Windows Inter Process Communication A Deep Dive Beyond the Surface - Part 1

sud0ru.ghost.io/windows-inter-process-communication-a-deep-dive-beyond-the-surface-part-1

Sud0Ru April 28, 2025

Apr 28, 2025 4 min read

Windows Inter-Process Communication (IPC) is one of the most complex technologies in the Windows operating system. It consists of multiple layers that can work together or operate independently, depending on the usage context.

For example, you can use **RPC (Remote Procedure Call)** to invoke functions on a remote machine when you need simple function-level communication. On the other hand, if your task requires more advanced features—such as object lifecycle management,

security, or event handling—you can use **DCOM (Distributed Component Object Model)**. DCOM is built on top of RPC and extends its capabilities by enabling object-oriented communication over the network.

I know this is a complex topic, which is exactly why I started this series. It has several goals—some already achieved, and others still in progress.

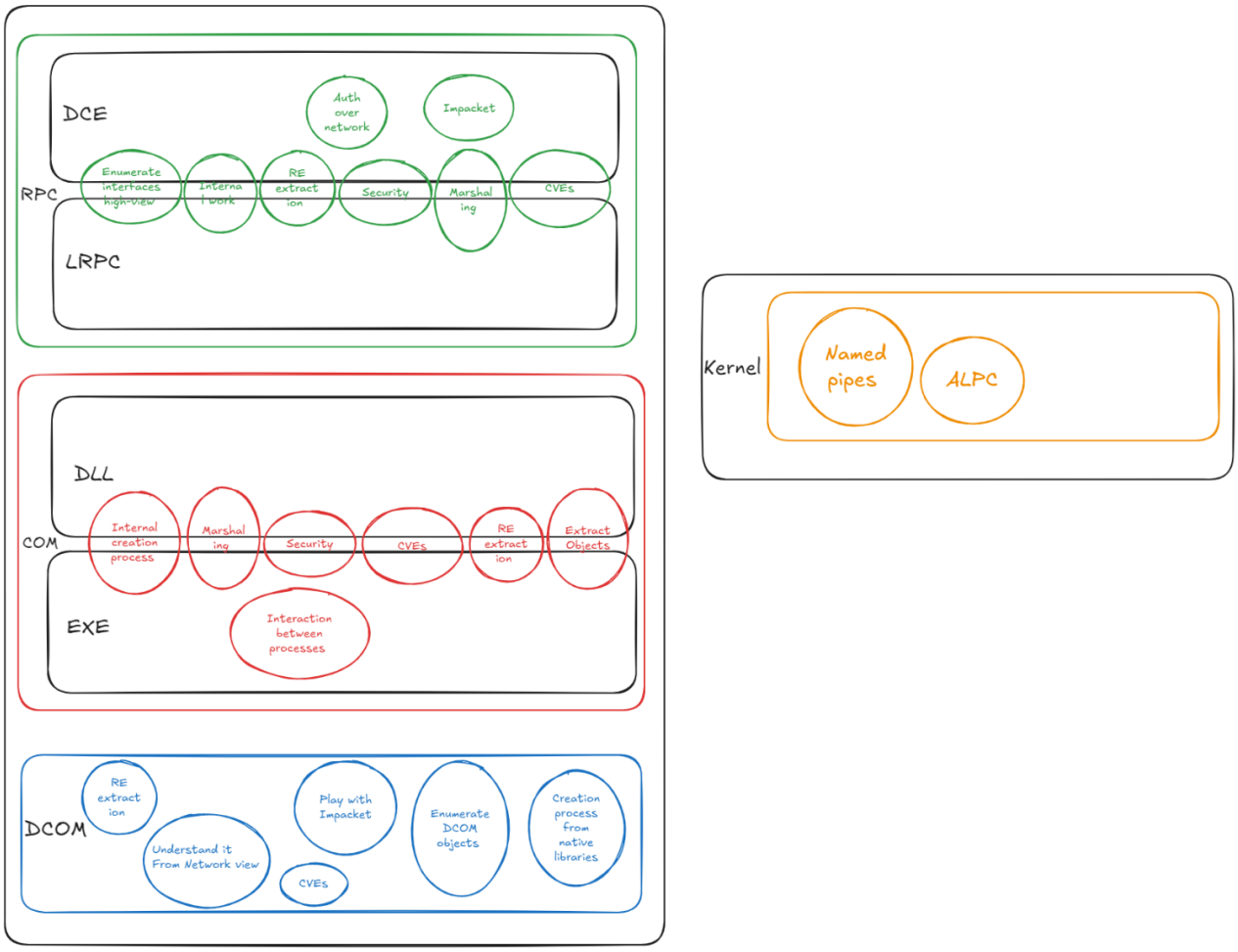
The main purpose of this series is to simplify this technology and make it more accessible to everyone. The second goal is to dive deeper into the internal workings of these systems to accomplish a range of objectives, which I'll discuss it later.

This has been a major research project, with far more depth under the surface than you might imagine. I started working on it over a year ago—despite having many other responsibilities, I still dedicated significant time and resources to it.

A lot has been accomplished so far, and I've made solid progress, but I believe it will take about another year to fully complete the research.

In this part, I'll share the **first draft of the roadmap** for this research that I initially created. We'll walk through each stage together, discussing where we currently stand and what the ultimate goals are. The idea is to provide a clear overview of the journey—highlighting both what has been explored so far and the exciting work still ahead

RoadMap:



IPC roadmap first draft

The diagram above was the first draft I created to outline what I needed to do and how deep I should go into each area. Since then, I've continuously refined and expanded it—adding more goals and expected outcomes for each part of the research.

I won't be sharing the final version just yet, but instead, I'll briefly walk through each section

The roadmap consists of four main layers, starting from high-level technologies like **DCOM**, down to lower-level mechanisms such as **kernel-mode named pipes**. For each section, I initially outlined key areas to research using simple circles representing basic ideas or entry points into each technology. These ideas have evolved and become more detailed as the research progressed.

RPC

I started with **RPC** because it forms the foundation for both **COM** and **DCOM**—the more I invested in understanding RPC, the more results I could achieve in those higher-level technologies. So it made perfect sense to begin here.

RPC consists of two main parts: **LRPC** and **DCERPC**. Technically, they work in a very similar way—but the difference lies in how the communication happens:

- **DCE** refers to making RPC calls **over the network**, which requires a transport layer to handle the network communication.
- **LRPC** (where the "L" stands for *Local*) is used for communication between processes **on the same machine**. It typically uses specialized transport mechanisms like **ALPC** (Advanced Local Procedure Call).

*(It's worth noting that these terms aren't always used strictly due to historical decisions within Microsoft—but for the sake of our discussion, we'll use them to distinguish between **network-based** and **local** RPC calls.)*

Inside the RPC journey, I chose to begin with **DCERPC**—the network-level part—by experimenting with **Impacket** and its scripts. My goal was to understand this technology from a very high-level perspective. I began by studying **how interfaces are called** and getting a grip on **RPC security mechanisms**.

This effort paid off when I achieved one of my first goals: I discovered a **new method for collecting domain information without authentication**, including retrieving domain users. I've already described this technique in detail [here].

Although LRPC and DCERPC share the same internal logic and structure, the key difference is the **transport layer**. So once you understand one, you're most of the way toward understanding the other.

Next, we'll explore **how to create RPC servers and clients using native Windows libraries**, diving deep into their implementation and security features.

Then we'll examine the **low-level code** that underpins RPC communication—code that's **automatically generated by a special compiler** when you define your interface.

After that, we'll step into the world of **reverse engineering** to understand what the server and client look like once compiled—and how we can extract and analyze RPC interfaces through reverse engineering techniques.

We'll also uncover how applications interact with the **RPC runtime library**, and dive into an important concept called **marshaling**—the process of packaging and unpackaging data during client-server communication.

Once we've built this deep technical foundation, we'll explore some of the **most well-known attacks and vulnerabilities** related to RPC. And who knows—maybe we'll even discover **new techniques or previously unknown vulnerabilities** along the way. 🙂

COM/DCOM

When it comes to **COM and DCOM**, the concept is very similar to **LRPC and DCERPC**—**COM** is used for **local communication**, while **DCOM** is used when the communication happens **over the network**.

The roadmap for COM/DCOM will largely follow the **same steps as we took for RPC**, but with key differences, such as:

- Understanding **COM object lifecycles**
- Exploring how these objects are **defined, registered, and instantiated**
- Learning how to **extract object definitions**
- Figuring out how to **interact with undocumented COM objects**

I'm not going into too much detail here, because this part of the roadmap is still in progress. It'll be better to explain everything clearly once we reach the section dedicated to **COM and DCOM**.

ALPC / Named Pipes

ALPC (Advanced Local Procedure Call) and **Named Pipes** are **kernel-level technologies** that power much of Windows' **inter-process communication (IPC)**—especially in local scenarios where **network sockets** are not used.

In this section, we'll go even deeper—looking into how to:

- Interact directly with these **kernel objects**
- Understand **how they work under the hood**

As with previous sections, we'll follow the **same structured approach**: starting from high-level use, diving into technical implementation, and finishing with **security implications and potential vulnerabilities**.

I know this introduction might sound a bit complicated and full of new terms, but don't worry—I'll walk you through everything step by step. I'll make sure things become clearer and easier to understand as we go.

So, see you in the next section, where we'll start diving into **RPC**!