Devolutions

Extending the Microsoft RDP Client with API Hooking

IRDP ON WINDOWS IS BY FAR THE MOST POPULAR.

Out of all the protocols and platforms supported by Remote Desktop Manager, RDP on Windows is by far the most popular. While Microsoft offers their official remote desktop client on all platforms except Linux, the only RDP client interface exposed for third-party integration is on Windows. FreeRDP is the only suitable cross-platform RDP client, but it will never be on par with the official Windows RDP client (MSTSC). The reality is that in order to deliver features customers want, we need the flexibility of FreeRDP (open-source), but inside the Microsoft RDP client (closed-source).

Building our own doors

While the Microsoft RDP client is great, it does not expose everything required to implement all the feature requests that we have. When hitting a wall, one can choose to stop, or try building new doors in places that didn't have them. This is what we've been working on with the Devolutions MsRdpEx project: a set of new «doors» we've opened using the Microsoft Detours Library for API hooking. It builds on top of the public RDP ActiveX interface to expose internal components and properties not accessible otherwise. An early RDP API Hooking feature is already available in RDM 2022.2.

What's API Hooking?

API hooking is scarier than it sounds: it modifies a program in-memory to intercept a function call and redirect it to a «hook» function which you control. For instance, by hooking the LoadLibrary function, it is possible to tweak its behaviour such that loading «mstscax.dll» loads «MsRdpEx.dll» instead. In other words, you can «lie» to the application by altering what the function actually does. Not all functions can easily be hooked: some are more subject to change than others, or simply out-of-reach.

Reverse Engineering

It's closed-source, right? So that not only means no documentation, but no source code to dig through. You can learn more about reverse engineering Microsoft binaries in our «Finding Secret RDP Registry Keys Using IDA Free» blog post. This is by far the most tedious and time consuming process, even if the end result is just flipping a bit somewhere. For instance, our unofficial MSRDC integration broke recently with a mysterious error code 3334, and it took two full days of intense debugging to find out a new internal property had to be set to «true.»

Extended RDP Options

What kind of option in the RDP client requires API hooking? Here's a quick list:

- **UserSpecifiedServerName:** An internal option required to specify the server name for TLS and Kerberos validation and used alongside the RD Gateway protocol, but not exposed for regular RDP connections like the ones we do for Devolutions Gateway.

- **KDCProxyName:** A publicly-documented option to inject a KDC proxy server dynamically to allow Kerberos, but it is only applied for RD Gateway connections.
- **UsingSavedCreds:** An internal option incorrectly set if a password is set at connection time, despite not being saved, breaking credential injection when the «Always prompt for password upon connection» group policy is enabled.
- **DisableUDPTransport:** believe it or not, this internal option can only be set through registry keys, preventing fine-grained control from a connection manager.

The first two options, **UserSpecifiedServerName** and **KDCProxyName**, are essential to faciliate Kerberos when outside of the corporate network, yet they are restricted to the RD Gateway protocol specifically. We have a KDC proxy feature coming up in the Devolutions Gateway to leverage those, but we would have preferred not to take such a long detour with API hooking to make it possible in the first place.

Azure Virtual Desktop

Many users have requested Azure Virtual Desktop support, and we are currently halfway through it. If things go well, we should have a first AVD integration in RDM 2022.3, with improvements added later on. Here is a list of challenges we're facing:

- MSRDC is required for AVD, but does not officially expose an API for integration
- MSRDC requires RDP signed server-side, preventing client-side settings management
- The AVD protocol extensions are not documented, unlike the regular RDP protocol
- The AVD control calls for the RDP webfeed made with Azure are also undocumented

The first step is already done: we've managed to integrate MSRDC for regular RDP connections, in both embedded and external modes. MSRDC ships an out-of-box DLL (rdclientax.dll) that exposes the same RDP ActiveX interface as the MSTSC built-in DLL (mstscax.dll).

AVD adds about 10 new RDP file options, with corresponding internal properties for the ActiveX. Just figuring out what they were and what they meant was a challenge, but we've at least managed to properly import/export them in RDM for now.

A problematic signature

It was fine until we hit a first major blocker: AVD RDP files are signed server-side and published over a special web feed that only MSRDC knows how to fetch. However, MSRDC won't make AVD connections with unsigned RDP files, and you can't modify RDP options client-side without breaking the signature. Houston, we have a problem.

When Remote Desktop Manager launches MSTSC or MSRDC in external mode, it generates a temporary .RDP file with all the options from the connection entry. If you edit the RDP connection entry in RDM, you'll get a different RDP file. The only way around it is to create a connection entry linked to a local, unmodified RDP file, but then you still can't modify it.

Why not sign the files locally? We've considered it, but this requires a certificate from a known, trusted authority. API hooking strategies to use a valid signature from a self-signed certificate were also limited. It took about two weeks of effort to finally figure out a way to bypass the RDP file signature check for AVD connections, and get a first AVD connection launched with MSRDC from RDM.

Are we there yet?

We're making slow, but steady progress. AVD connections have an additional authentication context with Azure, and managing it nicely will be a project in itself. Injection of the regular RDP credentials is also broken, so we'll likely need to figure out a different way to do it. The user interface for the AVD-specific options is not done yet, and properly naming and structuring them is difficult when they're not documented. The current focus is to make it work with MSRDC as an external program first, since AVD connections with MSRDC in embedded mode comes with a different set of challenges.

Conclusion

API hooking is great, but it should be considered a last resort solution. Ideally, most of these extended options would have been publicly exposed officially such that we didn't need to spend all of this time reverse engineering and hooking APIs. Are you an Azure Virtual Desktop customer? If not, are you interested in using MSRDC instead of MSTSC? Since both of these cases are not officially supported, we would appreciate if you could send feedback to Microsoft about it!