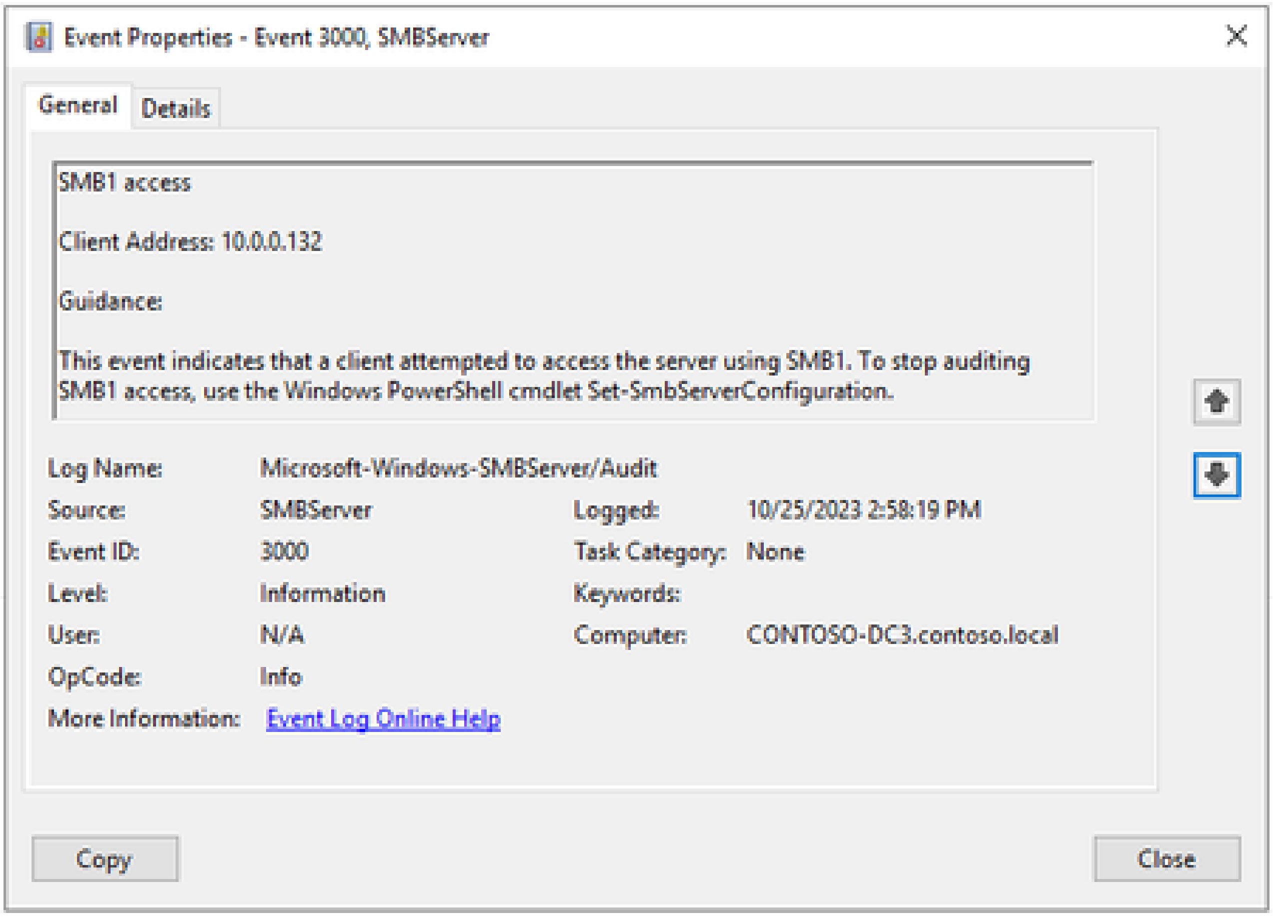
Active Directory Hardening Series - Part 2 – Removing SMBv1

techcommunity.microsoft.com/blog/coreinfrastructureandsecurityblog/active-directory-hardening-series---part-2-—-removing-smbv1/3988317



```
SMB1 access

Client Address: 10.0.0.132

Guidance:

This event indicates that a client attempted to access the server using SMB1. To stop auditing
SMB1 access, use the Windows PowerShell cmdlet Set-SmbServerConfiguration.

Log Name:          Microsoft-Windows-SMBServer/Audit
Source:            SMBServer                Logged:         10/25/2023 2:58:19 PM
Event ID:          3000                     Task Category:  None
Level:             Information              Keywords:
User:              N/A                      Computer:       CONTOSO-DC3.contoso.local
OpCode:            Info
More Information:  Event Log Online Help
```

Blog Post

Hi All! Jerry Devore back again with another hardening Active Directory topic. Before we jump into the technical stuff, I would like to briefly share some tips for structuring a protocol hardening project. I picked up these suggestions from working with customers who have been successful in their protocol hardening efforts.

Tip #1 - Collaborate with Applications Owners - Application dependencies always seem to be the preverbal "long pole in the tent" when it comes to removing insecure protocols. Often application owners lack an understanding of the risks or how their applications could be reconfigured to remove dependencies. Achieving remediation may require going the extra mile to provide technical explanations and guidance. It is often helpful to provide them with reports showing which accounts and devices are dependent on the weak protocol. Whatever you do, don't simply chuck it over the wall to the applications owners and expect things to get fixed.

Tip #2 - Get sponsorship for the project - On prem applications are heavily dependent on Active Directory and the impact to the organization will be felt far and wide if it becomes compromised. A hardening project should not be solely driven by the Active Directory operations or architecture teams. The security organization should be fully behind this effort and assist by imposing compliance requirements and deadlines. There is a good chance that the CISO's office already has some of these protocols on its radar so be proactive and ask to join forces.

Tip #3 - Work smarter not harder - If you are fortunate enough to have a SIEM, leverage queries and dashboards to visualize the data without a bunch of busy work. If the events you need are not already ingested into the SEIM you may get push back which is why you want to have sponsorship from the CISO. Also, keep in mind that the impact to log collection can be minimized if you collect only the events required as opposed to entire logs.

Tip #4 - Put a full court press on removing legacy systems - In an ideal world, organizations would only have current and supported operating systems. However, in my role I have found there are often a few legacy servers for which there is no simple replacement. While some partial mitigations can be implemented for those old systems, don't lose sight of the reality that they represent ongoing vulnerabilities that put your entire organization at risk.

Disabling SMBv1

Ok, let’s get into today's topic which is removing SMBv1 from domain controllers. Like my previous blog on NTLM, a lot of great content has already been written on SMBv1. My objective is to not to rehash the **why** but rather focus on **how** you can take action in a production environment. However, it is worth quickly summarizing the considerations for making SMBv1 removal a priority:

- SMBv1 is old. Like OS2 1.0 old. Just like you should not get your 386 running Windows for Workgroups out of the attic and connect it to the Internet, you should not be trusting SMBv1 with your enterprise security.
- Numerous security and performance enhancements have been introduced with SMB 2.x and 3.x. While Windows will negotiate the highest mutually supported version, your devices can be duped into falling back to SMBv1 if it is enabled.
- SMBv1 is susceptible to some nasty vulnerabilities.

- Windows Server 2003 (and earlier) and Windows XP are limited to SMBv1. That should be a moot piece of trivia, but unfortunately many organizations have a few such legacy devices lingering around.

Auditing

The good news is that auditing can be enabled on 2008 R2 / Windows 7 and later to log when SMBv1 connections are negotiated. The steps to enabling that auditing can be found here. The not so good news is that the server event (ID 3000) will often only capture the IP address of the client and does not log the name of the authenticating account.

If you are lucky, enabling auditing will show no SMBv1 connections being made to your domain controller. If you find that to be the case, go buy some lottery tickets and then proceed with disabling SMBv1 on your servers. If you are not so fortunate and want to quickly export a list of SMBv1 clients from the events you could use a script like this.

```
# Define the log name and output file path

$logName = 'Microsoft-Windows-SMBServer/Audit'

$outputFile = 'C:\temp\SMBv1_Connections.csv'


# Get events with EventID 3000 from the specified log

$events = Get-WinEvent -LogName $logName -FilterXPath "*[System[EventID=3000]]"


# Create an empty array to store event data

$eventData = @()


# Iterate through each event and extract required fields

foreach ($event in $events) {

  $clientAddress = $event.Properties[0].Value # Extract ClientName data from EventData
  $timeCreated = $event.TimeCreated # Extract TimeCreated SystemTime


  # Create a custom object with required fields
  $eventObject = New-Object PSObject -Property @{
    'ClientAddress' = $clientAddress
    'TimeCreated' = $timeCreated
  }


  # Add the custom object to the event data array
  $eventData += $eventObject
}


# Export the event data to CSV file
$eventData | Export-Csv -Path $outputFile -NoTypeInformation


Write-Output "Events exported to $outputFile"
```

Such a log file is a nice start, but I think we can do better. After all, wouldn't it be nice if we could capture the name of the authenticating account and the device name rather than an IP address? Here are some suggestions on how to create a more intelligent script.

Step 1 - Capture Account name

The PowerShell Cmdlet Get-SmbSession can be used to list all active SMB sessions on a server. As you can see from this example the properties for the session returned the ClientUserName and the SMB dialect version. By filtering based on the dialect version we can focus on only the SMBv1 connections.

Step 2 - Resolve Computer and map to AD object

The obvious solution to resolving the IP address from ID 3000 events is a DNS reverse lookup. However, PTR records are often inaccurate or missing. Microsoft Defender for Identity uses logic referred to as NNR to query devices directly to resolve computer names and will fall back to DNS if those techniques are not successful. NNR leverages three types of queries which are NTLM over RPC, NetBIOS and RDP Client Hello pings. Of those, NetBIOS queries using **nbtstat -A** is the easiest one to script with PowerShell. The other two would require more advanced coding. Once the computer name is resolved a call to Active Directory using Get-Adcomputer can be used to get other helpful information such as OS Version and the OU location.

Step 3 - Triggering the script

When a device has a dynamic address, it is important to resolve the computer name before the address is assumed by another device. One way that can be done is to use a scheduled task to run the script every time an ID 3000 event is logged. If you didn't know, scheduled tasks have an "**On an event**" trigger which can cause the scheduled task to run when an event is logged instead of at a particular time interval.

Bringing it all together

Here is an example of a script that implements those steps. It is far from perfect given it creates individual local logs per domain controller, it is prone to logging duplicate connections and it only uses nbtstat -a to directly resolve device names. However, it provides a good starting point to help seed your code.

```
# Get SMB sessions for the local server

$sessions = Get-SmbSession | Where-Object { $_.Dialect -lt "2.0" }


# Create an array to store session information
```

```
$sessionInfo = @()


# Iterate through each SMB session

foreach ($session in $sessions) {

  $clientComputerName = $session.ClientComputerName

  $resolutionMethod = "Not Resolved"


  # Use nbtstat to resolve the NetBIOS name if needed

  if ($clientComputerName -match "\d+\.\d+\.\d+\.\d+") {

    $nbtstatResult = nbtstat -A $clientComputerName 2>&1

    $netbiosName = "Not Found"

    $netbiosMatch = $nbtstatResult | Select-String -Pattern '(\S+)\s+
<00>\s+UNIQUE\s+Registered'

    if ($netbiosMatch) {

      $netbiosName = $netbiosMatch.Matches[0].Groups[1].Value

      $clientComputerName = $netbiosName

      $resolutionMethod = "nbtstat"

    }

  }


  # If nbtstat resolution fails, attempt reverse DNS lookup and truncate domain suffix

  if ($resolutionMethod -eq "Not Resolved" -and $clientComputerName -match
"\d+\.\d+\.\d+\.\d+") {

    try {

      $resolvedName =
[System.Net.Dns]::GetHostEntry($clientComputerName).HostName

      # Truncate domain suffix to get short computer name
      $clientComputerName = $resolvedName.Split('.')[0]

      $resolutionMethod = "Reverse DNS"

    } catch {

      # Reverse DNS lookup failed, keep the original client computer name

    }

  }


  # Query Active Directory for additional information

  $computer = Get-ADComputer -Filter { Name -eq $clientComputerName } -Properties
DistinguishedName, OperatingSystem, OperatingSystemVersion


  if ($computer) {

    $sessionInfo += [PSCustomObject]@{

      ClientComputerName = $clientComputerName

      ResolutionMethod = $resolutionMethod

      DistinguishedName = $computer.DistinguishedName

      OperatingSystem = $computer.OperatingSystem

      OperatingSystemVersion = $computer.OperatingSystemVersion

      ClientUserName = $session.ClientUserName

    }

  } else {

    # If the computer object is not found in Active Directory, include "Not Found" for the
additional details

    $sessionInfo += [PSCustomObject]@{

      ClientComputerName = $clientComputerName

      ResolutionMethod = $resolutionMethod

      DistinguishedName = "Not Found"
      OperatingSystem = "Not Found"

      OperatingSystemVersion = "Not Found"

      ClientUserName = $session.ClientUserName

    }

  }

}


# Define the output CSV file path

$csvFilePath = "SMB1SessionsWithADInfo.csv"


# Export session information to CSV file

$sessionInfo | Export-Csv -Path $csvFilePath -NoTypeInformation

Write-Host "SMB sessions information (less than SMB 2) exported to $csvFilePath"
```

Lingering legacy devices

I am often asked about the impact of disabling SMBv1 on domain controllers while there are still a few legacy Windows devices remaining. I am all for incremental gains where you can, but it is important to keep the following in mind.

- The legacy device will no longer be able to process group policies since the sysvol share will not be accessible. Previously applied GPOs will remain in effect but new and changed policies will not be read.
- SMB is required for a Windows device to join a domain. If you need to re-join a legacy device to the domain, it will be necessary to temporarily re-enable SMBv1 on a domain controller.
- Not every application is AD site aware. If you adopt a strategy of disabling SMBv1 in sites without legacy clients, don't be surprised if some of those clients attempt to connect to DCs outside their site. When that happens the client often gravitates to the PDCe.
- Some but not all remote named pipe sessions rely on SMB for transport.

- When you do encounter a dependency on SMBv1 the error message is often:
  - "The specified network name is no longer available."
  - "*Computer name* cannot be managed because the computer was not found on the network"
  - "The network path was not found"

Do's and Don'ts for disabling SMBv1 in a domain

Hopefully this information will help you take action if your domain controllers still have SMBv1 enabled. Just keep the following in mind and you will be well on your way.

- **Don't** wait until your legacy devices are all retired before auditing for SMBv1 use. There is a chance other devices and applications are also a source of SMBv1 connections. For a list of possible culprits check out this clearinghouse page that Ned maintains.
- **Don't** lose sight of the fact that legacy systems requiring SMBv1 are putting the entire domain at risk. The blast radius is not limited to the devices holding back the environment.
- **Do** ask for support from your CISO's office. This battle should not fall solely on the Active Directory team.
- **Do** leverage auditing to fully understand your environment. Knowledge is power.
- **Do** become proficient at network captures so you can investigate issues. Pro tip: Windows natively supports taking network captures using netsh. Those captures can be converted to the Wireshark format using ETL2PCAPNG.

Disclaimer

Updated Nov 08, 2024

Version 5.0
ADHardening
JerryDevore

\n\n\n\n \n \n \n \n \n \n \n \n \n\n \n \n \n \n \n \n \n \n\n\n \n \n \n \n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n\n\n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n \n\n\n\n \n \n \n \n \n \n \n

\n \n \n \n \n \n \n \n \n\n \n \n \n \n \n \n \n \n\n\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n\n\n \n \n \n \n \n \n \n \n\n \n \n \n
\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n \n
\n \n \n \n \n\n\n\n \n \n \n \n \n \n \n