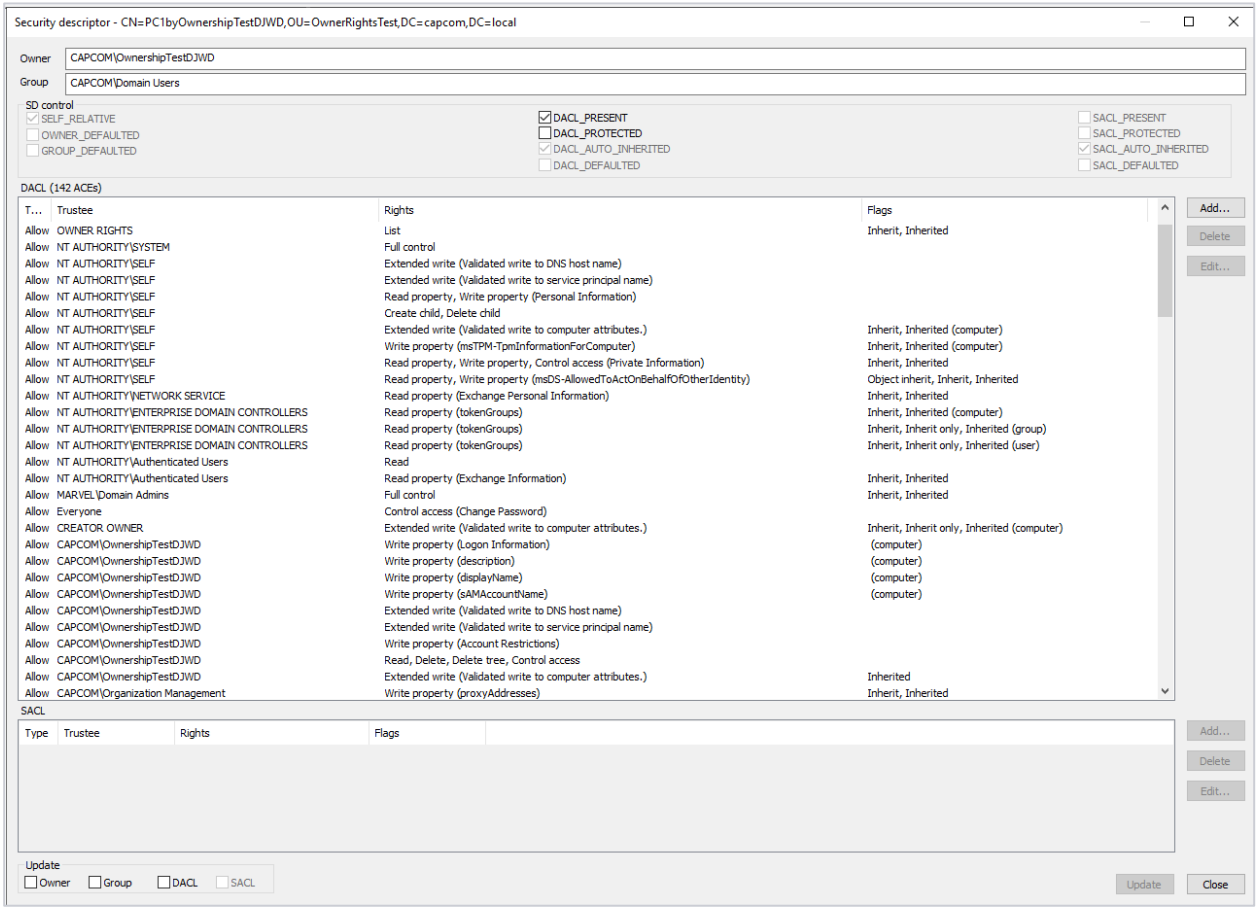
```
<# This snippet collects ACE entries for nonstandard Owners for the newly created test objects. Run this after Create-ObjectOwners.ps1#>
$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Get-ObjectOwnerInfo.txt -append

$ADObjects = @()
$TargetOU = (Get-ADDomain).DistinguishedName
$TargetOU1 = "OU=OwnerRightsTest,"+$TargetOU
$TargetOU2 = "OU=NoOwnerRightsTest,"+$TargetOU
$ACLList = @()
$ADObjects = Get-ADObject -Filter * -SearchBase $TargetOU1 -Properties ntSecurityDescriptor | Select-Object -Property DistinguishedName, Name, @{Name='ntSecurityDescriptorOwner'; Expression={$_.ntSecurityDescriptor.Owner }}
$ADObjects += Get-ADObject -Filter * -SearchBase $TargetOU2  -Properties ntSecurityDescriptor | Select-Object -Property DistinguishedName, Name, @{Name='ntSecurityDescriptorOwner'; Expression={$_.ntSecurityDescriptor.Owner }}
Set-Location AD:
foreach($ADObject in $ADObjects) {
    Write-Host "AD Object: " $ADObject.DistinguishedName
    Write-Host "Owner: " $ADObject.ntSecurityDescriptorOwner
    $ACL = (get-acl $ADObject.DistinguishedName).Access
    $ACLList = $ACL | Where-Object { $_.IdentityReference -eq $ADObject.ntSecurityDescriptorOwner}
    $ACLList
    Write-Host "-------------------------------------------------"
}
Stop-Transcript
```

The results of this script are lengthy so here are a few samples of the data along with GUI screenshots from LDP.exe for that same object:

PC1byOwnershipTestDJWD: (Delegated Domain Join Workstation at Domain Root)

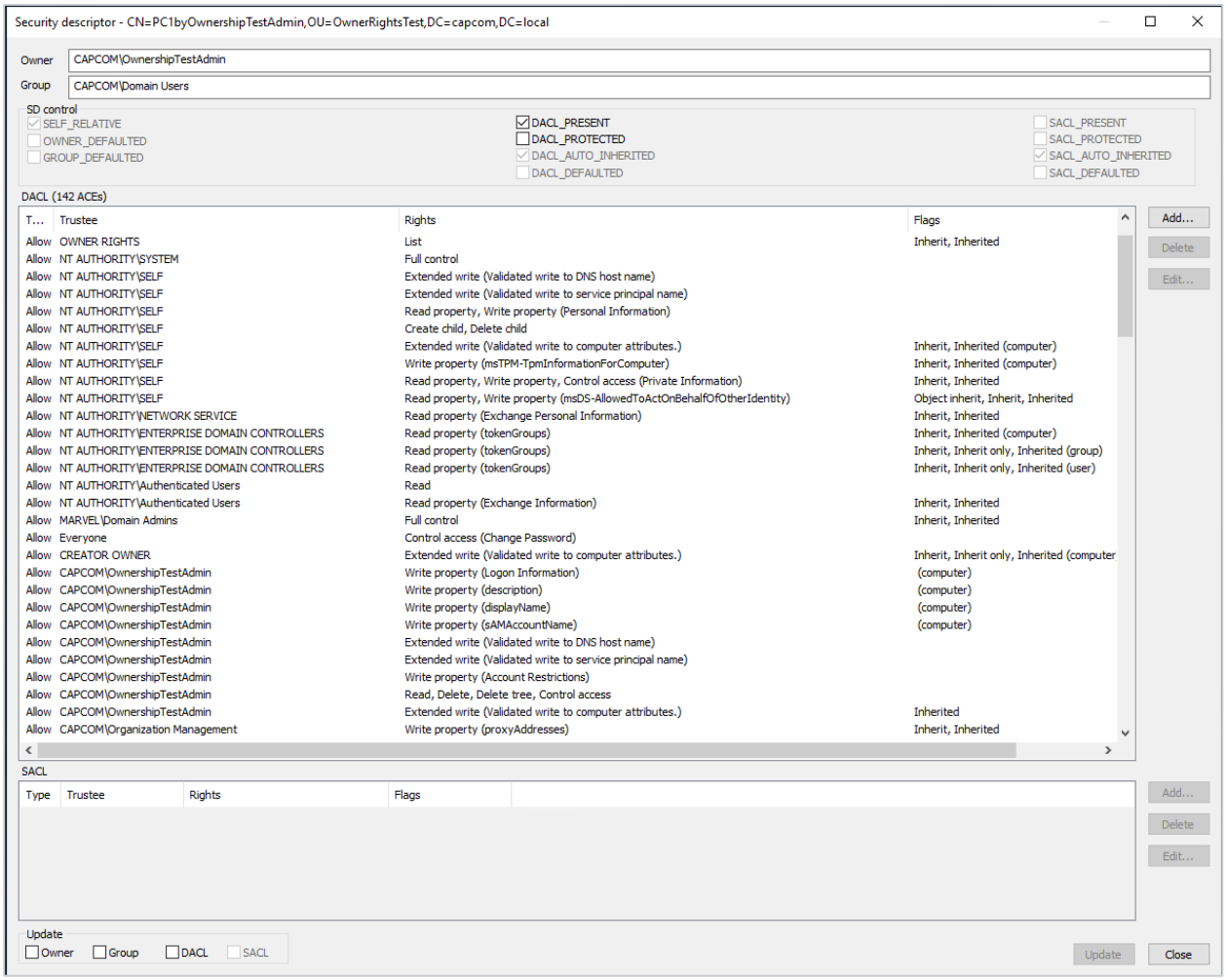
```
1 ------------------------------------------------
2 AD Object:  CN=PC1byOwnershipTestDJWD,OU=OwnerRightsTest,DC=capcom,DC=local
3 Owner:  CAPCOM\OwnershipTestDJWD
4
5 IdentityReference                          ActiveDirectoryRights ObjectType
6 -----------------                          --------------------- ----------
7 CAPCOM\OwnershipTestDJWD DeleteTree, ExtendedRight, Delete, GenericRead 00000000-0000-0000-0000-
8 CAPCOM\OwnershipTestDJWD                                  WriteProperty 4c164200-20c0-11d0-a768-00aa006e0529
9 CAPCOM\OwnershipTestDJWD                                           Self f3a64788-5306-11d1-a9c5-0000f80367c1
10 CAPCOM\OwnershipTestDJWD                                          Self 72e39547-7b18-11d1-adef-00c04fd8d5cd
11 CAPCOM\OwnershipTestDJWD                                 WriteProperty 3e0abfd0-126a-11d0-a060-00aa006c33ed
12 CAPCOM\OwnershipTestDJWD                                 WriteProperty bf967953-0de6-11d0-a285-00aa003049e2
13 CAPCOM\OwnershipTestDJWD                                 WriteProperty bf967950-0de6-11d0-a285-00aa003049e2
14 CAPCOM\OwnershipTestDJWD                                 WriteProperty 5f202010-79a5-11d0-9020-00c04fc2d4cf
15 CAPCOM\OwnershipTestDJWD                                          Self 9b026da6-0d3c-465c-8bee-5199d7165cba
```

Note: On this series of "screenshots" the 00000000-0000.. GUID that wraps the line is All Objects.



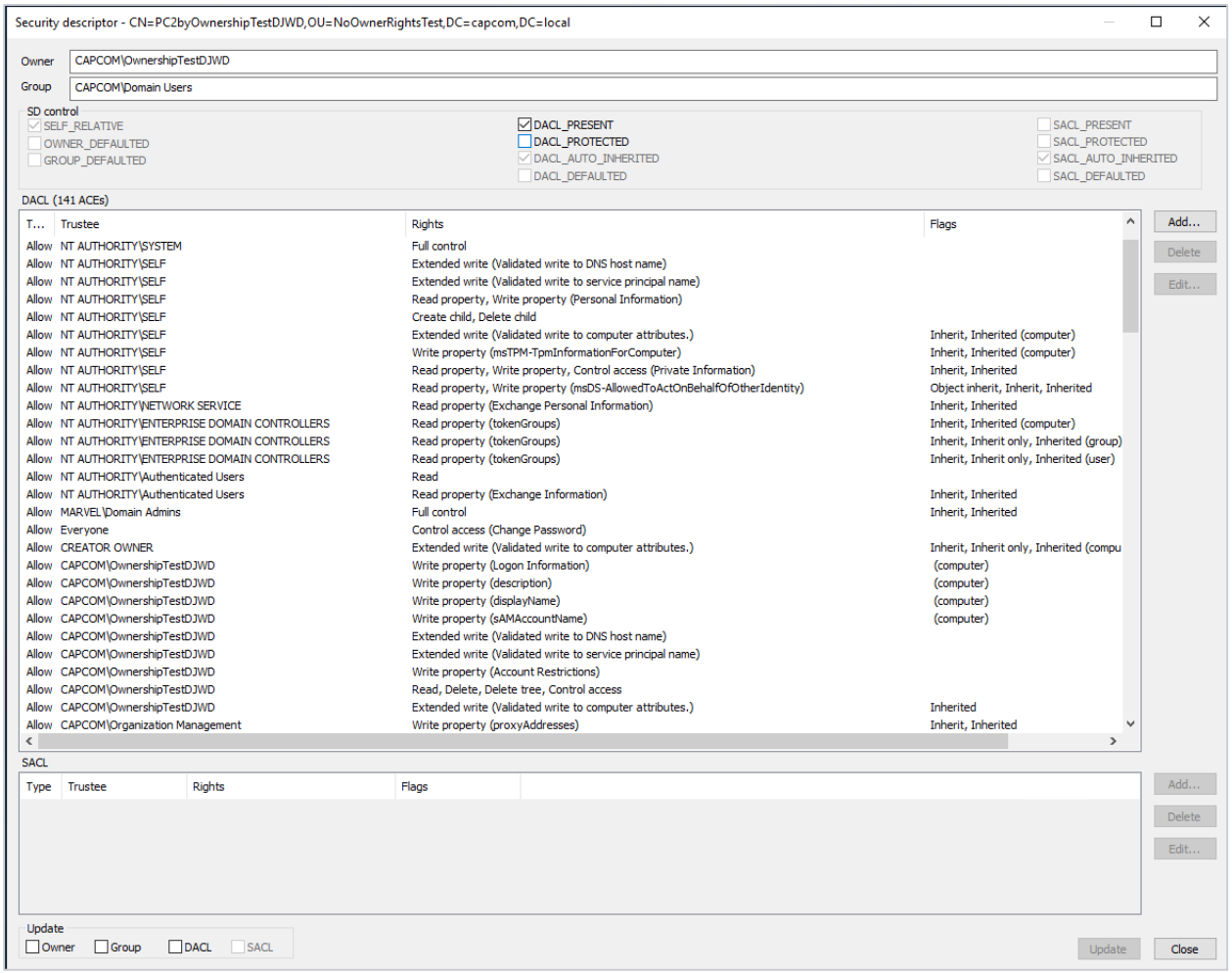
PC1byOwnershipTestAdmin: (Member of BUILTIN\Administrators for the Domain)

```
1 ------------------------------------------------
2 AD Object:  CN=PC1byOwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local
3 Owner:  CAPCOM\OwnershipTestAdmin
4
5 IdentityReference                          ActiveDirectoryRights ObjectType
6 -----------------                          --------------------- ----------
7 CAPCOM\OwnershipTestAdmin DeleteTree, ExtendedRight, Delete, GenericRead 00000000-0000-0000-0000-
8 CAPCOM\OwnershipTestAdmin                                  WriteProperty 4c164200-20c0-11d0-a768-00aa006e0529
9 CAPCOM\OwnershipTestAdmin                                           Self f3a64788-5306-11d1-a9c5-0000f80367c1
10 CAPCOM\OwnershipTestAdmin                                          Self 72e39547-7b18-11d1-adef-00c04fd8d5cd
11 CAPCOM\OwnershipTestAdmin                                  WriteProperty 3e0abfd0-126a-11d0-a060-00aa006c33ed
12 CAPCOM\OwnershipTestAdmin                                  WriteProperty bf967953-0de6-11d0-a285-00aa003049e2
13 CAPCOM\OwnershipTestAdmin                                  WriteProperty bf967950-0de6-11d0-a285-00aa003049e2
14 CAPCOM\OwnershipTestAdmin                                  WriteProperty 5f202010-79a5-11d0-9020-00c04fc2d4cf
15 CAPCOM\OwnershipTestAdmin                                          Self 9b026da6-0d3c-465c-8bee-5199d7165cba
```



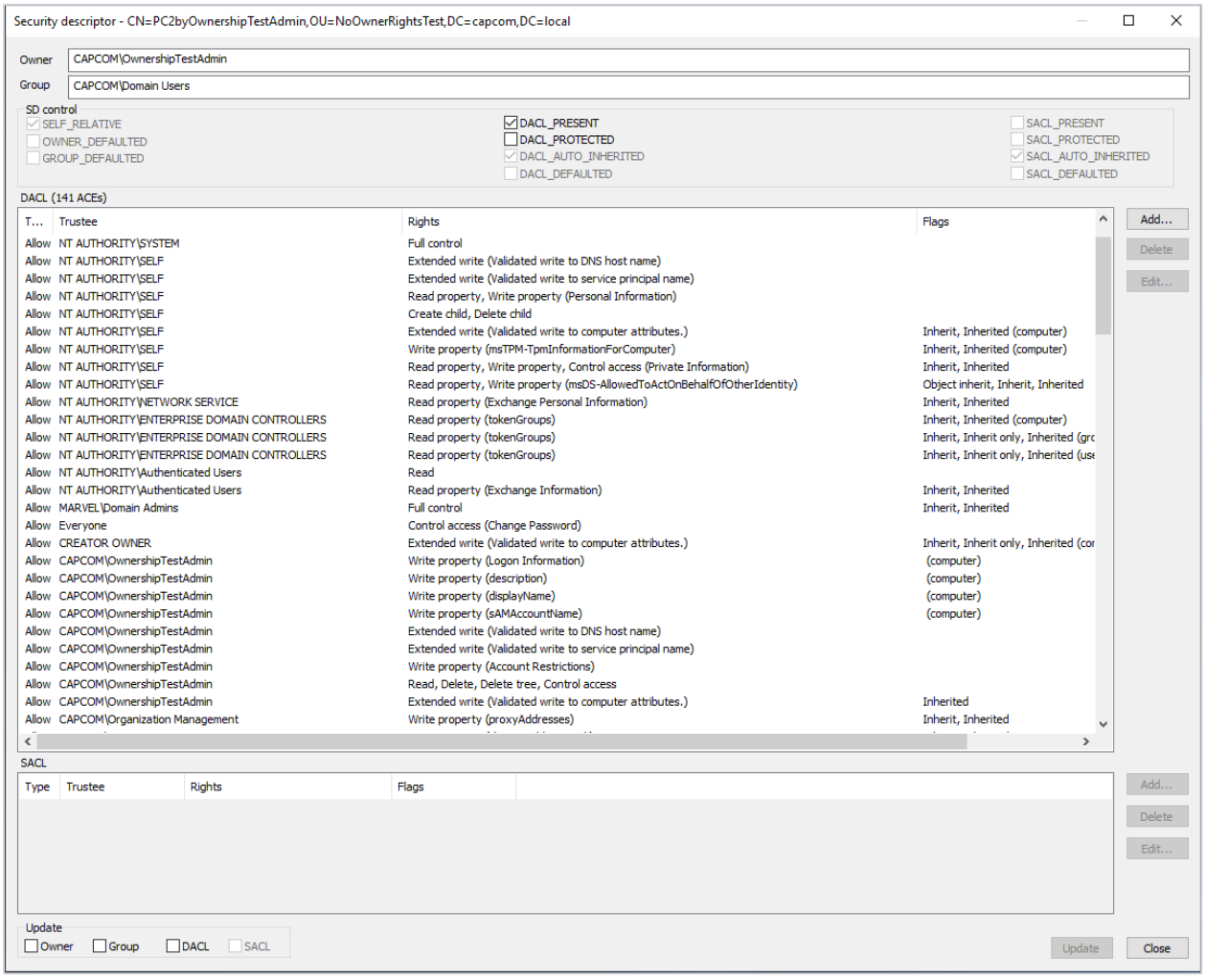
PC2byOwnershipTestDJWD: (Delegated Domain Join Workstation at Domain Root)

```
1 ------------------------------------------------
2 AD Object:  CN=PC2byOwnershipTestDJWD,OU=NoOwnerRightsTest,DC=capcom,DC=local
3 Owner:  CAPCOM\OwnershipTestDJWD
4
5 IdentityReference                          ActiveDirectoryRights ObjectType
6 -----------------                          --------------------- ----------
7 CAPCOM\OwnershipTestDJWD DeleteTree, ExtendedRight, Delete, GenericRead 00000000-0000-0000-0000-
8 CAPCOM\OwnershipTestDJWD                                 WriteProperty 4c164200-20c0-11d0-a768-00aa006e0529
9 CAPCOM\OwnershipTestDJWD                                          Self f3a64788-5306-11d1-a9c5-0000f80367c1
10 CAPCOM\OwnershipTestDJWD                                         Self 72e39547-7b18-11d1-adef-00c04fd8d5cd
11 CAPCOM\OwnershipTestDJWD                                WriteProperty 3e0abfd0-126a-11d0-a060-00aa006c33ed
12 CAPCOM\OwnershipTestDJWD                                WriteProperty bf967953-0de6-11d0-a285-00aa003049e2
13 CAPCOM\OwnershipTestDJWD                                WriteProperty bf967950-0de6-11d0-a285-00aa003049e2
14 CAPCOM\OwnershipTestDJWD                                WriteProperty 5f202010-79a5-11d0-9020-00c04fc2d4cf
15 CAPCOM\OwnershipTestDJWD                                         Self 9b026da6-0d3c-465c-8bee-5199d7165cba
```



PC2byOwnershipTestAdmin: (Member of BUILTIN\Administrators for Domain)

```
1 ------------------------------------------------
2 AD Object:  CN=PC2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local
3 Owner:  CAPCOM\OwnershipTestAdmin
4
5 IdentityReference                          ActiveDirectoryRights ObjectType
6 -----------------                          --------------------- ----------
7 CAPCOM\OwnershipTestAdmin DeleteTree, ExtendedRight, Delete, GenericRead 00000000-0000-0000-0000-
8 CAPCOM\OwnershipTestAdmin                                  WriteProperty 4c164200-20c0-11d0-a768-00aa006e0529
9 CAPCOM\OwnershipTestAdmin                                           Self f3a64788-5306-11d1-a9c5-0000f80367c1
10 CAPCOM\OwnershipTestAdmin                                          Self 72e39547-7b18-11d1-adef-00c04fd8d5cd
11 CAPCOM\OwnershipTestAdmin                                  WriteProperty 3e0abfd0-126a-11d0-a060-00aa006c33ed
12 CAPCOM\OwnershipTestAdmin                                  WriteProperty bf967953-0de6-11d0-a285-00aa003049e2
13 CAPCOM\OwnershipTestAdmin                                  WriteProperty bf967950-0de6-11d0-a285-00aa003049e2
14 CAPCOM\OwnershipTestAdmin                                  WriteProperty 5f202010-79a5-11d0-9020-00c04fc2d4cf
15 CAPCOM\OwnershipTestAdmin                                          Self 9b026da6-0d3c-465c-8bee-5199d7165cba
```



What isn't shown in these screenshots is the default rights granted to the Owner, which are implied.

The only objects with explicit ACEs matching the Object Owner are PCs. This is due to the CREATOR OWNER ACE that is applied to AD Computer Objects at creation by default.

Note: In one of my lab AD Forests with Microsoft Exchange 2016 installed there are 578 different types of object classes defined in the Schema. Of those 578 classes, 267 of them have a defaultSecurityDescriptor defined. Of those 267 classes with a defaultSecurityDescriptor, 38 classes have ACEs that apply to CREATOR OWNER, including the Computer object class.

The defaultSecurityDescriptor for the Computer class can be found here: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adsc/142185a8-2e23-4628-b002-cf31d57bb37a. Parsing out that SDDL a bit and filtering only for those that apply to Creator_Owner(CO):

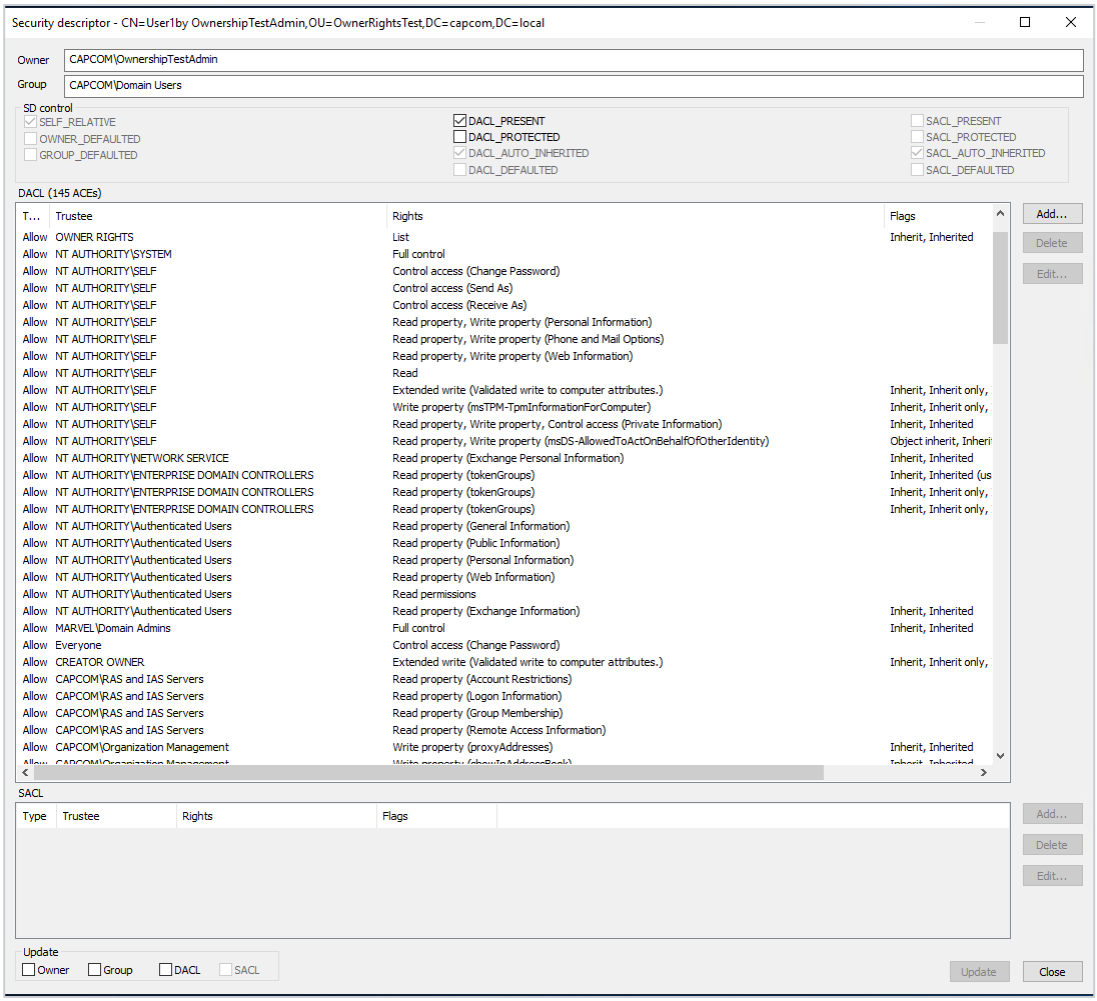
- `(A;;RPCRLCLORCSDDT;;;CO)`
  - `AccessAllowed;;Read All Properties|All Extended Rights|List Contents| List Object|Read Permissions|Delete|Delete Subtree;;Creator_Owner`
- `(OA;;WP;4c164200-20c0-11d0-a768-00aa006e0529;;CO)`
  - `ObjectAccessAllowed;;Write All Properties;User-Acount-Restrictions(PropertySet);;Creator_Owner`
- `(OA;;SW;72e39547-7b18-11d1-adef-00c04fd8d5cd;;CO)`
  - `ObjectAccessAllowed;;All Validated Writes; DNS-Host-Name-Attributes(PropertySet);;Creator_Owner`
- `(OA;;SW;f3a64788-5306-11d1-a9c5-0000f80367c1;;CO)`
  - `ObjectAccessAllowed;;All Validated Writes;Validated-SPN;;Creator_Owner`
- `(OA;;WP;3e0abfd0-126a-11d0-a060-00aa006c33ed;bf967a86-0de6-11d0-a285-00aa003049e2;CO)`
  - `ObjectAccessAllowed;;Write All Properties;SAM-Account-Name;Computer;Creator_Owner`
- `(OA;;WP;5f202010-79a5-11d0-9020-00c04fc2d4cf;bf967a86-0de6-11d0-a285-00aa003049e2;CO)`
  - `ObjectAccessAllowed;;Write All Properties;User-Logon(PropertySet);Computer;Creator_Owner`
- `(OA;;WP;bf967950-0de6-11d0-a285-00aa003049e2;bf967a86-0de6-11d0-a285-00aa003049e2;CO)`
  - `ObjectAccessAllowed;;Write All Properties;Description;Computer;Creator_Owner`
- `(OA;;WP;bf967953-0de6-11d0-a285-00aa003049e2;bf967a86-0de6-11d0-a285-00aa003049e2;CO)`
  - `ObjectAccessAllowed;;Write All Properties;Display-Name;Computer;Creator_Owner`

Parsing that out further, the specific rights granted to the CREATOR OWNER on Computer Objects are:

- Read All Properties
- All Extended Rights *(Those that apply to Computers Objects)*
  - Allowed-To-Authenticate *(Note: Another seemingly forgotten Active Directory powerhouse)*
  - Enable-Per-User-Reversibly-Encrypted-Password
  - Migrate-SID-History
  - Receive-As
  - Send-As
  - User-Change-Password
  - User-Force-Change-Password
- List Contents
- List Object
- Read Permissions
- Delete
- Delete Subtree
- Write Property (User Account Restrictions property set)
  - Account-Expires

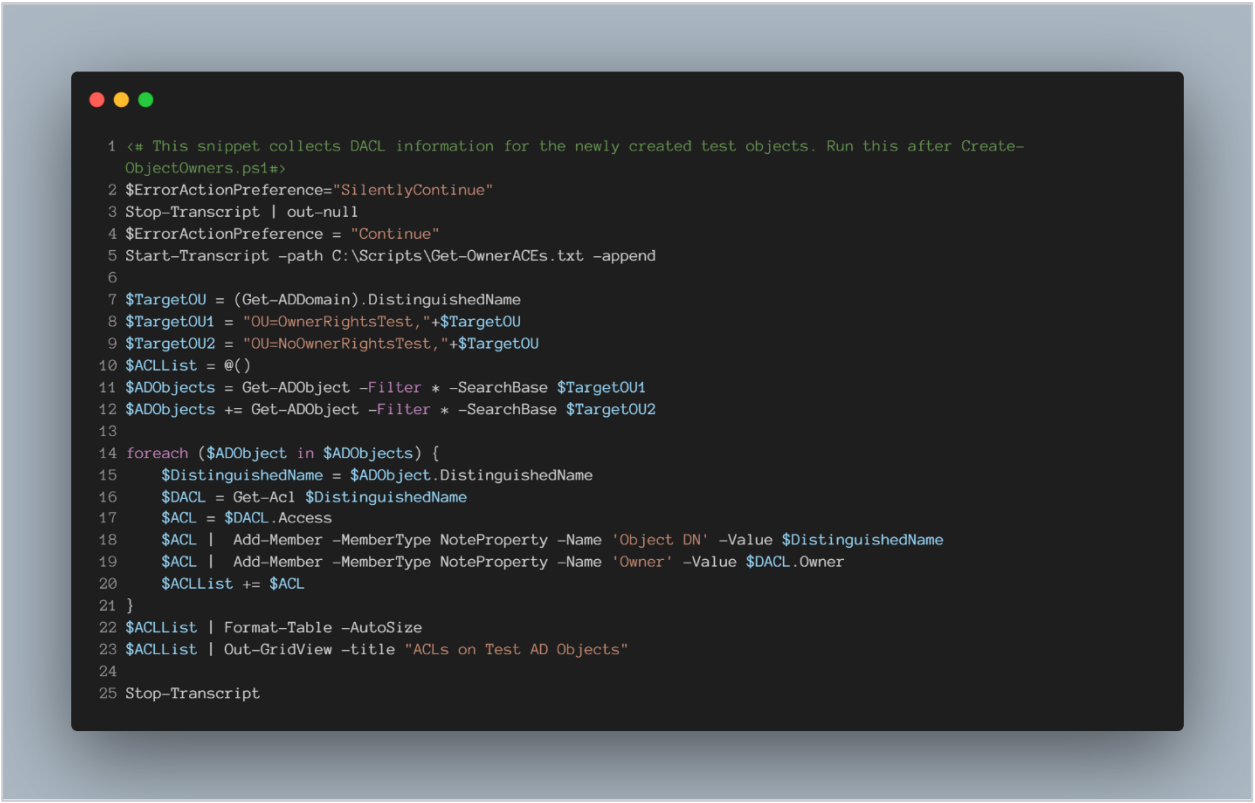
    - ms-DS-User-Account-Control-Computed
    - ms-DS-User-Password-Expiry-Time-Computed
    - Pwd-Last-Set
    - User-Account-Control
    - User-Parameters
- Validated Write (DNS Host Name property set)
    - DNS-Host-Name
    - Ms-DS-Additional-Dns-Host-Name
- Validated Write (Validated-SPN)
- Write Property (SAM-Account-Name)
- Write property (User Logon property set)
    - Bad-Pwd-Count
    - Home-Directory
    - Home-Drive
    - Last-Logoff
    - Last-Logon
    - Logon-Hours
    - Logon-Workstation
    - Profile-Path
- Write Property (Description)
- Write Property (Display-Name)

For example, “User1by OwnershipTestAdmin” in the OwnerRightsTest OU doesn’t have any explicit ACEs for the Owner:



The other main difference between the Computer Objects that are created is that those in the OwnerRightsTest OU have the OWNER RIGHTS List contents permission applied as an inherited permission, whereas those in the NoOwnerRightsTest OU do not.

I've also ran another PowerShell snippet (Get-OwnerACEs.ps1) that collects a comprehensive permissions set for all the objects in the two test OUs.



```powershell
<# This snippet collects DACL information for the newly created test objects. Run this after Create-ObjectOwners.ps1#>
$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Get-OwnerACEs.txt -append

$TargetOU = (Get-ADDomain).DistinguishedName
$TargetOU1 = "OU=OwnerRightsTest,"+$TargetOU
$TargetOU2 = "OU=NoOwnerRightsTest,"+$TargetOU
$ACLList = @()
$ADObjects = Get-ADObject -Filter * -SearchBase $TargetOU1
$ADObjects += Get-ADObject -Filter * -SearchBase $TargetOU2

foreach ($ADObject in $ADObjects) {
    $DistinguishedName = $ADObject.DistinguishedName
    $DACL = Get-Acl $DistinguishedName
    $ACL = $DACL.Access
    $ACL |  Add-Member -MemberType NoteProperty -Name 'Object DN' -Value $DistinguishedName
    $ACL |  Add-Member -MemberType NoteProperty -Name 'Owner' -Value $DACL.Owner
    $ACLList += $ACL
}
$ACLList | Format-Table -AutoSize
$ACLList | Out-GridView -title "ACLs on Test AD Objects"

Stop-Transcript
```

This results in a PowerShell GridView table, but I've also saved it as a CSV (converted to Excel here) so we can compare it with the DACLs after we attempt abuse.

8. Revoke Privileged Access

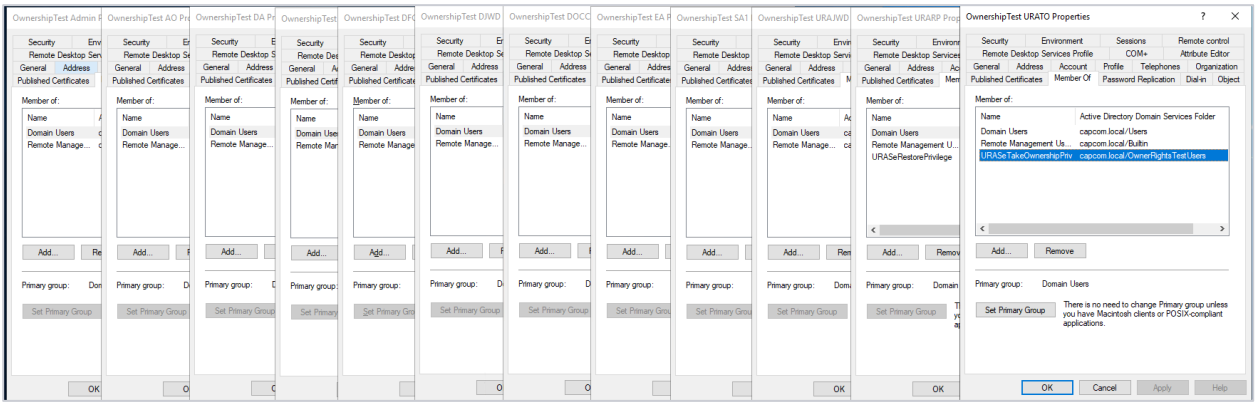
Now we'll remove all the users from their respective security groups to remove any delegated permissions from the users so that they rely only on any latent rights from being an AD Object Owner, and for Computer Objects the rights they were assigned as CREATOR OWNER.

This is to simulate the concept of a user having once been a privileged user, but having that access revoked.

Note: PowerShell snippet available here: Remove-TestUserGroups.ps1

```powershell
<# This snippet removes the test users from their groups, thus revoking their privileges.
This allows the snippet in Abuse-Ownership.ps1 to be run so that any actions performed are done only as
permissions that remain as an Owner.
Run after collecting a baseline with Get-Owners.ps1, Get-OwnerACEs.ps1, and Get-ObjectOwnerInfo.ps1, but before
Abuse-Ownership.ps1#>

$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Remove-TestUserGroups.txt -append

# Remove Test Users from Groups for AD Object Ownership Testing
Remove-ADGroupMember -Identity "Domain Admins" -Members OwnershipTestDA -Confirm:$False
Remove-ADGroupMember -Identity "Enterprise Admins" -Members OwnershipTestEA -Confirm:$False
Remove-ADGroupMember -Identity "Administrators" -Members OwnershipTestAdmin -Confirm:$False
Remove-ADGroupMember -Identity "Account Operators" -Members OwnershipTestAO  -Confirm:$False
Remove-ADGroupMember -Identity "DelegatedFullControlDomain" -Members OwnershipTestDFCD -Confirm:$False
Remove-ADGroupMember -Identity "DelegatedFullControlOU" -Members OwnershipTestDFCO -Confirm:$False
Remove-ADGroupMember -Identity "DelegatedJoinWorkstationDomain" -Members OwnershipTestDJWD -Confirm:$False
Remove-ADGroupMember -Identity "DelegatedOUCreateComputer" -Members OwnershipTestDOCC -Confirm:$False
Remove-ADGroupMember -Identity "URAJoinWorkstationDomain" -Members OwnershipTestURAJWD -Confirm:$False
Remove-ADGroupMember -Identity "Server Operators" -Members OwnershipTestSA1 -Confirm:$False

Stop-Transcript
```

Here we can see that all but two of the groups have lost their delegated permissions. All of these users only have privileges consistent with being members of Domain Users, Remote Management Users, having been the CREATOR OWNER, or the implicit rights from being an Object Owner.



We'll leave OwnershipTestURARP and OwnershipTestURATO with their permissions for now to do some different tests later.

9. Attempt Abuse

Now we'll run a script that will identify all the AD Objects in the domain that have non-standard ownership and then for each of those objects, utilize that Object Owner to modify the DACL on the object which would allow the "Attacker Controlled" user Full Control over the object. In this way, we'll know that any objects non-standard ownership which end up with an ACE for "Attacker Controlled" resulted in abuse of the Object Ownership.

Note: PowerShell snippet available here: Abuse-Ownership.ps1

```powershell
<#This snippet will attempt to use the remaining permissions that AD Object Owners may have to create a dangerous ACE for an "attacker" controlled account.
Run this after Remove-TestUserGroups.ps1 and ensuring all permissions are revoked on the test users.
Then run the Get-ObjectOwnerInfo.ps1 and Get-OwnerACEs.ps1 again to compare and determine where the attempt to abuse ownership privileges was successful.#>

$Password = Read-Host "Enter a password for test users:"  -AsSecureString

$ErrorActionPreference="SilentlyContinue"
Stop-Transcript | out-null
$ErrorActionPreference = "Continue"
Start-Transcript -path C:\Scripts\Abuse-OwnershipNew.txt -append

$Domain = Get-ADDomain
$DomainNETBIOS = $Domain.NetBIOSName
$TargetPC = 'localhost'
$AttackerName = "AttackerControlled"
[string]$DN
[string]$DelegatedUser

$ADObjects = Get-ADObject -Filter * -properties ntSecurityDescriptor | Select-Object -Property Name, @{Name='ntSecurityDescriptorOwner'; Expression={$_.ntSecurityDescriptor.Owner }}, DistinguishedName | Where-Object { $_.ntSecurityDescriptorOwner -notlike "$DomainNETBIOS\Domain Admins" -and $_.ntSecurityDescriptorOwner -notlike "$DomainNETBIOS\Enterprise Admins" -and $_.ntSecurityDescriptorOwner -notlike "NT AUTHORITY\SYSTEM" -and $_.ntSecurityDescriptorOwner -notlike "BUILTIN\Administrators" -and $_.ntSecurityDescriptorOwner -notlike "$DomainNETBIOS\*`$" }

Set-Location AD:
foreach($ADObject in $ADObjects) {
    Write-Host "AD Object: " $ADObject.DistinguishedName
    Write-Host "Owner: " $ADObject.ntSecurityDescriptorOwner
    $DelegatedUser = $ADObject.ntSecurityDescriptorOwner.Split("\")[1]
    $Credential = New-Object -TypeName "System.Management.Automation.PSCredential" -ArgumentList $DelegatedUser,$Password
    $DN = $AdObject.DistinguishedName
    # Attempt to add Full Control ACE for an Attacker Controlled Account to each object that has a non-standard Owner.
    $output =  Invoke-Command -ComputerName $TargetPC  -Credential $Credential -ArgumentList $AttackerName, $DelegatedUser, $DN, $ACL -ScriptBlock {param($AttackerName, $DelegatedUser, $DN, $ACL )
    whoami.exe /All
    $ADSI = [ADSI]"LDAP://$DN"
    $IdentityReference = [System.Security.Principal.IdentityReference] ([System.Security.Principal.SecurityIdentifier](Get-ADUser $AttackerName).SID)
    $InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] 'None'
    $ControlType = [System.Security.AccessControl.AccessControlType] 'Allow'
    $ADRights = [System.DirectoryServices.ActiveDirectoryRights] 'GenericAll'
    $ACE = New-Object System.DirectoryServices.ActiveDirectoryAccessRule $IdentityReference, $ADRights, $ControlType, $InheritanceType
    $ADSI.PsBase.Options.SecurityMasks = 'Dacl'
    $ADSI.PsBase.ObjectSecurity.SetAccessRule($ACE)
    $ADSI.PsBase.CommitChanges()
    }
    Write-Host "-------------------------------------------------"
}

Stop-Transcript
```

Here the script output is lengthy again, but this is the interesting part when it comes to utilizing OWNER RIGHTS as a proactive defense:

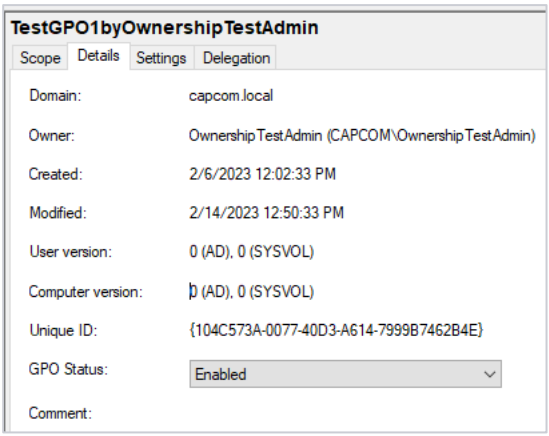
```
AD Object:  OU=OUbyOwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
```

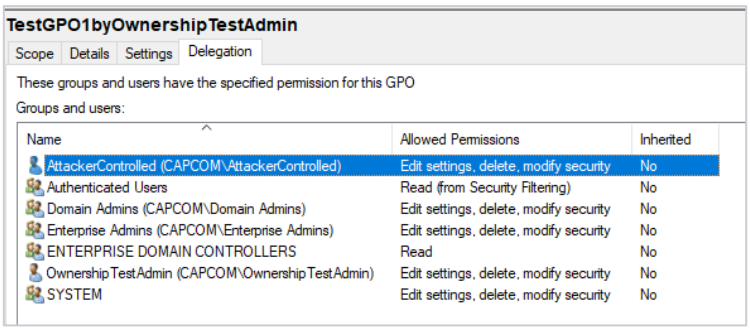
```
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=User1by OwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=PC1byOwnershipTestAdmin,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN={104C573A-0077-40D3-A614-
7999B7462B4E},CN=Policies,CN=System,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=Machine,CN={104C573A-0077-40D3-A614-
7999B7462B4E},CN=Policies,CN=System,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=User,CN={104C573A-0077-40D3-A614-
7999B7462B4E},CN=Policies,CN=System,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  OU=OUNobyOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=User2by OwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=PC2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN={482EDC52-2697-4851-8005-
CA4D65DF19E2},CN=Policies,CN=System,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=Machine,CN={482EDC52-2697-4851-8005-
CA4D65DF19E2},CN=Policies,CN=System,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=User,CN={482EDC52-2697-4851-8005-
CA4D65DF19E2},CN=Policies,CN=System,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAdmin
-------------------------------------------------
AD Object:  CN=User1by OwnershipTestAO,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAO
```

```
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=PC1byOwnershipTestAO,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAO
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=User2by OwnershipTestAO,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAO
-------------------------------------------------
AD Object:  CN=PC2byOwnershipTestAO,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestAO
-------------------------------------------------
AD Object:  OU=OUbyOwnershipTestDFCD,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCD
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=User1by OwnershipTestDFCD,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCD
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=PC1byOwnershipTestDFCD,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCD
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  OU=OUNobyOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCD
-------------------------------------------------
AD Object:  CN=User2by OwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCD
-------------------------------------------------
AD Object:  CN=PC2byOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCD
-------------------------------------------------
AD Object:  OU=OUbyOwnershipTestDFCO,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCO
```

```
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=User1by OwnershipTestDFCO,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCO
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=PC1byOwnershipTestDFCO,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCO
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  OU=OUNobyOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCO
-------------------------------------------------
AD Object:  CN=User2by OwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCO
-------------------------------------------------
AD Object:  CN=PC2byOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDFCO
-------------------------------------------------
AD Object:  CN=PC1byOwnershipTestDJWD,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDJWD
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=PC2byOwnershipTestDJWD,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDJWD
-------------------------------------------------
AD Object:  CN=PC1byOwnershipTestDOCC,OU=OwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDOCC
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : capcom-19

-------------------------------------------------
AD Object:  CN=PC2byOwnershipTestDOCC,OU=NoOwnerRightsTest,DC=capcom,DC=local
Owner:  CAPCOM\OwnershipTestDOCC
-------------------------------------------------
```

Look at all these “Access is denied” errors! At first it didn’t look like anything happened at all, but then quite a few of the changes were successfully committed. When we correlate between the commands that errored out, we see they’re all for objects in the OwnerRightsTest OU, where we had assigned the OWNER RIGHTS principal only List Contents permissions. The abuse of Object Owner implicit WRITE_DAC permissions only worked against objects in OU=NoOwnerRightsTest,DC=capcom,DC=local and CN=Policies,CN=System,DC=capcom,DC=local. The GPOs linked to both OUs were abusable because the permissions on the Policies container are what matter, not the links to the OUs.





When I run the script to collect all permissions in the two test OUs and filter it on OUs that have an ACE for CAPCOM\AttackerControlled we find that only the Objects in the NoOwnershipTest were abused.

Object DN	Object Owner	IdentityReference	AD Rights
OU=OUNobyOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDFCO	CAPCOM\AttackerControlled	Generic All
OU=OUNobyOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDFCD	CAPCOM\AttackerControlled	Generic All
OU=OUNobyOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestAdmin	CAPCOM\AttackerControlled	Generic All
CN=User2by OwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDFCO	CAPCOM\AttackerControlled	Generic All
CN=User2by OwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDFCD	CAPCOM\AttackerControlled	Generic All
CN=User2by OwnershipTestAO,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestAO	CAPCOM\AttackerControlled	Generic All
CN=User2by OwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestAdmin	CAPCOM\AttackerControlled	Generic All

CN=PC2byOwnershipTestDOCC,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDOCC	CAPCOM\AttackerControlled	GenericAll
CN=PC2byOwnershipTestDJWD,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDJWD	CAPCOM\AttackerControlled	GenericAll
CN=PC2byOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDFCO	CAPCOM\AttackerControlled	GenericAll
CN=PC2byOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestDFCD	CAPCOM\AttackerControlled	GenericAll
CN=PC2byOwnershipTestAO,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestAO	CAPCOM\AttackerControlled	GenericAll
CN=PC2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=capcom,DC=local	CAPCOM\OwnershipTestAdmin	CAPCOM\AttackerControlled	GenericAll

Note: There's a spreadsheet with post-abuse ACEs on GitHub, if you're following along at home.

Defensive Recommendation

Note: Test this in a lab environment first to make sure the deployment has the desired results before deploying to any production environment.

Apply the OWNER RIGHTS well-known security principal to the DACL of any high value OUs with an ACE that grants minimal inheritable permissions. This can be done through the GUI, but it's fairly easy to mess up when attempting to apply very limited permissions through the GUI. Instead, I'd recommend using PowerShell:

```
# Assign inheritable ListChildren permissions to OWNER RIGHTS for a specific DistinguishedName
$DN = <DN of Target OU>

Set-Location AD:
# Need to manually add the SID here due to Owner Rights being a well-known identity.
$GroupSID =  New-Object System.Security.Principal.SecurityIdentifier("S-1-3-4")
$ACL = Get-Acl -Path $DN

$Identity = [System.Security.Principal.IdentityReference] $GroupSID
$ADRight = [System.DirectoryServices.ActiveDirectoryRights] "ListChildren"
$Type = [System.Security.AccessControl.AccessControlType] "Allow"
$InheritanceType = [System.DirectoryServices.ActiveDirectorySecurityInheritance] "All"
$Rule = New-Object System.DirectoryServices.ActiveDirectoryAccessRule($Identity, $ADRight, $Type,
$InheritanceType)

$ACL.AddAccessRule($Rule)
Set-Acl -Path $DN -AclObject $ACL
```

It's tempting to assign this permission at the Domain Root, but frankly I haven't tested that scenario and I'm unsure of areas of potential downfall in that method. It may cause issues with DNS nodes that are stored in a Legacy (Windows 2000) zone, but you really shouldn't be using Legacy DNS zones anymore. Targeted deployment in a well-designed and tiered OU structure would be best.

Configuring OWNER RIGHTS with limited permissions on the Domain Controllers OU would help mitigate some RBCD attacks against DCs with non-standard ownership.

Note: This should be combined with considering locking down the DACL on the Domain Controllers OU and disabling inheritance regardless as individual Domain Controller computer objects are not protected by AdminSDHolder.

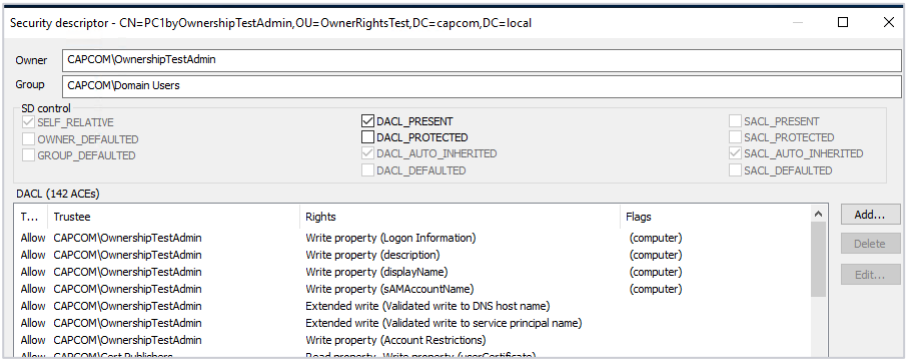
There are some other possible defensive uses of the OwnerRights (S-1-3-4) SID to be found here:

https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc749445(v=ws.10)?redirectedfrom=MSDN

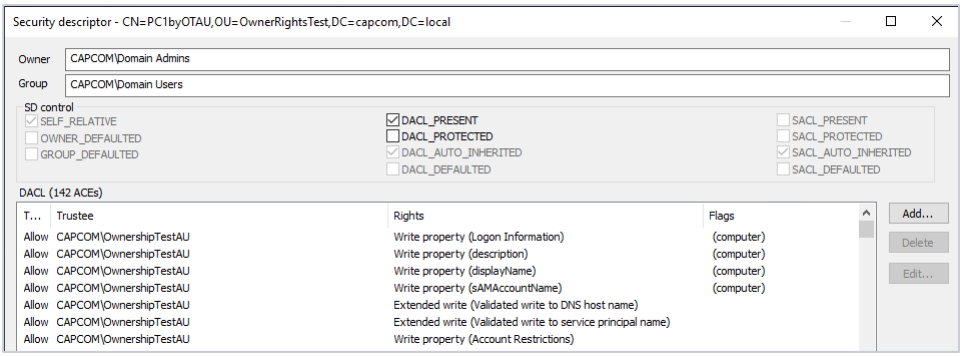
- OwnerRights can be used in an NTFS file system to prevent users from changing settings on files or folders they've created.
- Services can be assigned a SID in current Windows versions, allowing resource isolation.
- Prevent token size increase.

Applying a restrictive OWNER RIGHTS ACE to the default Computers container and any OUs where lower-privileged users are delegated permissions to create computers or users seems like a quick win.

But even without the ability to abuse the Owner's Write_DAC, the computer objects we created still have explicit ACEs that came out of CREATOR OWNER:



Even the computer object that was created by a standard user with nothing more than Domain User rights (allowing for the default Add workstations to the domain right) receives CREATOR OWNER ACEs:



This leads us to the next section:

dSHeuristics & CVE-2021-42291

LDAP Add & Modify

Enter CVE-2021-42291, which is an Active Directory Domain Services Elevation of Privilege Vulnerability.

This patch (and additional remediation steps in KB5008383) for this vulnerability have to do with changes to permissions checks that occur during LDAP Add and Modify operations. Once the patches from November 2021 Patch Tuesday (November 9, 2021) are installed, there's an additional audit-by-default functionality that occurs when someone sets potentially suspicious permissions on a computer or computer-derived object and an upcoming enforcement mode that blocks such attempts. Enforcement mode ~~will have happened automatically when April 11, 2023 patches are installed,~~ should happen in January 2024, or it could be done manually with dSHeuristics for organizations that desire the strongest security posture as soon as possible.

Note: I was surprised by a last-minute extension of the final deployment phase for this issue. At the original time of publishing, the final deployment phase appeared to have already been a done deal. I removed some language in this section relating to how to modify dSHeuristics to enable enforcement mode as it seemed no longer relevant. I now wish I had left it in regardless, so I'm adding it back below. Eventually, it will no longer be necessary to modify dSHeuristics for this feature after April 2023 January 2024 patches are installed.

Domain Controllers will ignore these settings once the final deployment phase is completed by patches. I suggest monitoring KB5008383 in case the final deployment timeline is changed again.

The audit-by-default logging introduces EventIDs 3044-3056 in the Directory Service log on Domain Controllers, with the patch installed. These logs indicate when a user might have excessive privileges to create computer accounts with arbitrary security-sensitive attributes.

LDAP Add operations for computer objects have additional authorization checks that prevent users without administrative rights in the domain from setting the securityDescriptor or other attributes to values that might grant excessive permissions on computer-derived AD objects.

LDAP Modify operations for computer objects have an additional function to temporarily remove implicit owner rights to prevent users without administrative rights in the domain from setting the securityDescriptor to values that might grant excessive permissions on existing computer-derived AD objects. Basically, this check makes sure the user would have rights to Write_DAC without the implicit owner privileges.

The details for this change can be found in the MS-ADTS Active Directory Technical Specifications, specifically sections:

- 3.1.3.3.4.1 LDAP Extended Controls
- 3.1.3.4.1.11 LDAP_SERVER_SD_FLAGS_OID
- 3.1.1.5.2.1 Security Considerations
- 3.1.1.5.2.1.1 Per Attribute Authorization for Add Operation
- 5.1.3.3.1 Null vs Empty DACLs
- 6.1.1.2.4.1.2 dsHeuristics
- 6.1.3.5 Security Considerations.

Note: Computer-derived AD objects includes (g)MSA accounts (ms-DS-Managed-Service-Account, ms-DS-Group-Managed-Service-Account, and the new for Windows Server 2025 ms-DS-Delegated-Managed-Service-Account) classes in the AD Schema. These are Schema classes with a subClassOf 'computer' in my AD labs. ms-Exch-Computer-Policy is another computer-derived AD object that's present when the AD Schema is extended for Microsoft Exchange Server. I was going to validate this further but ran out of time and my lab went "poof".

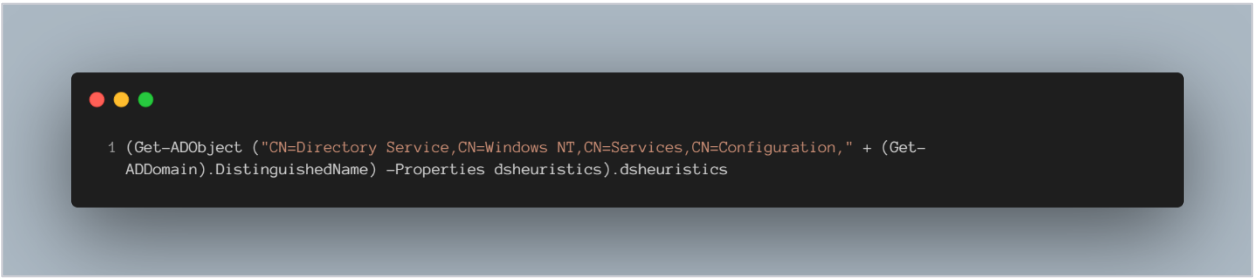
~~By the time this blog is published, enforcement mode will be enabled (April 11, 2023 security updates), at which point the audit mode (default) and disabled mode will be unavailable. As long as April 2023 patches are installed on AD DS Domain Controllers, it will not be necessary to modify the environment to gain the benefits of enforcement mode.~~

dSHeuristics: The Attribute

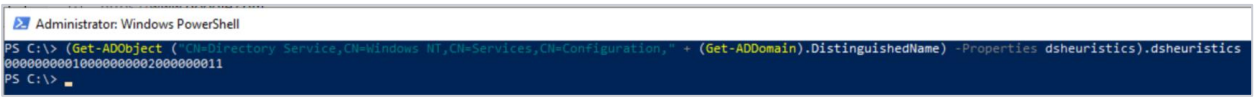
Traditionally, Trimarc hasn't recommended making changes to the dSHeuristics attribute for the domain because it can be tricky to configure and it's very possible to lessen the security of AD through this functionality if care is not taken.

dSHeuristics is a Unicode string attribute where each character in the string represents a heuristic that determines behavior in Active Directory. By default, the dSHeuristics value doesn't exist and unless otherwise specified, the default value of each character is '0'. The order of the characters in the string is fixed. When modifying an existing dSHeuristics string, the values of all existing characters that are not related to the change need to be left alone. By default, the dSHeuristics attribute will be blank.

dSHeuristics is located in CN=Directory Service,CN=Windows NT,CN=Services,CN=Configuration,DC=CONTOSO,DC=COM, so this isn't an attribute you're going to find in the standard Active Directory Users and Computers (ADUC) GUI. Looking at dSHeuristics from a GUI involves the somewhat scary territory of ADSIEdit or LDP.exe. But luckily, PowerShell does a great job of interacting with it also.

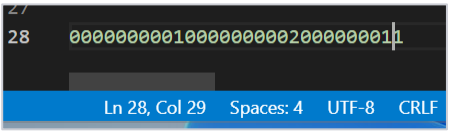


By default, any authenticated user can query this value. But it takes an AD Admin to set it. When both the LDAP Add and Modify bits are set, per the KB5008383 documentation, it'll look like this (assuming all the other bits are default):



"0000000001000000002000000011"

It's helpful to copy the dSHeuristics string value into a text editor that displays columns or character counts. Something like VSCode:

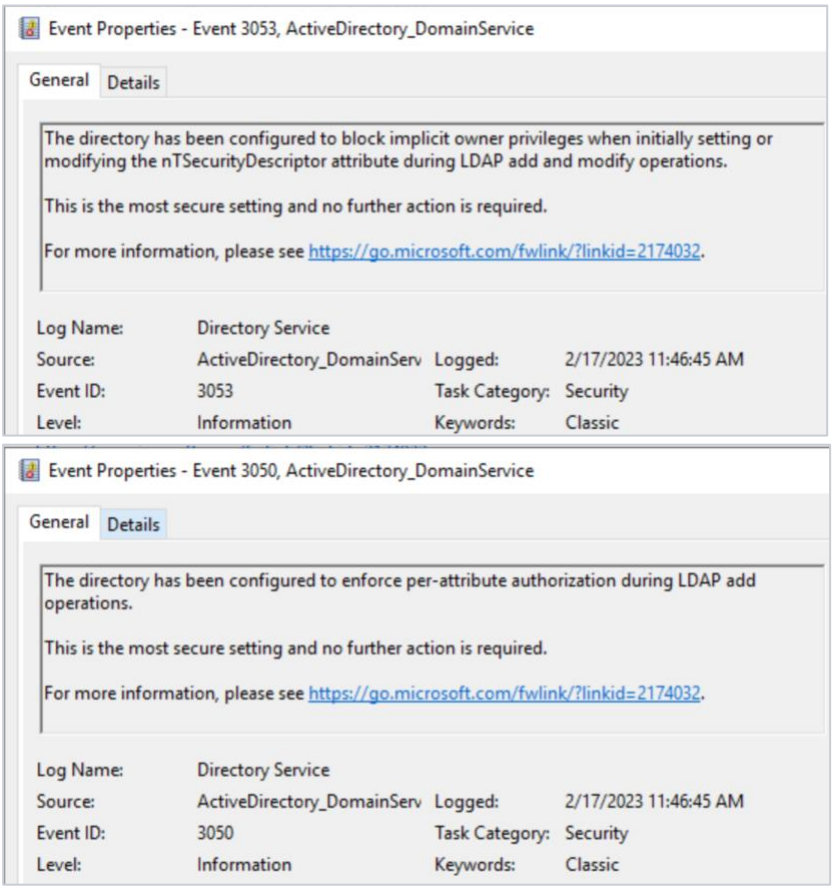


This way you can be certain that the correct values are in the correct columns. This is how I validate when building out a dSHeuristics value to set in a domain as well.

Testing, Testing

In another lab domain I replicated the same experiment as I had for Owner Rights, but this time I first set the dSHeuristics attribute to have 1s in positions 28 and 29, consistent with the KB5008383 documentation. I did this with the PowerShell snippets here in my GitHub. This is consistent with

enforcement mode in the KB article. I rebooted all the DCs in the domain to ensure that the change had taken effect:



Then I created all the OUs, groups, users, and delegations that are the foundation of the test before using the Create-ObjectOwners.ps1 script and some Powermad functions to recreate all the test objects. Then I collected the ownership information and all the DACLs on the test objects before removing group membership from the test users and attempting to Abuse-Ownership.

Path	Owner
OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
OU=OUbyOwnershipTestDFCO,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCO**
CN=User1by OwnershipTestDFCO,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCO**
CN=PC1byOwnershipTestDFCO,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCO**
CN=PC1byOwnershipTestDJWD,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDJWD**
CN=PC1byOwnershipTestDOCC,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDOCC**
CN=PC1byOTURAJWD,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=PC1byOTAU,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=PC1byOTSA1,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
OU=OUbyOwnershipTestEA,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Enterprise Admins
CN=User1by OwnershipTestEA,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Enterprise Admins
CN=PC1byOwnershipTestEA,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Enterprise Admins

OU=OUbyOwnershipTestDA,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=User1by OwnershipTestDA,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=PC1byOwnershipTestDA,OU=OwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
OU=OUbyOwnershipTestAdmin,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAdmin**
CN=User1by OwnershipTestAdmin,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAdmin**
CN=PC1byOwnershipTestAdmin,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAdmin**
CN=User1by OwnershipTestAO,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAO**
CN=PC1byOwnershipTestAO,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAO**
OU=OUbyOwnershipTestDFCD,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCD**
CN=User1by OwnershipTestDFCD,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCD**
CN=PC1byOwnershipTestDFCD,OU=OwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCD**
OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
OU=OUNobyOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCO**
CN=User2by OwnershipTestDFCO,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCO**
CN=PC2byOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCO**
CN=PC2byOwnershipTestDJWD,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDJWD**
CN=PC2byOwnershipTestDOCC,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDOCC**
CN=PC2byOTURAJWD,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=PC2byOTAU,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=PC2byOTSA1,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
OU=OUNobyOwnershipTestEA,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Enterprise Admins
CN=User2by OwnershipTestEA,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Enterprise Admins
CN=PC2byOwnershipTestEA,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Enterprise Admins
OU=OUNobyOwnershipTestDA,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=User2by OwnershipTestDA,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
CN=PC2byOwnershipTestDA,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\Domain Admins
OU=OUNobyOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAdmin**
CN=User2by OwnershipTestAdmin,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAdmin**
CN=PC2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAdmin**
CN=User2by OwnershipTestAO,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAO**
CN=PC2byOwnershipTestAO,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestAO**
OU=OUNobyOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCD**
CN=User2by OwnershipTestDFCD,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCD**
CN=PC2byOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=marvel,DC=local	**MARVEL\OwnershipTestDFCD**

The created objects look about the same as the Owner Rights test above, which is a good sign.

It's when we attempt to abuse object ownership that we find a difference. None of the computer objects were abused, not even the ones in an OU without an Owner Rights ACE assigned.

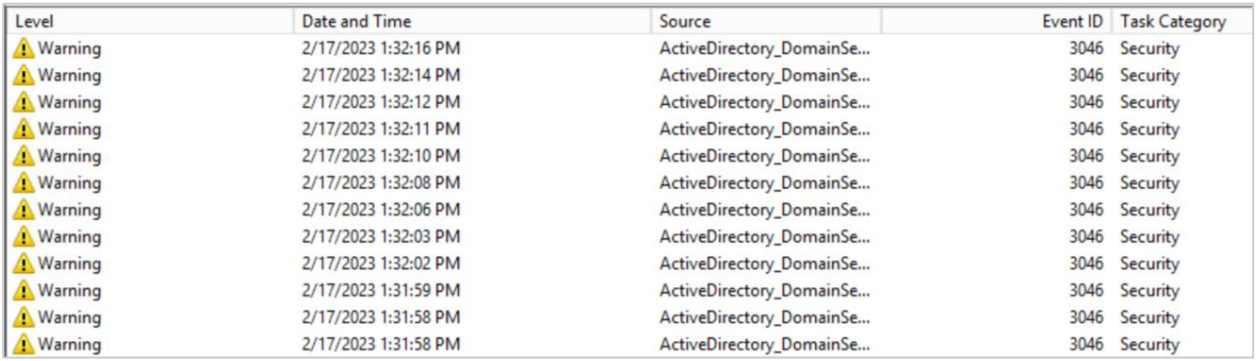
```
AD Object:  CN=PC2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=marvel,DC=local
Owner:  MARVEL\OwnershipTestAdmin
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : tm-win12-marvel
```

```
Exception calling "CommitChanges" with "0" argument(s): "Access is denied.
"
    + CategoryInfo          : NotSpecified: (:) [], MethodInvocationException
    + FullyQualifiedErrorId : DotNetMethodException
    + PSComputerName        : tm-win12-marvel

----------------------------------------------------
```

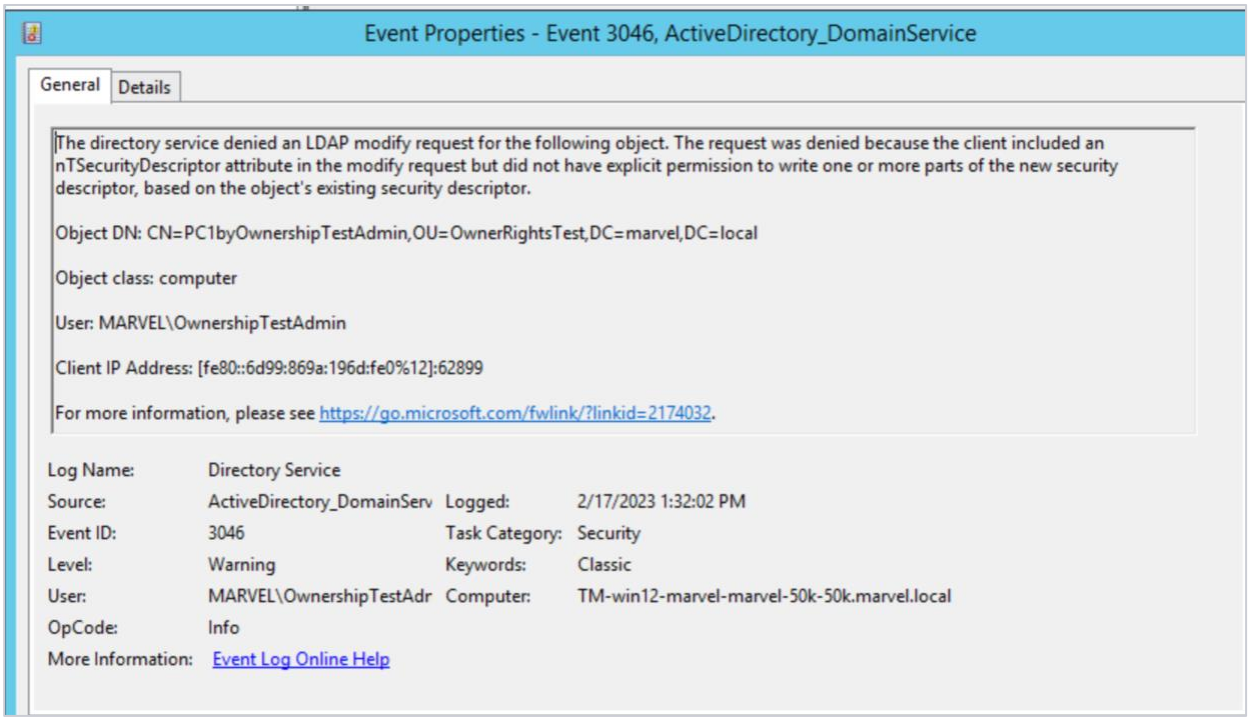
Looking at the DACLs for all the test objects, we only see that no computer objects were affected this time around:

Object DN	Object Owner	IdentityReference	AD Rights
OU=OUNobyOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestDFCO	MARVEL\AttackerControlled	GenericAll
CN=User2byOwnershipTestDFCO,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestDFCO	MARVEL\AttackerControlled	GenericAll
OU=OUNobyOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestAdmin	MARVEL\AttackerControlled	GenericAll
CN=User2byOwnershipTestAdmin,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestAdmin	MARVEL\AttackerControlled	GenericAll
CN=User2byOwnershipTestAO,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestAO	MARVEL\AttackerControlled	GenericAll
OU=OUNobyOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestDFCD	MARVEL\AttackerControlled	GenericAll
CN=User2byOwnershipTestDFCD,OU=NoOwnerRightsTest,DC=marvel,DC=local	MARVEL\OwnershipTestDFCD	MARVEL\AttackerControlled	GenericAll

Reviewing the event log on the Domain Controller we targeted, we find several 3046 events:



Level	Date and Time	Source	Event ID	Task Category
Warning	2/17/2023 1:32:16 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:14 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:12 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:11 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:10 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:08 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:06 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:03 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:32:02 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:31:59 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:31:58 PM	ActiveDirectory_DomainSe...	3046	Security
Warning	2/17/2023 1:31:58 PM	ActiveDirectory_DomainSe...	3046	Security



So, we find that the dSHeuristics character 29 for temporary removal of implicit owner for LDAP Modify operations is effective for computer objects, but not for any other AD object types.

The LDAP Modify operation change doesn't prevent an attacker from using Powermad or other methods to create (or takeover) a computer object with an attacker-known credential that can be modified by SELF rights to edit the SPN, DNSHostname, and PrincipalsAllowedToActOnBehalfOf. This means that Resource-Based Constrained Delegation (RBCD) attacks are still on the table when Authenticated Users have User Rights to "Add workstations to the domain" & the Machine-Account-Quota is a non-zero value, although perhaps not as readily targeted.

I am not 100% confident that the LDAP Modify operation is the only method to modify a computer object in AD. I wasn't able to discover another protocol to perform a modify operation on a computer object in order to abuse default object owner rights, but that doesn't mean one doesn't exist.

This change does provide friction against abusing object ownership of high value computer objects, such as Domain Controllers, for direct RBCD attacks. But it should be assumed that ownership of an AD computer object could still result in the takeover of the corresponding host:

The dSHeuristics character 28 for additional AuthZ verifications for LDAP Add operations did not prevent the use of Powermad to create attacker-controlled computer objects with a known password. My understanding is that the constraints on the LDAP Add operations primarily center around LDAP operations that attempt to supply a non-default security descriptor with the add request.

Operational Effects of KB5008383

I am not aware of any common operational effects for KB5008383 as most environments don't utilize LDAP operations to manage computer objects. That's not to say there aren't obscure or bespoke computer management solutions that don't use LDAP add or modify operations, but I wasn't able to locate any concern among systems administrators in online forums related to this patch and enforcement action. Whereas with KB5020276 (NetJoin Hardening Changes) there has been concern among system administrators and Microsoft followed up on that concern by creating exclusions and workarounds for imaging solutions.

In my opinion, the effects of KB5008383 will primarily affect malicious attempts to abuse non-standard ownership of computer objects, not production systems management.

Other Thoughts

Group Policy Creator Owners

Membership in Group Policy Creator Owners (while not also being a member of DA/EA) is another way for low-privileged users to end up as the individual object owner on a Group Policy Object. If a policy was originally created by a standard user or a lower-tier admin account and then subsequently linked to the Domain root, the Domain Controllers OU, OUs where AD Admins are stored, or OUs with other high-value Tier0 targets like AD CS CA hosts or Exchange servers are located, that non-standard ownership can be abused to escalate privilege and compromise the domain.

Another possible, although less common scenario, would be an attacker having the right to link a GPO to a high-level target OU (GP-Link right or All Extended Rights or higher on the OU) and also being able to compromise a GPO that a standard or lower-privileged user is owner on to link there.

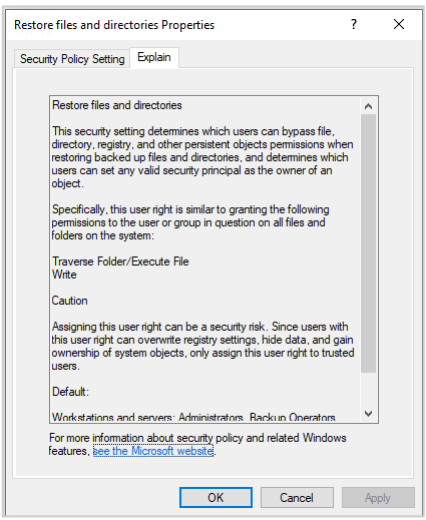
Applying Owner Rights to the CN=Policies,CN=System,DC=contoso,DC=com container could be helpful in preventing this potential for abuse. However, it's likely an even better idea to ensure that only AD Admins can create GPOs or apply them to OUs containing highly privileged objects.

Who Else Can Be the Owner?

I briefly touched on other ways that a security principal can become an owner in the Who's the Owner? section, at least beyond creating an object.

- Domain Administrators can always modify the ownership of AD Objects.
- A security principal with GenericAll (Full Control) rights on an object can make themselves the owner on that object.
- A security principal with WriteOwner rights on an object can make themselves the owner on that object.
- A security principal that has been granted the SeTakeOwnershipPrivilege (Take Ownership of files or other objects) right can make themselves or the Administrators group the owner of any object. Generally, this right is granted through the User Rights Assignment section in the 'winning' GPO applied to the Domain Controllers OU. By default, this right is granted to members of Administrators.
- Any security principal that has been granted the SeRestorePrivilege (Restore files and directories) right has the capability to change the owner in theory. I can prove this out on NTFS permissions model and services just fine, but I wasn't able to replicate it with AD Objects (yet).

While WriteOwner and SeTakeOwnershipPrivilege can only set the owner to themselves, SeRestorePrivilege, which is assigned by default to Administrators, Backup Operators, and Server Operators, can assign ownership to any security principal. At least I can confirm when it's combined with WriteOwner or SeTakeOwnershipPrivilege it allows setting any arbitrary security principal as owner.

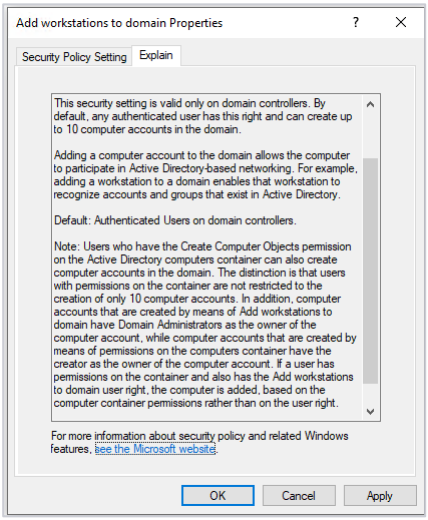


I was able to abuse SeTakeOwnershipPrivilege to fully compromise an AD domain with a simple PowerShell snippet by taking ownership on the domain root and then granting myself rights to DCSync the domain. But after quite a lot of effort I had no luck abusing SeRestorePrivilege in similar ways. I was able to see that SeRestorePrivilege does have some effect on AD Objects through its ability to grant modification of ownership to arbitrary security principals, but I wasn't able to abuse it directly

through PowerShell, ADSI, or LDAP in the ways that I did with SeTakeOwnershipPrivilege. Still, it seems possible if the correct method were utilized. After all, most AD-aware backup solutions utilize SeRestorePrivilege to not only restore files, directories, and registry keys but also AD Objects.

Other Remediations

While mitigating the effects of non-standard object ownership is the primary focus of this paper, it's important to consider issues around standard users having CREATOR_OWNER rights on a computer object. The default rights granted (outlined in the Review ACLs section) to the CREATOR_OWNER on a computer object are enough for many skilled attackers to compromise that device. And perhaps even worse, when authenticated users are allowed the default right of "Add workstations to the domain" (ms-DS-MachineAccountQuota + SeMachineAccountPrivilege) they can utilize tools like Powermad (and more) to create a computer account with a known password. This allows an attacker to abuse not just the CREATOR_OWNER rights, but the Self rights a computer object has on itself. Additionally, an attacker who has a machine account with a known password can abuse that machine account for authentication in the domain.



Make sure only members of your IT staff can add computers to the domain. Change the default SeMachineAccountPrivilege that's assigned by default to Authenticated User in the Default Domain Controllers policy GPO and/or set the ms-DS-MachineAccountQuote to 0. This makes many attack primitives around computer accounts significantly more challenging.

More To Discover?

While this ended up being a long paper, I don't believe it's exhaustive. Part of what has taken me so long to get this published is that I keep finding things to add as I dig into Active Directory and organization's Active Directory environments.

I still want to dig deeper into how security descriptors for new directory objects are created:

- https://learn.microsoft.com/en-us/windows/win32/ad/how-security-descriptors-are-set-on-new-directory-objects
- https://learn.microsoft.com/en-us/windows/win32/ad/creating-a-security-descriptor-for-a-new-directory-object

I've been thinking about other securable objects and how ownership impacts them:

- Directory Objects:
  - ADIDNS zones, nodes, records
  - Trust Objects
  - Schema
  - CN=Configuration
  - ACDS & CN=Public Key Services,CN=Services,CN=Configuration
- Files and Folders
- Named Pipes
- Processes
- Access Tokens
- Registry Keys
- Windows Service Objects
- Printer Objects
- Network Shares

I'd like to learn more about the intersection of security descriptors and software engineering concepts beyond PowerShell, ADSI, and LDAP. Things like Low-level Access Control:

- GetSecurityDescriptorOwner
- SetSecurityDescriptorOwner

I also didn't touch on some other defensive options around DACL abuse as it relates to security monitoring and detection. Properly configuring Advanced Auditing categories on Domain Controllers combined with setting up targeted SACLs could provide valuable detection capabilities.

There are also some relatively recent changes to NetJoin that intersect with computer object ownership, or at least creation. KB5020276 details the domain join hardening changes that were implemented in response to CVE-2022-38042. These NetJoin changes will have operational effects for many organizations that use solutions to re-image computers.

Conclusions

Object Ownership is not often considered when maintaining and defending Active Directory, where there are many other operational and security concerns that may take priority. Abuse of Object Ownership, and DACL abuse more generally, aren't usually mentioned in publicly available DFIR

reports. It's challenging to detect abuses for which the logging isn't enabled and collected by default. DACL abuse is not glitzy, glamorous, or expensive like the latest solutions on the RSAC show floor.

The reality is that when defenders and administrators don't consider the less glamorous bits of security and administration, it leaves openings for sufficiently advanced attackers to consider and take advantage of it for you. Better your own staff gets a handle on the issues before a threat actor in your systems comes to understand your weaknesses better than you do.

There are reactive approaches to the issue of Object Ownership abuse that involve regularly assessing the environment and making corrections, but these reactive approaches leave time and room for error.

Proactive approaches like applying the Owner Rights well-known SID to the environment in a properly designed OU structure and the recent approach taken by Microsoft with KB5008383 blocking the implicit WRITE_DAC capability of object owners on computer objects are more helpful.

These approaches should help organizations protect their on-premises Active Directory from ownership abuse. But what about ownership in ~~Azure AD~~ Entra ID? Maybe that's Part 2?

"The most secure network is a well-administered one." – Law Number Seven from the 10 Immutable Laws of Security Administration, Version 1

References:

I learned a lot about AD Object ownership and abusing it through conversations with Mike Saunders (@hardwaterhacker) at Red Siege Information Security. I also read through a lot of blogs and documentation:

- Managing Object Ownership: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/cc732983(v=ws.11)?redirectedfrom=MSDN
- Secure Identity: Who is the object owner in your domain? https://secureidentity.se/object-owner/
- SecAuthZ – Owner of a New Object: https://learn.microsoft.com/en-us/windows/win32/secauthz/owner-of-a-new-object
- AD DS: Owner Rights: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2008-R2-and-2008/dd125370(v=ws.10)
- Security Identifies (SIDS) New for Windows Vista: https://learn.microsoft.com/en-us/previous-versions/windows/it-pro/windows-vista/cc749445
- KB5008383—Active Directory permissions updates (CVE-2021-42291): https://support.microsoft.com/en-us/topic/kb5008383-active-directory-permissions-updates-cve-2021-42291-536d5555-ffba-4248-a60e-d6cbc849cde1
- MS-ADTS Open Specs 3.1.1.5.2.1.1 Per Attribute Authorization for Add Operation: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/ff004f3e-8920-4ba4-aaa7-346710171972
- CVE-2021-42291: https://msrc.microsoft.com/update-guide/vulnerability/CVE-2021-42291
- MS-ADTS Open Specs 6.1.3.4 Blocking Implicit Owner Rights: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/fb7c101d-ec8b-4fbf-bca8-7d7c2d747d0c
- MS-ADTS Open Specs 6.1.1.2.4.1.2 dSHeuristics: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/e5899be4-862e-496f-9a38-33950617d2c5
- Reddit /r/sysadmin KB5008383 Manual Steps Required: https://www.reddit.com/r/sysadmin/comments/rxp68w/kb5008383_manual_steps_required/
- Active Directory Concepts Part 1: https://social.technet.microsoft.com/wiki/contents/articles/16968.active-directory-concepts-part-1.aspx
- Four Active Directory EoP vulnerabilities were addressed in the November 2021 updates by Sander Berkouwer: https://dirteam.com/sander/2021/11/09/four-active-directory-elevation-of-privilege-vulnerabilities-were-addressed-in-the-november-2021-updates/

- Creating and Deleting Objects in Active Directory Domain Services: https://learn.microsoft.com/en-us/windows/win32/ad/creating-and-deleting-objects-in-active-directory-domain-services
- Active Directory Domain Services Overview: https://learn.microsoft.com/en-us/windows-server/identity/ad-ds/get-started/virtual-dc/active-directory-domain-services-overview
- Get-IncorrectADComputerObjectOwner by Jim Sykora: https://github.com/JimSycurity/ADPoSH/blob/main/Get-IncorrectADComputerObjectOwner
- Spiceworks, Changing Owner on AD Objects via PowerShell or DSACLS: https://community.spiceworks.com/topic/1245325-changing-owner-on-ad-objects-via-powershell-or-dsacls
- Setting a Control Access Right ACE in an Object's ACL: https://learn.microsoft.com/en-us/windows/win32/ad/setting-a-control-access-right-ace-in-an-objectampaposs-acl
- Delegating the Administration of Windows Server 2008 Active Directory Domain Services: https://www.microsoftpressstore.com/articles/article.aspx?p=2231764&seqNum=3
- Active Directory – Why change the owner of an object? https://learn.microsoft.com/en-us/answers/questions/817599/active-directory-why-change-the-owner-of-an-object.html
- User PowerShell to Explore Active Directory Security: https://devblogs.microsoft.com/scripting/use-powershell-to-explore-active-directory-security/
- MS-ADTS Owner and Group Defaulting Rules: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/3ac403a6-4e8e-488d-8ef6-c7fa1aa785b6
- MS-ADTS [Security Descriptor] Security Concerns: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/7afacb02-548b-4e74-bf96-04b0bf0c71b6
- MS-ADTS [Security Descriptor] Processing Specifics: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/e42f988c-72a0-4f8d-a705-7235eac175d9
- MS-ADTS SD Flag Control: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/932a7a8d-8c93-4448-8093-c79b7d9ba499
- MS-ADTS ACE Ordering Rules: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/dbfdc00c-1e4b-4165-939b-974e8ea23733
- MS-ADTS Security Descriptor Requirements: https://learn.microsoft.com/en-us/openspecs/windows_protocols/ms-adts/081c41f0-4c8d-4ab0-971d-77ec2504375a
- Understanding Get-ACL and AD Drive Output: https://devblogs.microsoft.com/powershell-community/understanding-get-acl-and-ad-drive-output/
- An Ace Up the Sleeve (pdf): https://specterops.io/wp-content/uploads/sites/3/2022/06/an_ace_up_the_sleeve.pdf
- BloodHound 1.3 – The ACL Attack Path Update: https://wald0.com/?p=112
- The Old New Thing: https://devblogs.microsoft.com/oldnewthing/20050818-09/?p=34533
- Abusing Forgotten Permissions on Computer Objects in Active Directory: https://dirkjanm.io/abusing-forgotten-permissions-on-precreated-computer-objects-in-active-directory/
- The Hacker Recipes - DACL abuse: https://www.thehacker.recipes/ad/movement/dacl

- ReadTeaming-Tactics-and-Techniques - Abusing Active Directory ACLs/Aces: https://github.com/mantvydasb/RedTeaming-Tactics-and-Techniques/blob/master/offensive-security-experiments/active-directory-kerberos-abuse/abusing-active-directory-acls-aces.md
- Access Control (Authorization) https://learn.microsoft.com/en-us/windows/win32/secauthz/access-control
- AD – Creator or Owner of an Object: https://social.technet.microsoft.com/Forums/en-US/e68e378b-8a19-461b-b3f0-a2719242106c/active-directory-creator-or-owner-of-an-object
- Taking Object Ownership in C++: https://learn.microsoft.com/en-us/windows/win32/secauthz/taking-object-ownership-in-c--
- SetSecurityDescriptorOwner function: https://learn.microsoft.com/en-us/windows/win32/api/securitybaseapi/nf-securitybaseapi-setsecuritydescriptorowner

Revision History

Date	Version	Description
2023-06-26	1.0	Initial release
2023-09-21	1.1	Updated dSHeuristics & CVE-2021-42291 Section
2024-08-26	1.2	Updated, revised, and reworded the opening Who's the Owner? section to read better and to explain the "default administrators group" functionality that determines who the owner is on newly created objects.
2025-02-21	1.3	Added detail on defaultSecurityDescriptors from the AD Schema, formatting, edits for readability, fix hyperlinks