

Windows Inter Process Communication A Deep Dive Beyond the Surface

 sud0ru.ghost.io/windows-inter-process-communication-a-deep-dive-beyond-the-surface-part-6

Sud0Ru

August 17, 2025





Welcome to a new part of the IPC series, and the third part of RPC security. In the previous parts, we talked about RPC authentication levels, RPC bindings, and how to secure your RPC interface.

Today, we will continue our discussion on RPC security, but this time we'll focus on securing the endpoint and the system policies that are enforced to protect the RPC server.

As with the previous parts, I want to start by mentioning the resources behind this work. This post is based on my own research, along with:

- Microsoft's official documentation (MSDN),
- The excellent work by @0xcsandker on [offensive Windows IPC](#),
- [James Forshaw's blog post](#),
- And [Ben Barnea's detailed write-up on the Akamai blog](#).

So let's jump in

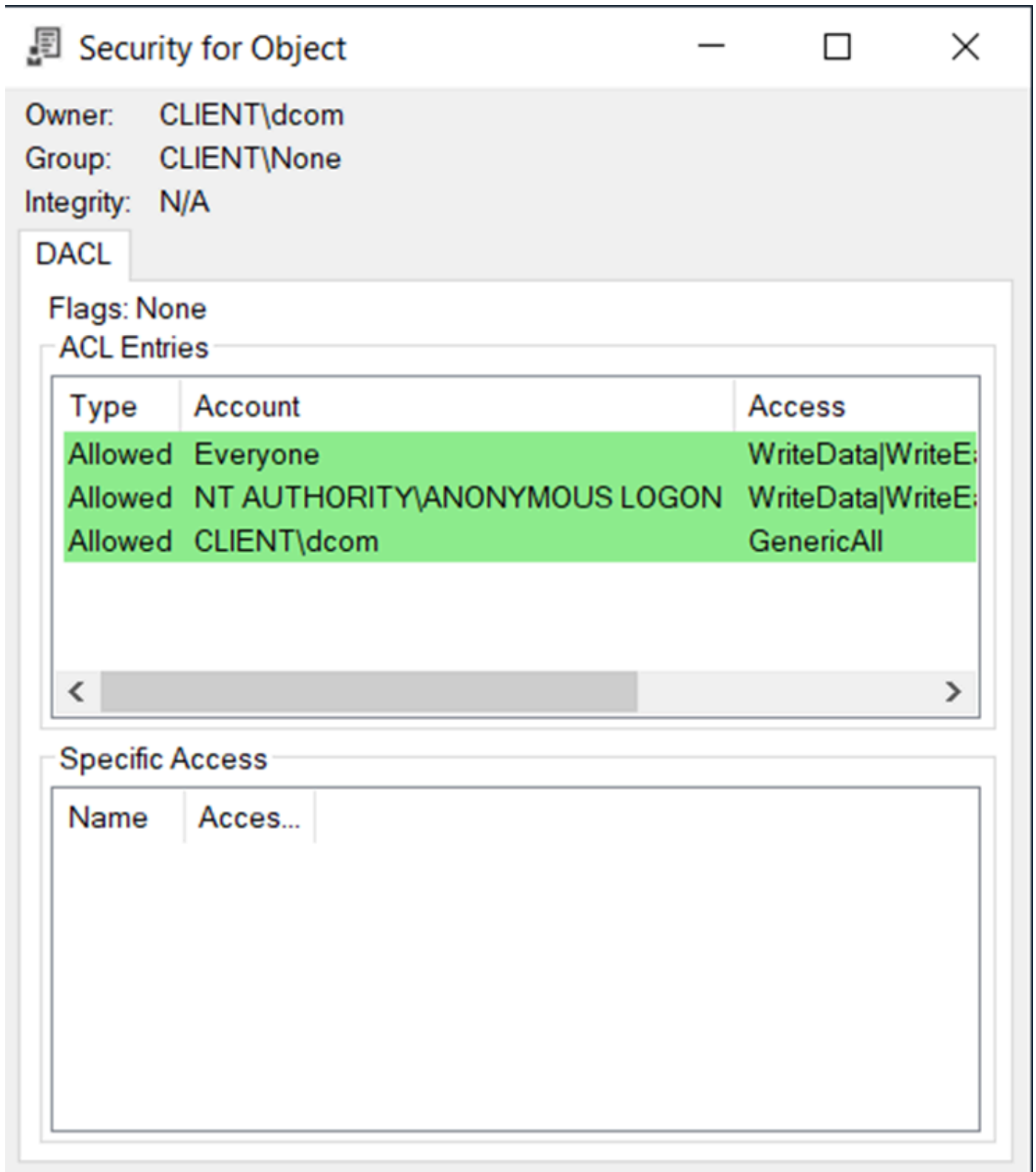
Securing the Endpoint

We have different types of endpoints, such as TCP ports or named pipes. As mentioned before, to register an endpoint we can use the `RpcServerUseProtseqEp` function, defined as follows:

```
RPC_STATUS RpcServerUseProtseqEp(
    RPC_CSTR      Protseq,
    unsigned int  MaxCalls,
    RPC_CSTR      Endpoint,
    void          *SecurityDescriptor
);
```

This function takes the protocol sequence and the related endpoint. The last argument is the security descriptor, which allows you to set a security descriptor for the endpoint (for named pipes and ALPC) but not for TCP. (As we mentioned in the previous part, TCP is an unauthenticated protocol.)

For named pipes, if you don't set any security descriptor, access will be granted to the users shown in the screenshot below.



As you can see, if we register a named pipe as an endpoint without specifying a security descriptor, **Everyone**, **Anonymous Logon**, and the user account under which the RPC server process runs will all get access to this named pipe.

If you are interested in anonymous named pipes, you can check my blog post here:

[What Makes Anonymous Pipes?](#)

Endpoint Multiplexing

Microsoft recommends **not** relying on endpoint security and the security descriptor of the call, since it is only maintained for backward compatibility. You can read more here:

[Do not use endpoint security](#).

The reasoning behind this is something called **endpoint multiplexing**. This means that if the RPC server registers multiple endpoints (e.g., Named Pipe, ALPC port), then each interface registered in the process can be accessed through any of these endpoints.

This design exposes a serious security hole. For example, let's assume your RPC server exposes two interfaces:

- The first interface is exposed through an **ALPC port**, which has a restrictive security descriptor and can only be accessed locally.
- The second interface is exposed through a **Named Pipe**, which is accessible over the network and has no security descriptor applied.

Because of endpoint multiplexing, the first interface (intended to be restricted to local access) can still be reached through the Named Pipe endpoint, completely bypassing the security restrictions on the ALPC port.

The Policy

The last piece of RPC security is the RPC runtime policy “**Restrict Unauthenticated RPC Clients.**”

This policy controls how the RPC runtime handles the security of connected clients. It has three possible values:

1. **Not Configured**
2. **Enabled**
3. **Disabled**

When **Enabled**, there are three sub-options:

1. **Authenticated:** The RPC runtime blocks access to TCP clients that have not authenticated. (There are some exceptions, which we'll cover shortly.)
2. **Authenticated without exceptions:** All unauthenticated connections are blocked.
3. **None:** All RPC clients are allowed to connect to RPC servers running on the machine.

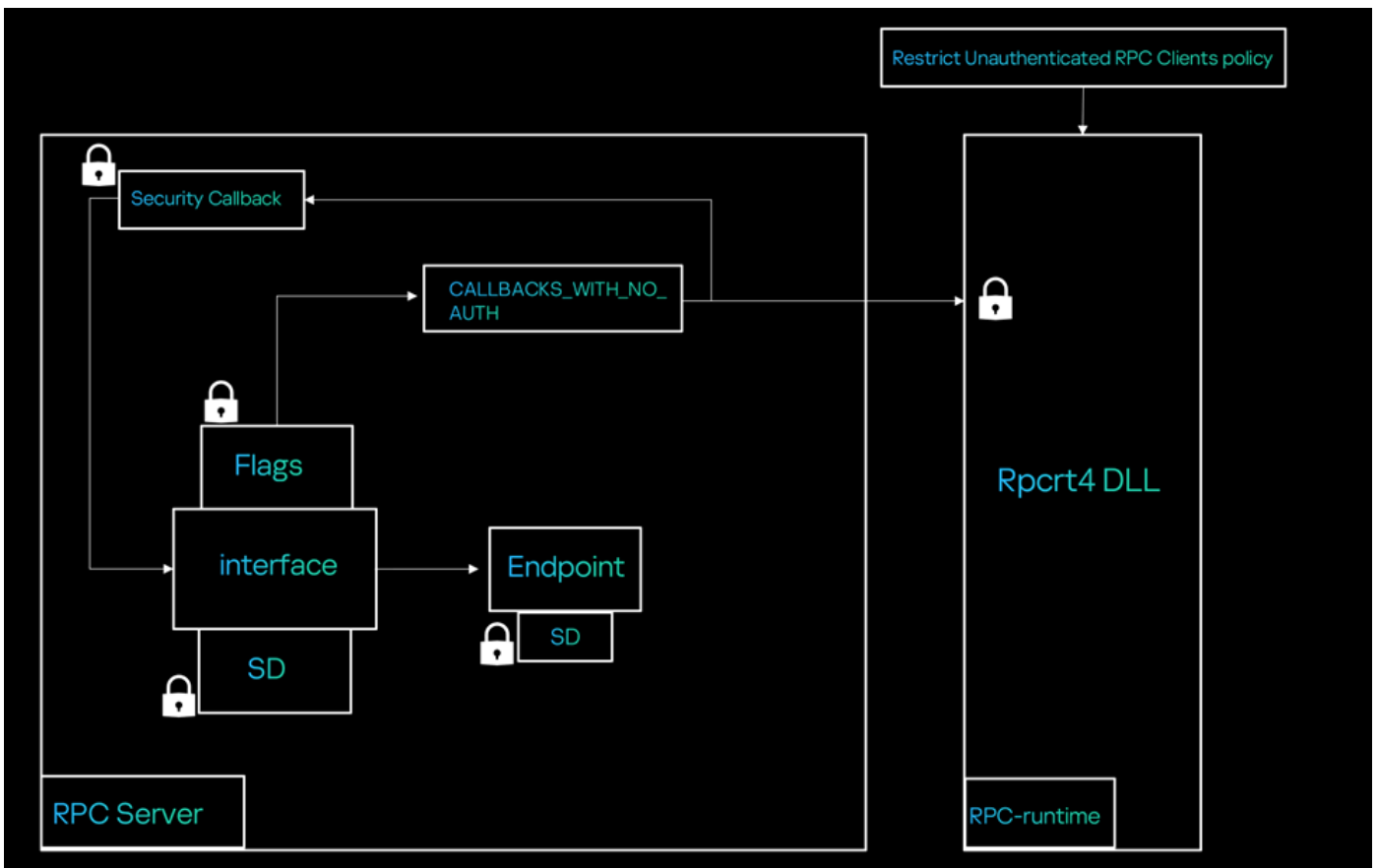
By default, the policy is “**Not Configured.**” However, the behavior of this default differs between Windows Servers and Windows Clients:

- On **Windows Server**, “Not Configured” is equivalent to **None**, meaning all clients (even unauthenticated ones) are allowed to connect.
- On **Windows Workstations** (e.g., Windows 10), “Not Configured” is equivalent to **Authenticated**, meaning clients must authenticate before connecting to the RPC server.

Now, even if the policy is set to **Authenticated**, if the RPC interface is registered with the **`RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH`** flag (and uses a security callback), then the policy will not take effect. In this case, authentication handling is delegated to the application itself.

To fully prevent unauthenticated clients from accessing interfaces registered with this flag, the policy must be set to **Authenticated without exceptions**. From this, we can conclude that the “exceptions” are specifically those interfaces registered with the `RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH` flag.

The following figure illustrates how this policy links to the RPC server:



Access Matrix and Access Check

Now that we've covered RPC security, the **access matrix** mentioned here:

Offensive Windows IPC - Part 2: RPC

provides a clear overview of which clients can connect to which servers, along with the specific error codes returned in each case.

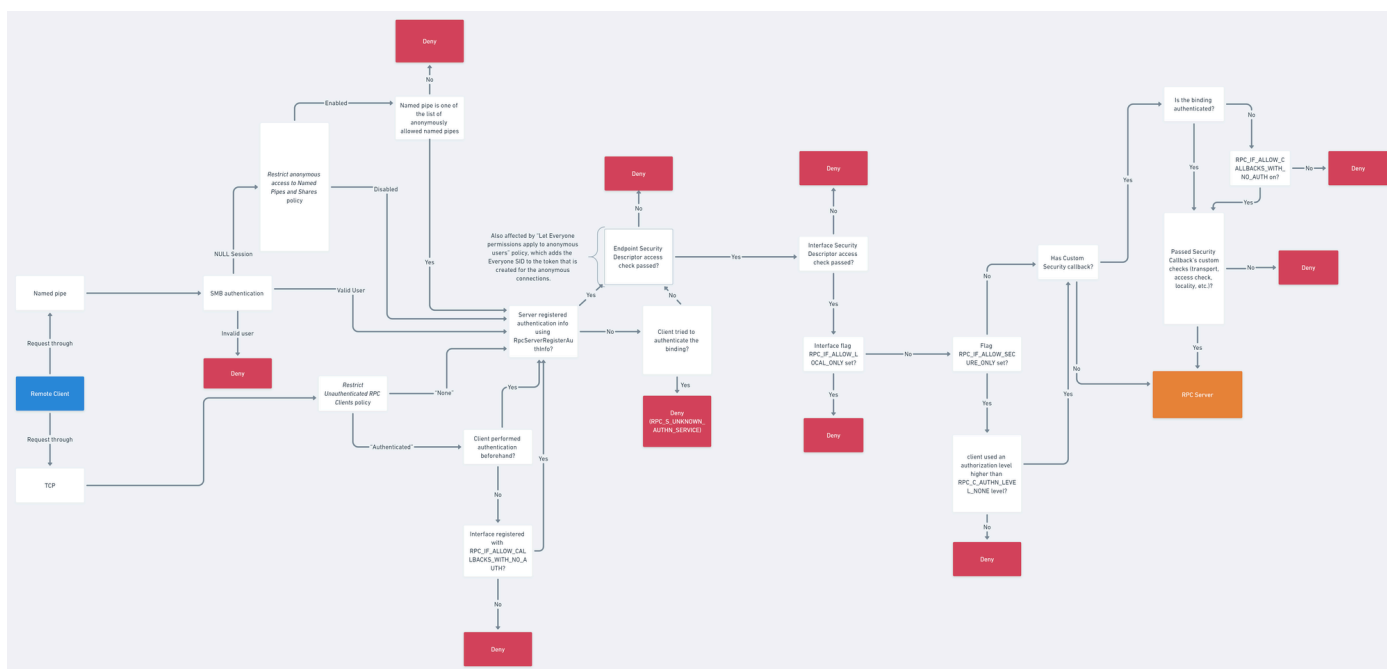
Client/Server		Unauthenticated Binding NoFlags, NoSecurityCallback	Unauthenticated Binding NoFlags, SecurityCallback	Unauthenticated Binding Flags', NoSecurityCallback	Unauthenticated Binding Flags', SecurityCallback	Authenticated Binding NoFlags, NoSecurityCallback	Authenticated Binding NoFlags, SecurityCallback	Authenticated Binding Flags', NoSecurityCallback	Authenticated Binding Flags', SecurityCallback
Unauthenticated Binding		Success	Error 5 (Access Denied)	Success	Success	Success	Error 5 (Access Denied)	Success	Success
Authenticated Binding	NoQOS	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Success	Success	Success	Success
Authenticated Binding	QOS	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Error 1747 (RPC_S_UNKNOWN_A UTHN_SERVICE)	Success	Success	Success	Success

QOS Quality of Service
Flags¹ RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH

In addition to the access matrix, there is also a very useful **flowchart** from Akamai's research:

[MSRPC Security Flowchart](#)

This chart helps you determine the access checks that are performed on a client before it reaches the server. Using it, you can map all the pieces we've discussed so far and see how they stack on top of each other.



Personally, I prefer to extend this flow by adding one more box for the **“Restrict Unauthenticated RPC Clients” policy** when it is set to **“Authenticated without exceptions.”** in addition to “Authenticated” and “None.”

- If the client performs authentication, it can move on to the next level of checks.
- If the client does **not** authenticate, it is immediately denied access without even checking for the `RPC_IF_ALLOW_CALLBACKS_WITH_NO_AUTH` flag.

I think with this part we've finished covering all aspects of RPC security. I hope the explanations were clear and not too complicated.

Thanks for reading, and stay tuned for the next part of the RPC series!