What makes anonymous pipes?

sud0ru.ghost.io/what-makes-anonymous-pipes

Sud0Ru July 21, 2025

This post should be part of the IPC (Inter-Process Communication) series, but because it's an important topic and I can't find any post that discusses it, I decided to put it in a separate blog. The whole story started when I wanted to expose an anonymous pipe for my RPC server. I thought it was very straightforward, but it seems it's not.
I started to search online for some guides, but I was not lucky to find anything that helped me.

So today we will talk mainly about the ways you can use to make your pipe accessible through anonymous login, and we will see how Microsoft exposes such pipes like lsarpc, samr, and netlogon. So let's jump in.

Null Session

When we talk about anonymous login, we should mention the famous Windows null session. This type of anonymous login allows you to execute sensitive functions within several critical Windows RPC interfaces, which can be used to obtain sensitive information from a remote server.

Anonymous access is achieved by connecting to the IPC$ share using empty credentials. After establishing this connection, you can interact with named pipes that expose RPC interfaces.

Microsoft still supports null sessions today, but with many restrictions. Almost nothing significant can be done with a null session on modern systems. However, it is retained for compatibility with legacy systems that need to collect basic information from Windows servers.

Despite these limitations, administrators can still control null session behavior through two policies:

1- Network access: Restrict anonymous access to Named Pipes and Shares
When enabled, Windows only allows anonymous connections to shares and pipes explicitly whitelisted in policy. By default, on a domain controller, only a few system pipes such as \pipe\netlogon, \pipe\samr, and \pipe\lsarpc are included in this allowed list. All other pipes are blocked from null-session access unless they are manually added to the “Named Pipes that can be accessed anonymously” list.

2- Network access: Named Pipes that can be accessed anonymously
This policy defines the whitelist of pipe names for the setting above. If this list is empty (and the previous policy is enforced), no named pipe can be accessed via a null session. Adding a pipe name here permits anonymous users to access that pipe, assuming its security descriptor allows it.

Setting Up Our Testbed

To get a closer view of anonymous login over named pipes, we should set up a testbed that consists of an RPC server exposing an RPC interface through a named pipe over SMB, and a client that attempts to access this named pipe using anonymous login.

The server is straightforward: it exposes an interface through a named pipe called `\\pipe\\MyPipe` and there are no additional security measures implemented.

For the client, we need to call the RPC server **without using the standard Windows API**. This is because when you access an RPC server via the Windows API, it allows you to specify no authentication at the **RPC level** but when using named pipes as the endpoint, you **cannot control SMB-level authentication** (which is the transport layer in this case). The Windows API will always use your current session credentials to authenticate to SMB, regardless of the RPC-level authentication settings.

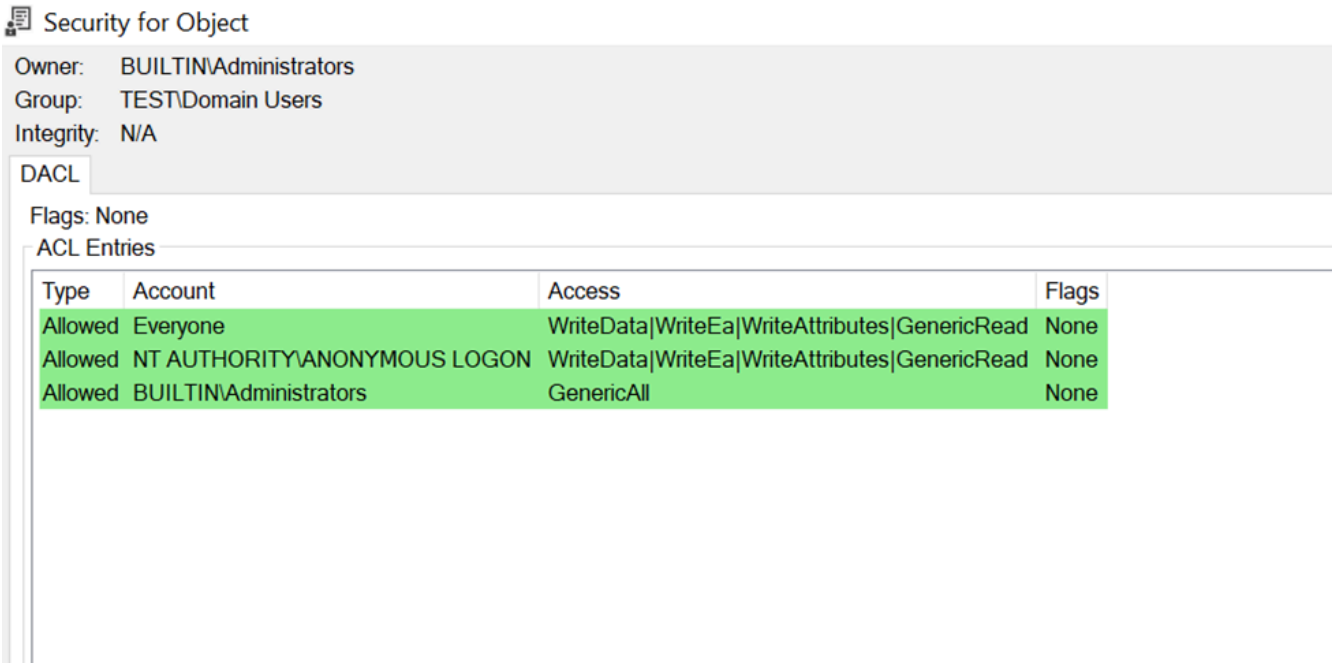
A simple solution is to implement your client using **Impacket and Python**.

RPC Server with Default Security Descriptor

In the server, we use the following call to register our named pipe:

```
status = RpcServerUseProtseqEp(
    (RPC_CSTR)"ncacn_np",
    RPC_C_PROTSEQ_MAX_REQS_DEFAULT,
    (RPC_CSTR)"\\pipe\\MyPipe",
    NULL);
```

The `RpcServerUseProtseqEp` function accepts a security descriptor as the last argument, but in our case, we don't provide one. As a result, the default DACL is applied as shown in the screenshot.



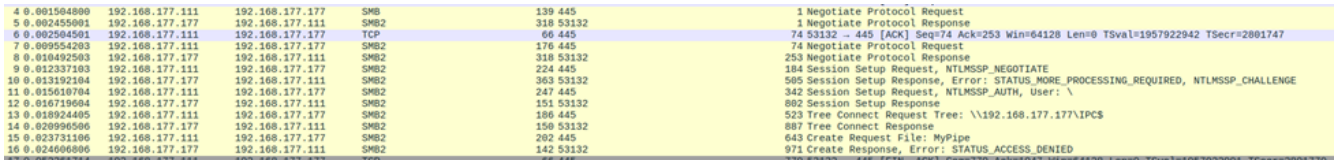
From the screenshot, we can see that **Anonymous Logon** is present in the DACL, meaning **anonymous access is allowed by default**.

Now, let's run the RPC server and connect using our RPC client. (The RPC server is running on **Windows Server 2022** with default configuration.)

After running the client, we receive an **Access Denied** error:

```
SMB SessionError: code: 0xc0000022 - STATUS_ACCESS_DENIED - {Access Denied}
A process has requested access to an object but has not been granted those access righ
```

And inside **Wireshark**, we observe the following traffic:

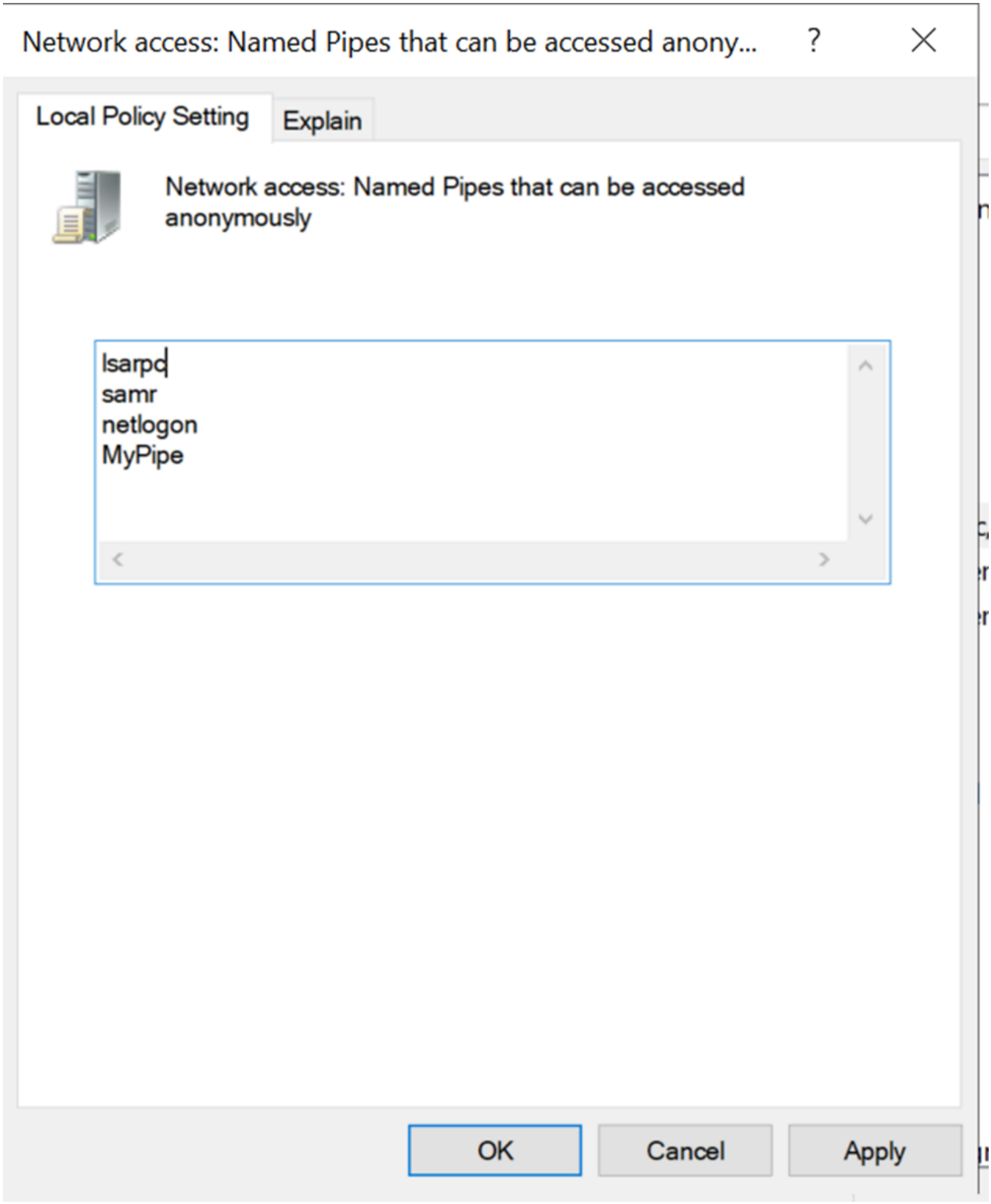


4	0.001504800	192.168.177.111	192.168.177.177	SMB	139 445	1 Negotiate Protocol Request
5	0.002455001	192.168.177.177	192.168.177.111	SMB2	318 53132	1 Negotiate Protocol Response
6	0.002504501	192.168.177.111	192.168.177.177	TCP	66 445	74 53132 → 445 [ACK] Seq=74 Ack=253 Win=64128 Len=0 TSval=1957922942 TSecr=2801747
7	0.009554203	192.168.177.111	192.168.177.177	SMB2	176 445	74 Negotiate Protocol Request
8	0.010492503	192.168.177.177	192.168.177.111	SMB2	318 53132	253 Negotiate Protocol Response
9	0.012337103	192.168.177.111	192.168.177.177	SMB2	224 445	184 Session Setup Request, NTLMSSP_NEGOTIATE
10	0.013192104	192.168.177.177	192.168.177.111	SMB2	363 53132	505 Session Setup Response, Error: STATUS_MORE_PROCESSING_REQUIRED, NTLMSSP_CHALLENGE
11	0.015610704	192.168.177.111	192.168.177.177	SMB2	247 445	342 Session Setup Request, NTLMSSP_AUTH, User: \
12	0.016719604	192.168.177.177	192.168.177.111	SMB2	151 53132	802 Session Setup Response
13	0.018924405	192.168.177.111	192.168.177.177	SMB2	186 445	523 Tree Connect Request Tree: \\192.168.177.177\IPC$
14	0.020996506	192.168.177.177	192.168.177.111	SMB2	150 53132	887 Tree Connect Response
15	0.023731106	192.168.177.111	192.168.177.177	SMB2	202 445	643 Create Request File: MyPipe
16	0.024606806	192.168.177.177	192.168.177.111	SMB2	142 53132	971 Create Response, Error: STATUS_ACCESS_DENIED

As shown, we successfully access `IPC$`, but when we try to access the named pipe (`MyPipe`), we get **Access Denied**, even though the pipe's DACL allows anonymous access.

However, if we recall the system policies, this is expected behavior. There is a **Windows policy** that controls anonymous access to named pipes.

So it now makes sense: if we add `MyPipe` to the policy list of **"Network access: Named Pipes that can be accessed anonymously"**, we should be able to access it without authentication.



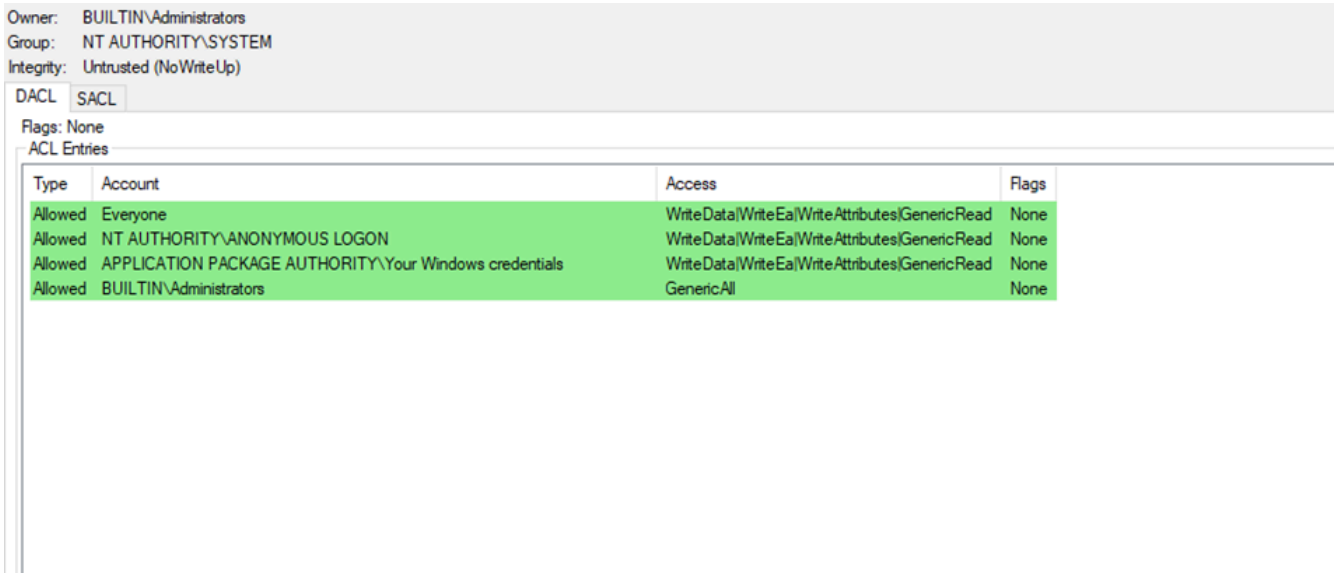
But even after adding the pipe to that list, we still receive **Access Denied**.
(*Note: The policy change only takes effect after a server reboot.*)

So now the question is: **what's the problem?**

Closer Look at Built-in Named Pipes

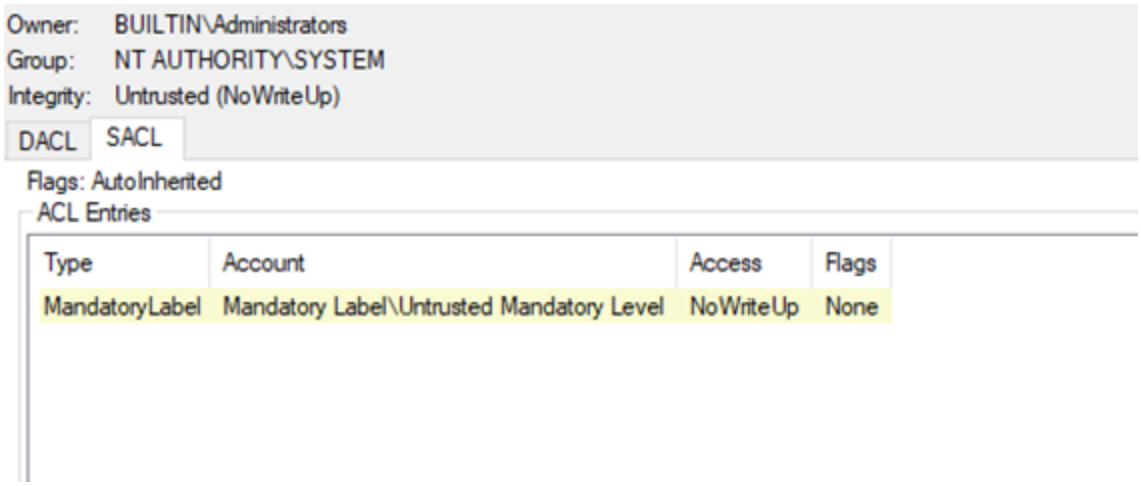
To answer why this behavior occurs, we need to take a closer look at how Microsoft secures named pipes that allow anonymous access (such as `lsarpc`, `samr`, and `netlogon`).

Let's check the **DACL** for `lsarpc`:



From the screenshot above, you can see that it's almost the same DACL as our custom pipe. So the issue is **not with the DACL**. However, as you can see, the Security Descriptor (**SD**) for `lsarpc` includes a **SACL**, while ours does not.

Now let's check the **SACL** in the screenshot below:



If you inspect the SACL, you'll see that it sets the **integrity level** to **"untrusted"**.

In Windows, *integrity* refers to the **Integrity Level (IL)** of a process, object, or security token. It's a Windows security mechanism that enforces isolation and prevents unauthorized access or privilege escalation—even if ACLs would otherwise permit it.

Integrity levels define how trustworthy an object or subject is. **Lower-integrity subjects** cannot modify or access **higher-integrity objects**.

There are five predefined integrity levels, as shown in the image below:

Value	Description	Symbol
0x0000	Untrusted level	SECURITY_MANDATORY_UNTRUSTED_RID
0x1000	Low integrity level	SECURITY_MANDATORY_LOW_RID
0x2000	Medium integrity level	SECURITY_MANDATORY_MEDIUM_RID
0x3000	High integrity level	SECURITY_MANDATORY_HIGH_RID
0x4000	System integrity level	SECURITY_MANDATORY_SYSTEM_RID

As you can see in the `lsarpc` named pipe, the integrity level is set to **untrusted**. But in our case, we haven't set any integrity level—so the default is **medium**.
Since **Anonymous Logon** has an **untrusted** mandatory level, it **cannot** access our pipe (which is medium integrity), but it **can** access the `lsarpc` pipe.

Here's where you can see the integrity level of the **Anonymous Logon token**:

```
PS C:\Users\Administrator> get-nttoken -Anonymous | fl Groups

Groups : {Mandatory Label\Untrusted Mandatory Level}


PS C:\Users\Administrator>
```

To allow anonymous access to our custom pipe, we need to set the **Security Descriptor (SD)** for the named pipe to include:

- Access for **Anonymous Logon**, and
- **Untrusted integrity level**

In the **SDDL** inside that code, I've tried to replicate the same security descriptor used by `lsarpc`.

```
const wchar_t* sddl = L"S:(ML;;NW;;;S-1-16-0)D:(A;;0x12019f;;;WD)(A;;0x12019B;;;AN)";
```

Now, if you run the server and then the client, you'll see that we've successfully reached the server using **anonymous access** (null session).

> Don't forget the earlier step where we added our pipe to the **"Named Pipes that can be accessed anonymously"** policy list!

Two Other Ways to Enable Anonymous Access

I found two other ways to enable anonymous access to a named pipe. This time, you don't need to set a low integrity level on the pipe. Instead, you grant the Anonymous Logon account more privileges.

I strongly advise against doing this on any production server—use it for experimentation only, especially the second method.

There is a policy called **"Let Everyone permissions apply to anonymous users."**
This policy is disabled by default. Enabling it causes the **Everyone** SID to be added to the anonymous login token, effectively granting anonymous users all permissions assigned to the "Everyone" group.

In addition, this policy raises the integrity level of the anonymous user to **Medium**, allowing it to access the named pipe using the default security descriptor (SD).

Token SIDs before enabling the policy:

Index	SID	Meaning
[0]	S-1-0-0	Null SID
[1]	S-1-5-2	Network — indicates network logon
[2]	S-1-5-15	This Organization — for org-based filtering
[3]	S-1-5-64-10	NTLM Authentication — used for NTLM or anonymous network logon
[4]	S-1-16-0	**Untrusted Integrity Level**

Token SIDs after enabling the policy:

Index	SID	Meaning
[0]	S-1-0-0	Null SID

Index	SID	Meaning
[1]	S-1-1-0	**Everyone** — all users
[2]	S-1-5-2	Network — indicates network logon
[3]	S-1-5-15	This Organization — for org-based filtering
[4]	S-1-5-64-10	NTLM Authentication — used for NTLM or anonymous network logon
[5]	S-1-16-8192	**Medium Integrity Level**

This second method is more unconventional but still works. Again, **do not use this in production.**

There is a policy called **“Impersonate a client after authentication.”**
It contains a list of accounts that are granted the **SeImpersonatePrivilege**. By adding the **Anonymous Logon** account to this list, you raise the integrity level of its token to **High**, which is sufficient to access the pipe.

After modifying this policy, the group SIDs inside the token remain mostly the same, but now include a new integrity level:

S-1-16-12288 — High Integrity Level

This suggests that **Windows assigns a high integrity level to any account added to this list**, even for regular domain users. If you test this with a normal domain user, you’ll notice that its integrity level is elevated from Medium to High.

Before I end this post, here's an important observation:

Regardless of the setting for **“Network access: Restrict anonymous access to Named Pipes and Shares,”**
if **“Named Pipes that can be accessed anonymously”** is empty, you will receive **Access Denied** when trying to access the IPC$ share itself.

This means the **initial SMB session to IPC$ fails** for null sessions—even before attempting to open a pipe. However, if you add any value to that list, the IPC$ check passes, and the usual access check for the named pipe proceeds as normal.

References:

NtObjectManager 1.1.32

This module adds a provider and cmdlets to access the NT object manager namespace.

PowerShell Gallery Home

Mandatory Integrity Control - Win32 apps

Provides a mechanism for controlling access to securable objects.

Microsoft Learnalvinashcraft