```
# ============================

# Set the files Output

$OutputFolder = "C:\Temp\ReportPKI\"

New-Item -ItemType Directory -Path $OutputFolder -Force | Out-Null


# ============================

# PKI data recovery

$ConfigContext = ([ADSI]"LDAP://RootDSE").configurationNamingContext

$TemplatePath = "LDAP://CN=Certificate Templates,CN=Public Key
Services,CN=Services,$ConfigContext"


# ============================

# Function to permit the grant Enroll and Read without PSPKI module

# It relies on ADSI ObjectSecurity object manipulation

function Grant-PkiTemplateEnrollmentPermission {

[CmdletBinding()]

param(

[Parameter(Mandatory=$true)]

[string]$TemplateName,

[Parameter(Mandatory=$true)]

[string]$IdentityName # Example: 'DOMAIN\Group' or 'User'

)

Write-Host "Attempting to grant 'Read' and 'Enroll' permissions on template
'$TemplateName' for '$IdentityName'..." -ForegroundColor Yellow


# ============================

# 1. Search template in Active Directory
```

```
try {
$searcher = New-Object System.DirectoryServices.DirectorySearcher
$searcher.SearchRoot = [ADSI]$TemplatePath
$searcher.Filter = "(cn=$TemplateName)"
$templateResult = $searcher.FindOne()
if (-not $templateResult) {
Write-Error "Template '$TemplateName' not find in Active Directory."
return
}
$template = $templateResult.GetDirectoryEntry()
$sd = $template.ObjectSecurity

# ============================
# 2. Get the identity object (SecurityIdentifier)
$Identity = New-Object System.Security.Principal.NTAccount($IdentityName)
# GUID for Enroll (required for 'ExtendedRight' permission)
$EnrollGuid = [Guid]::Parse("0e10c968-78fb-11d2-90d4-00c04f79dcaa")

# ============================
# 3. Define permissions (Access Control Entry - ACE)
# ACE for the 'Enroll' permission (ExtendedRight)
$EnrollAccess = [System.DirectoryServices.ActiveDirectoryRights]::ExtendedRight
$EnrollAce = New-Object System.DirectoryServices.ActiveDirectoryAccessRule(
$Identity,
$EnrollAccess,
"Allow",
```

```
$EnrollGuid, # ObjectType - GUID for the Enroll grant
"None"
)

# ACE for 'Read' permission (Generic and ReadProperty)
$ReadAccess = [System.DirectoryServices.ActiveDirectoryRights]::GenericRead -bor [System.DirectoryServices.ActiveDirectoryRights]::ReadProperty
$ReadAce = New-Object System.DirectoryServices.ActiveDirectoryAccessRule(
$Identity,
$ReadAccess,
"Allow"
)

# ============================
# 4. Add new ACEs in Security Descriptor
$sd.AddAccessRule($ReadAce)
$sd.AddAccessRule($EnrollAce)

# ============================
# 5.Save Security Descriptor
$template.CommitChanges()
Write-Host " 'Read' and 'Enroll' grant permitted successfully for '$IdentityName' on the template '$TemplateName'." -ForegroundColor Green
} catch {
Write-Error "Errors during grant assignment: $($_.Exception.Message)"
}
}
```

```
# The new Grant-PkiTemplateEnrollmentPermission function
$searcher = New-Object System.DirectoryServices.DirectorySearcher
$searcher.SearchRoot = [ADSI]$TemplatePath
$searcher.Filter = "(objectClass=pKICertificateTemplate)"
$searcher.PageSize = 1000
$results = $searcher.FindAll()

# GUID of specific grant
$GUIDs = @{
"Enroll" = [Guid]::Parse("0e10c968-78fb-11d2-90d4-00c04f79dcaa")
"Autoenroll" = [Guid]::Parse("a05b8cc2-17bc-4802-a710-e7c15ab866a2")
}

# Special group to filter Read/Write
# UPDATE: added "RAS and IAS Servers" for Fix 2 (RASAndIASServer)
$targetedAccounts = @(
"Domain Users",
"Domain Computers",
"Domain Controllers",
"Enterprise Read-only Domain Controllers",
"ENTERPRISE DOMAIN CONTROLLERS",
"RAS and IAS Servers"
)

# ============================
```

```
# Recovery data
$Report = @{}
foreach ($result in $results) {
$template = $result.GetDirectoryEntry()
$cn = $template.cn
$displayName = $template.displayName
$sd = $template.ObjectSecurity
foreach ($ace in $sd.Access) {
$identity = $ace.IdentityReference.ToString()
$key = "$cn|$displayName|$identity"
if (-not $Report.ContainsKey($key)) {
$Report[$key] = [PSCustomObject]@{
Template = $cn
DisplayName = $displayName
Identity = $identity
FullControl = $false
Read = $false
Write = $false
Enroll = $false
AutoEnroll = $false
}
}
$rights = $ace.ActiveDirectoryRights
$accessMask = $ace.AccessMask
$objectType = $ace.ObjectType
$accessType = $ace.AccessControlType
```

```
# ============================

# FullControl grant

if ($accessType -eq "Allow" -and $rights -eq
[System.DirectoryServices.ActiveDirectoryRights]::GenericAll) {

$Report[$key].FullControl = $true

$Report[$key].Read = $true

$Report[$key].Write = $true

$Report[$key].Enroll = $true

$Report[$key].AutoEnroll = $true

}


# ============================

# Read / Write grant

$isTargeted = $false

foreach ($t in $targetedAccounts) {

#Maintain the original matching, which is robust for identities like DOMAIN\Group

if ($identity -match [regex]::Escape($t)) {

$isTargeted = $true

break

}

}

if ($accessType -eq "Allow") {

# --- Logical Read ---

$canRead = ($rights -band
[System.DirectoryServices.ActiveDirectoryRights]::ReadProperty) -ne 0

# If the account is targeted,the condition is ReadProperty and bit will be 0x10
```

```
if ($isTargeted) {
$canRead = $canRead -and (($accessMask -band 0x10) -ne 0)
}
# If NOT targeted, the condition remains just ReadProperty (as calculated before)
if ($canRead) {
$Report[$key].Read = $true
}


# ============================
# --- Logical Write ---
# UPDATE FIX 1: We include WriteDacl (0x40000) which represents 'Writing' in the GUI interface.
$canWrite = ($rights -band [System.DirectoryServices.ActiveDirectoryRights]::WriteProperty) -ne 0 -or `
(($rights -band [System.DirectoryServices.ActiveDirectoryRights]::WriteDacl) -ne 0)
# If it's a targeted account ($isTargeted), we apply the restriction (only on WriteProperty):
if ($isTargeted) {
# We apply the 0x20 restriction only if WriteProperty (0x2) is set and WriteDacl (0x40000) is not.
# If WriteDacl is set, it is a generic write and 0x20 is not required.
if (($rights -band [System.DirectoryServices.ActiveDirectoryRights]::WriteProperty) -ne 0 -and `
(($rights -band [System.DirectoryServices.ActiveDirectoryRights]::WriteDacl) -eq 0))
{
$canWrite = $canWrite -and (($accessMask -band 0x20) -ne 0)
}
}
```

```
# If Write Property is set, but the WriteDacl check is False, the Write is not set.
if ($canWrite) {
$Report[$key].Write = $true
}
}

# ============================
# AutoEnroll
if ($accessType -eq "Allow" -and $objectType -eq $GUIDs.Autoenroll) {
$Report[$key].AutoEnroll = $true
}

# ============================
# Enroll
if ($accessType -eq "Allow" -and (
$objectType -eq $GUIDs.Enroll -or
($accessMask -band 0x2) -or
($rights -band [System.DirectoryServices.ActiveDirectoryRights]::ExtendedRight)
)) {
$Report[$key].Enroll = $true
}
}
}

# ============================
# CSV export
```

```
$csvPath = Join-Path $OutputFolder "PKI_Template_Assessment.csv"
$Report.Values | Export-Csv -Path $csvPath -NoTypeInformation -Encoding UTF8

# ============================
# HTML export with chart
$htmlPath = Join-Path $OutputFolder "PKI_Template_Assessment.html"
$timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
$username = $env:USERNAME
$computer = $env:COMPUTERNAME

# ============================
# Stats
$total = $Report.Count
$templates = ($Report.Values | Select-Object -ExpandProperty Template | Sort-Object -Unique).Count
$identities = ($Report.Values | Select-Object -ExpandProperty Identity | Sort-Object -Unique).Count
$counts = @{
FullControl = ($Report.Values | Where-Object { $_.FullControl }).Count
Read = ($Report.Values | Where-Object { $_.Read }).Count
Write = ($Report.Values | Where-Object { $_.Write }).Count
Enroll = ($Report.Values | Where-Object { $_.Enroll }).Count
AutoEnroll = ($Report.Values | Where-Object { $_.AutoEnroll }).Count
}
$perc = @{}
foreach ($k in $counts.Keys) {
$perc[$k] = if ($total -gt 0) { [math]::Round(($counts[$k] / $total) * 100, 1) } else { 0 }
```

```
}


# ============================
# HTML BUILD
$style = @"
<style>
body { font-family: Arial; margin: 20px; }
table { border-collapse: collapse; width: 100%; margin-bottom: 20px; }
th, td { border: 1px solid #ccc; padding: 8px; text-align: center; }
th { background-color: #f2f2f2; cursor: pointer; }
.true { color: green; font-weight: bold; }
.false { color: red; font-weight: bold; }
#searchInput { margin-bottom: 10px; padding: 5px; width: 300px; }
.summary { background-color: #f9f9f9; padding: 10px; border-radius: 8px; border: 1px solid #ccc; }
.chart-container { width: 100%; max-width: 600px; margin-top: 20px; }
.bar { height: 24px; background-color: #4caf50; text-align: right; padding-right: 8px; color: white; border-radius: 4px; transition: width 0.5s; }
.bar:hover { background-color: #2e7d32; }
.compact-table { width: auto; margin-top: 10px; }
.compact-table th { background-color: #e9e9e9; }
</style>
"@
$script = @"
<script>
function sortTable(n) {
var table = document.getElementById('pkiTable');
```

```
var rows = Array.from(table.rows).slice(1);
var asc = table.rows[0].cells[n].getAttribute('data-asc') !== 'true';
rows.sort(function(a, b) {
var x = a.cells[n].innerText.toLowerCase();
var y = b.cells[n].innerText.toLowerCase();
return asc ? (x > y ? 1 : -1) : (x < y ? 1 : -1);
});
rows.forEach(row => table.tBodies[0].appendChild(row));
table.rows[0].cells[n].setAttribute('data-asc', asc);
}
function filterTable() {
var input = document.getElementById('searchInput').value.toLowerCase();
var rows = document.getElementById('pkiTable').getElementsByTagName('tr');
for (var i = 1; i < rows.length; i++) {
var rowText = rows[i].innerText.toLowerCase();
rows[i].style.display = rowText.includes(input) ? '' : 'none';
}
}
function animateBars() {
document.querySelectorAll('.bar').forEach(bar => {
var value = bar.getAttribute('data-value');
bar.style.width = value + '%';
});
}
window.onload = animateBars;
</script>
```

"@

$html = @"

<html>

<head>

<meta charset='UTF-8'>

<title>PKI Template Assessment</title>

$style

</head>

<body>

<h2>PKI Template Assessment Report</h2>

Generated by: $username on $computer at $timestamp

LDAP Path: $TemplatePath

<input type='text' id='searchInput' onkeyup='filterTable()' placeholder=' Find using [username] or [template name]...'>

<table id='pkiTable'>

<thead>

<tr>

<th onclick='sortTable(0)'>Template</th>

<th onclick='sortTable(1)'>DisplayName</th>

<th onclick='sortTable(2)'>Identity</th>

<th onclick='sortTable(3)'>FullControl</th>

<th onclick='sortTable(4)'>Read</th>

<th onclick='sortTable(5)'>Write</th>

<th onclick='sortTable(6)'>Enroll</th>

<th onclick='sortTable(7)'>AutoEnroll</th>

</tr>

```
</thead>

<tbody>

"@

foreach ($row in $Report.Values) {

$fcStar = if ($row.FullControl) { " " } else { "" }

$html += "<tr>"

$html += "<td>$($row.Template)</td>"

$html += "<td>$($row.DisplayName)</td>"

$html += "<td>$($row.Identity)</td>"

$html += "<td class='$(if ($row.FullControl) {'true'} else {'false'})'>$($row.FullControl) $fcStar</td>"

$html += "<td class='$(if ($row.Read) {'true'} else {'false'})'>$($row.Read)</td>"

$html += "<td class='$(if ($row.Write) {'true'} else {'false'})'>$($row.Write)</td>"

$html += "<td class='$(if ($row.Enroll) {'true'} else {'false'})'>$($row.Enroll)</td>"

$html += "<td class='$(if ($row.AutoEnroll) {'true'} else {'false'})'>$($row.AutoEnroll)</td>"

$html += "</tr>"

}


# ============================

# Final report

$html += "</tbody></table>

<div class='summary'>

<h3> STATS REPORT</h3>

<p><strong>Total Templates:</strong> $templates<br>

<strong>Total Identities:</strong> $identities<br>

<strong>Total Records (ACE):</strong> $total</p>
```

```
<!-- Tabel centered -->

<div style='text-align:center;'>

<table class='compact-table' style='margin:auto; width:60%; font-size:13px;'>

<thead>

<tr><th>Permission</th><th>Count</th><th>Percentage</th></tr>

</thead>

<tbody>

<tr><td>FullControl</td><td>$($counts.FullControl)</td><td>$($perc.FullControl)%</td>
</tr>

<tr><td>Read</td><td>$($counts.Read)</td><td>$($perc.Read)%</td></tr>

<tr><td>Write</td><td>$($counts.Write)</td><td>$($perc.Write)%</td></tr>

<tr><td>Enroll</td><td>$($counts.Enroll)</td><td>$($perc.Enroll)%</td></tr>

<tr><td>AutoEnroll</td><td>$($perc.AutoEnroll)</td><td>$($perc.AutoEnroll)%</td></tr>

</tbody>

</table>

</div>

<!-- Bars chart -->

<div style='display:flex; justify-content:center; align-items:flex-end; gap:18px;
height:220px; margin-top:25px;'>

<div style='text-align:center;'>

<div style='background-color:#2e7d32; width:50px;
height:$([math]::Max(8,($perc.FullControl*2)))px; color:white; display:flex; align-
items:center; justify-content:center; border-radius:6px; font-
weight:bold;'>$($perc.FullControl)%</div>

<div>FullControl</div>

</div>

<div style='text-align:center;'>
```

<div style='background-color:#2e7d32; width:50px;
height:$([math]::Max(8,($perc.Read*2)))px; color:white; display:flex; align-items:center;
justify-content:center; border-radius:6px; font-weight:bold;'>$($perc.Read)%

Read

<div style='background-color:#2e7d32; width:50px;
height:$([math]::Max(8,($perc.Write*2)))px; color:white; display:flex; align-items:center;
justify-content:center; border-radius:6px; font-weight:bold;'>$($perc.Write)%

Write

<div style='background-color:#2e7d32; width:50px;
height:$([math]::Max(8,($perc.Enroll*2)))px; color:white; display:flex; align-items:center;
justify-content:center; border-radius:6px; font-weight:bold;'>$($perc.Enroll)%

Enroll

<div style='background-color:#2e7d32; width:50px;
height:$([math]::Max(8,($perc.AutoEnroll*2)))px; color:white; display:flex; align-
items:center; justify-content:center; border-radius:6px; font-
weight:bold;'>$($perc.AutoEnroll)%

AutoEnroll

$script

</body></html>

"

```
$html | Out-File -FilePath $htmlPath -Encoding UTF8

Write-Host " Report created and exported properly in: $OutputFolder"

# Usage example (uncomment to test the granting function)

# Grant-PkiTemplateEnrollmentPermission -TemplateName "WebServer" -IdentityName "MyDomain\WebServersGroup"
```