

In-process COM still starts in LPAC when lpacCom is absent

The capability check that guards COM runs inside the sandboxed process itself. Official startup is refused; a known in-process server can still be loaded. No out-of-process server was reached.

A reading of part 1 by Denis Nagayuk and Francisco Dominguez, published 13 August 2026.

The check runs on the calling thread, before any object exists

CoInitializeEx and RoInitialize share one internal function, `combase!_CoInitialize`. On a Less Privileged AppContainer it calls `RtlCheckTokenCapability` with the `lpacCom` capability. The access check runs in the kernel; the request to perform it comes from the sandboxed process.

Two capabilities, two gates

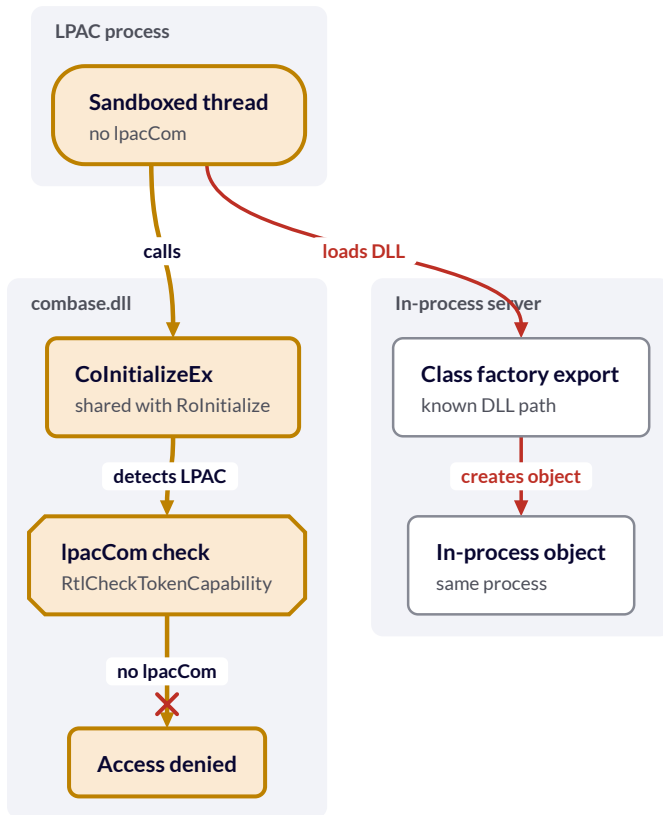
`lpacCom` gates this startup call. `registryRead` gates most registry keys, including COM class registration. This part assumes `registryRead` is present and sets it aside.

Figure 1

lpacCom stops official startup, not a DLL the process already knows

Call path inside one LPAC process with no lpacCom capability

— What lpacCom stops — Call — Blocked — At risk



Source: Hunt & Hackett, "On COM/WinRT initialization, apartments, and lpacCom capability bypasses, Part 1", 13 August 2026.

Startup fails closed, and four activation calls never begin

Without lpacCom the first call returns access denied. Ignoring that error does not leave a half-initialized thread: every activation call that follows reports that COM was never initialized.

```
CoInitializeEx(...) // access denied: the gate
RoInitialize(...) // access denied: same gate
CoCreateInstance(...) // CO_E_NOTINITIALIZED
CoGetClassObject(...) // CO_E_NOTINITIALIZED
RoActivateInstance(...) // CO_E_NOTINITIALIZED
RoGetActivationFactory(...) // CO_E_NOTINITIALIZED
```

The two highlighted calls are the gate. The other four fail only because the gate already refused the thread, which is a weaker signal: any other way of getting an object past initialization will not hit that error.

COM initialization is self-sufficient. It doesn't require any external entity (such as the kernel) to maintain or generate state.

Denis Nagayuk and Francisco Dominguez

Both routes the authors tried stop at the process boundary

The post tests two ways around the refusal. Neither reached an out-of-process server. They differ in what they need inside the process, and in how brittle the result is.

Whether the LPAC boundary held, by route

Route	In-process	Out-of-process	Condition the authors state
Official startup	● Holds	● Holds	First call returns access denied
Manual class factory	◆ Fails	● Holds	Known DLL. Many objects run; some expect an apartment and then break
Forced check result	◆ Fails	● Holds	Needs registryRead so class lookup can succeed

“Holds” means the authors observed access denied. For out-of-process activation they record the denial and do not show the check that enforced it.

Three claims stay outside the evidence of this part

Out-of-process denial

Activation stayed access denied with registryRead present. The enforcing check is not in this part.

No registry lookup

Class keys require registryRead. Instantiation without that capability is deferred.

A built-in skip

The authors say an existing feature skips the check with no patch. They do not name it.

For reviewers

Do not accept a missing lpacCom capability as proof that COM is unavailable

In this part the in-process boundary failed by two routes, and the out-of-process boundary held only as an observed denial. Review registryRead as a separate gate.

Method

Every result above is an observation reported by Nagayuk and Dominguez. This note does not re-run the tests and adds no measurements.

