What Is Kerberos Delegation? An Overview of Kerberos Delegation

blog.netwrix.com/2021/11/30/what-is-kerberos-delegation-an-overview-of-kerberos-delegation

Purpose of Kerberos Delegation

Kerberos delegation has been around for a long time (since Windows Server 2000, to be exact). But more often than not, engineers who work with Active Directory are not familiar with all the various implementations of Kerberos delegation, their uses and ways they can be abused. Some even confuse Kerberos delegation with delegated permissions.

Handpicked related content:

[Free Guide] Active Directory Security Best Practices

The practical use of Kerberos delegation is to enable an application to access resources hosted on a different server. One example is when an application, such as a web server, needs to access resources for the website hosted somewhere else, such as a SQL database. Instead of giving the service account running the web server access to the database directly, you can allow that service account to be delegated to the SQL server service. Once a user logs into the website, the service account will request access to the SQL server service on behalf of that user. This allows the user to get access to the content in the database that they've been provisioned to, without having to provision any access to the web server's service account itself.

Types of Kerberos Delegation

A few flavors of Kerberos delegation have evolved over the years. The original implementation from Windows Server 2000 is unconstrained delegation. Since then, stricter versions of delegation have come along that improve security: constrained delegation and resource-based constrained delegation. I'll dive deeper into each type of delegation below.

To configure delegation on a computer or user account, use the Delegation tab in Active Directory Users and Computers, as shown below. Note that user accounts must have a servicePrincipalName (SPN) set.

Figure 1. Delegation tab in Active Directory Users and Computers

The first option (in yellow) allows you to configure an account so that it is NOT allowed to be trusted for delegation; this is most commonly used for sensitive or administrative accounts that should never be used for delegation. The second option (in green) allows you to configure an account for unconstrained delegation. The third option (in red) allows you to configure an account for constrained delegation.

Unconstrained Delegation

This is the original implementation of delegation, and also the least secure. What does unconstrained delegation actually do? Under the covers, when unconstrained delegation is configured, the userAccountControl attribute of the object gets updated to include the "TRUSTED_FOR_DELEGATION" flag. When an object authenticates to a host with unconstrained delegation configured, the ticket-granting ticket (TGT) for that account gets stored in memory so that the host with unconstrained delegation configured can impersonate that user later, if needed.

Imagine a scenario where a privileged account authenticates to a host with unconstrained delegation configured. That account can access any configured service within the domain as that privileged user. To take it a step further, what if there were ways to force privileged accounts to authenticate to your host automatically? Using the "printer bug," you can get a domain controller to authenticate to your host, leaving the TGT for that account in memory.

Since mechanisms like the "printer bug" exist, unconstrained delegation is very insecure and should not be used if at all possible. One thing to note is that domain controllers, by default, are configured with unconstrained delegation. However, since your domain controllers should be much more secure than a random application server hosting a service, it should not be a problem.

Constrained Delegation

Introduced in Windows Server 2003, constrained delegation allows you to configure which services an account can be delegated to. This, in theory, limits the potential exposure if a compromise occurs.

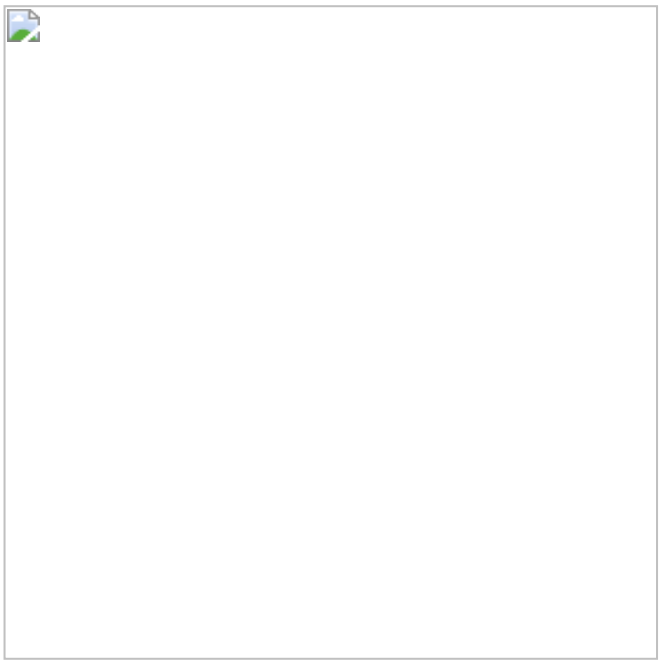

Figure 2. TestUserA can be delegated to the HTTP/test service.

One restriction to note for constrained delegation is that it does not work cross-forest.

When constrained delegation is set on an account, two things happen under the covers:

- The userAccountControl attribute for the object gets updated with the "TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION" flag.
- The msDS-AllowedToDelegateTo attribute gets populated with the SPN configured on the delegation tab.

Abusing constrained delegation is different than abusing unconstrained delegation. One common way it can be abused is if attackers are able to compromise the plaintext password or NTLM hash of a user account configured for constrained delegation. Using a tool like Kekeo, they are able to request a TGT for the account they have the password for, execute the TGS request for any user (as long as the user is not marked 'Sensitive'), and then inject the ticket and access the service they requested as that user.

Resource-Based Constrained Delegation

Introduced in Windows Server 2012, resource-based constrained delegation changes how you can configure constrained delegation, and it will work across a trust. Instead of specifying which object can delegate to which service, the resource hosting the service specifies which objects can delegate to it. From an administrative standpoint, this allows the resource owner to control who can access it. For example, instead of specifying with constrained delegation that the WebServer service account can delegate to the SQL Service to access the database, you can specify on the SQL server service account that the WebServer service account has permissions to delegate access to it.

Resource-based constrained delegation is configured by populating the msDS-AllowedToActOnBehalfOfOtherIdentity attribute on the target resource with the SID of the object that is allowed to delegate to it. To configure resource-based constrained delegation, you need to use PowerShell; there is no GUI component within Active Directory Users and Computers and the Attribute Editor page does not allow for manual modification of this attribute.

You can read more about Resource-Based Constrained Delegation and ways to abuse it here.

Identifying Existing Kerberos Delegation

Now that you understand some of the basics for the different types of delegation and some ways they can be abused, I want to share with you a method you can use to wrap your head around what types of delegation are already configured in your environment. We will specifically look to highlight insecure scenarios, such as unconstrained delegation being configured on objects other than domain controllers.

Here’s a script, which was originally posted in the Microsoft Technet gallery, that will identify accounts with unconstrained, constrained and resource-based constrained delegation configured, highlighting information and potential warnings about the configurations it lists:

```
<#
.Synopsis
    Search the domain for accounts with Kerberos Delegation.
.DESCRIPTION
    Kerberos Delegation is a security sensitive configuration. Especially
    full (unconstrained) delegation has significant impact: any service
    that is configured with full delegation can take any account that
    authenticates to it, and impersonate that account for any other network
    service that it likes. So, if a Domain Admin were to use that service,
    the service in turn could read the hash of KRBRTG and immediately
    effectuate a golden ticket. Etc :)

    This script searches AD for regular forms of delegation: full, constrained,
    and resource based. It dumps the account names with relevant information
(flags)
    and adds a comment field for special cases. The output is a PSObject that
    you can use for further analysis.

    Note regarding resource based delegation: the script dumps the target
    services, not the actual service doing the delegation. I did not bother
    to parse that out.

    Main takeaway: chase all services with unconstrained delegation. If
    these are _not_ DC accounts, reconfigure them with constrained delegation,
    OR claim them als DCs from a security perspective. Meaning, that the AD
    team manages the service and the servers it runs on.

.EXAMPLE
   .Search-KerbDelegatedAccounts.ps1 | out-gridview
.EXAMPLE
   .Search-KerbDelegatedAccounts.ps1 -DN "ou=myOU,dc=sol,dc=local"
.NOTES
    Version:   0.1 : first version.
                   0.2 : expanded LDAP filter and comment field.
    Author:         Willem Kasdorp, Microsoft.
    Creation Date:  1/10/2016
    Last modified:  4/11/2017
#>

[CmdletBinding()]
Param
(
    # start the search at this DN. Default is to search all of the domain.
    [string]$DN = (Get-ADDomain).DistinguishedName
)

$SERVER_TRUST_ACCOUNT = 0x2000
$TRUSTED_FOR_DELEGATION = 0x80000
$TRUSTED_TO_AUTH_FOR_DELEGATION= 0x1000000
$PARTIAL_SECRETS_ACCOUNT = 0x4000000
$bitmask = $TRUSTED_FOR_DELEGATION -bor $TRUSTED_TO_AUTH_FOR_DELEGATION -bor
$PARTIAL_SECRETS_ACCOUNT

# LDAP filter to find all accounts having some form of delegation.
# 1.2.840.113556.1.4.804 is an OR query.
$filter = @"
(&
```

```
  (servicePrincipalname=*)
  (|
    (msDS-AllowedToActOnBehalfOfOtherIdentity=*)
    (msDS-AllowedToDelegateTo=*)
    (UserAccountControl:1.2.840.113556.1.4.804:=$bitmask)
  )
  (|
    (objectcategory=computer)
    (objectcategory=person)
    (objectcategory=msDS-GroupManagedServiceAccount)
    (objectcategory=msDS-ManagedServiceAccount)
  )
)
"@ -replace "[sn]", ''

$propertylist = @(
    "servicePrincipalname",
    "useraccountcontrol",
    "samaccountname",
    "msDS-AllowedToDelegateTo",
    "msDS-AllowedToActOnBehalfOfOtherIdentity"
)
Get-ADObject -LDAPFilter $filter -SearchBase $DN -SearchScope Subtree -Properties
$propertylist -PipelineVariable account | ForEach-Object {
    $isDC = ($account.useraccountcontrol -band $SERVER_TRUST_ACCOUNT) -ne 0
    $fullDelegation = ($account.useraccountcontrol -band $TRUSTED_FOR_DELEGATION)
-ne 0
    $constrainedDelegation = ($account.'msDS-AllowedToDelegateTo').count -gt 0
    $isRODC = ($account.useraccountcontrol -band $PARTIAL_SECRETS_ACCOUNT) -ne 0
    $resourceDelegation = $account.'msDS-AllowedToActOnBehalfOfOtherIdentity' -ne
$null

    $comment = ""
    if ((-not $isDC) -and $fullDelegation) {
        $comment += "WARNING: full delegation to non-DC is not recommended!; "
    }
    if ($isRODC) {
        $comment += "WARNING: investigation needed if this is not a real RODC; "
    }
    if ($resourceDelegation) {
        # to count it using PS, we need the object type to select the correct
function... broken, but there we are.
        $comment += "INFO: Account allows delegation FROM other server(s); "
    }
    if ($constrainedDelegation) {
        $comment += "INFO: constrained delegation service count:
$(($account.'msDS-AllowedToDelegateTo').count); "
    }

    [PSCustomobject] @{
        samaccountname = $account.samaccountname
        objectClass = $account.objectclass
        uac = ('{0:x}' -f $account.useraccountcontrol)
        isDC = $isDC
        isRODC = $isRODC
        fullDelegation = $fullDelegation
```

```
        constrainedDelegation = $constrainedDelegation
        resourceDelegation = $resourceDelegation
        comment = $comment
    }
}
```

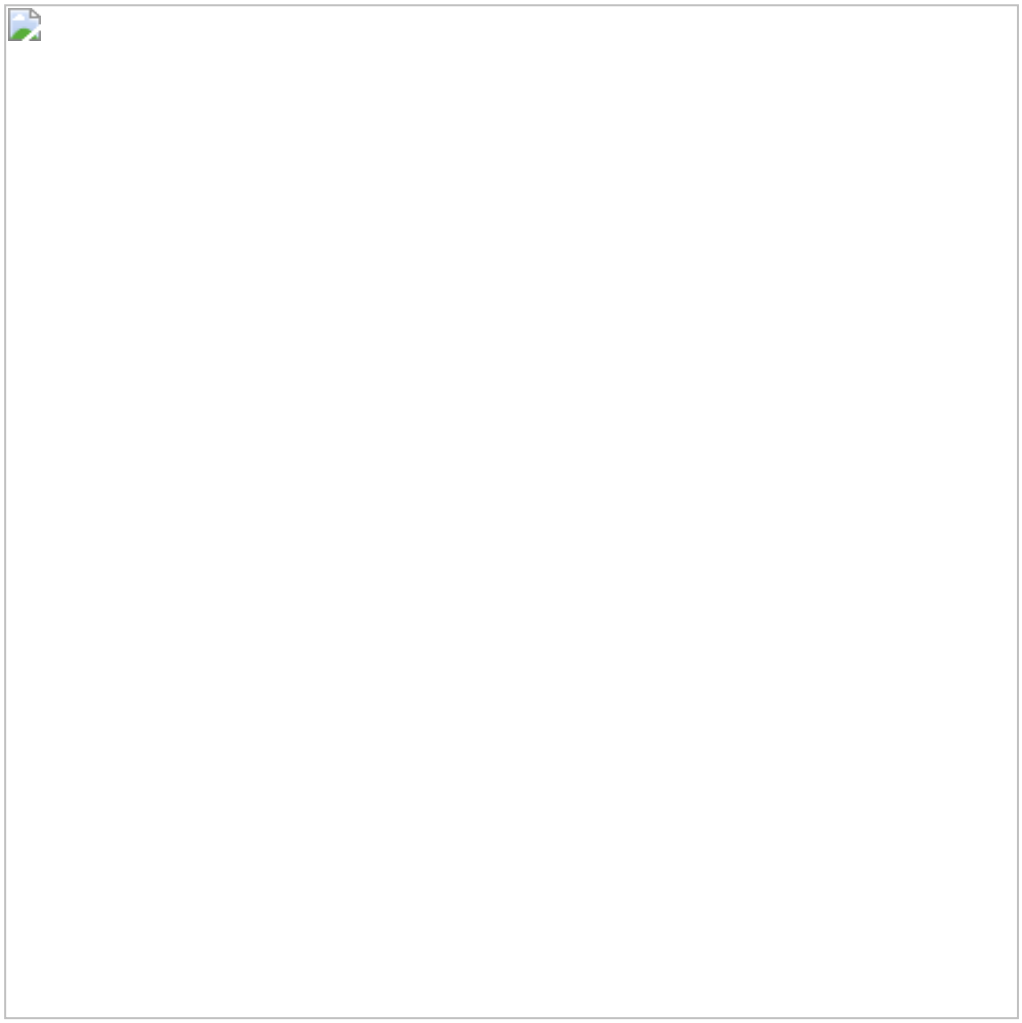

Figure 3. Sample script to identify problematic delegation

How Netwrix can help

The Netwrix Active Directory security solution helps you secure your Active Directory from end to end. You can:

- Identify and mitigate vulnerabilities in your Active Directory: excessive permissions, “shadow” admins, stale accounts, weak passwords, and more.
- Control AD configurations and permissions, enforce strong password policies, and prevent credential theft.
- Detect even advanced threats to stop bad actors in their tracks before they complete their mission.

- Instantly contain a security breach with automated response actions, minimizing the damage to your business.
- Roll back or recover from malicious or otherwise improper changes with minimal downtime.

Kevin Joyce
Senior Technical Product Manager at Netwrix. Kevin is passionate about cyber-security and holds a Bachelor of Science degree in Digital Forensics from Bloomsburg University of Pennsylvania.