Devolutions

Finding Secret RDP Registry Keys Using IDA Free

ANYONE CAN PERFORM SIMPLE REVERSE ENGINEERING TASKS

IDA is a state-of-the-art reverse engineering tool commonly used in the software industry to analyze closed-source binaries. While free or cheaper alternatives like Ghidra are gaining in popularity, they are no match for IDA's decompiler in terms of accuracy and maturity. Luckily for us, IDA Free now includes an x64 decompiler, which makes reversing possible ***without assembly language skills.***

The goal of this blog post is to show how anyone can perform simple reverse engineering tasks using nothing more than logical deduction and the right set of tools. Instead of focusing on the final result, the steps include detailed screenshots and comments to show the complete thought process.

Prerequisites

Download and install IDA Free. Please note that this specific edition of IDA is not suitable for commercial use, but since this guide is meant as an introduction, it shouldn't be an issue. While I have access to IDA Pro at work, all the screenshots were taken using IDA Free to avoid confusion.

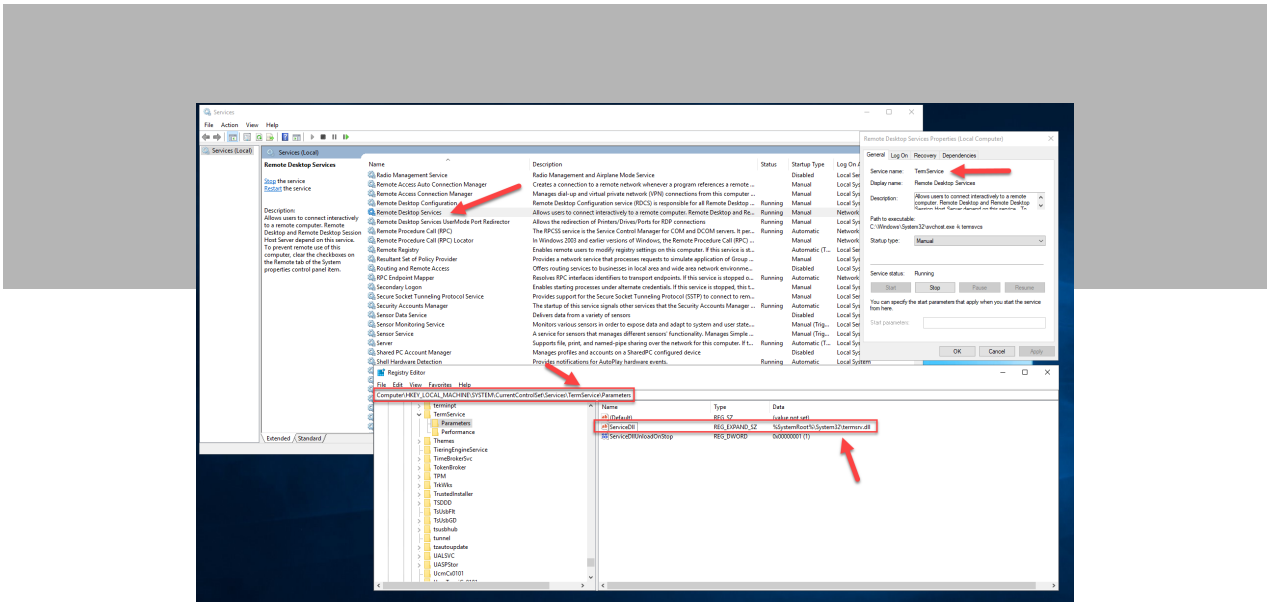
A Windows machine with RDP enabled is recommended to follow all the steps, but it is possible to do some of the tasks from another platform if necessary. I have used a clean Windows Server 2019 virtual machine for this project.

Finding a Goal

This may sound obvious, but rather than poke around random binaries, it is a good thing to start with a goal in mind. In this article, our goal will be to **identify secret registry keys affecting the RDP H.264 video encoder**.

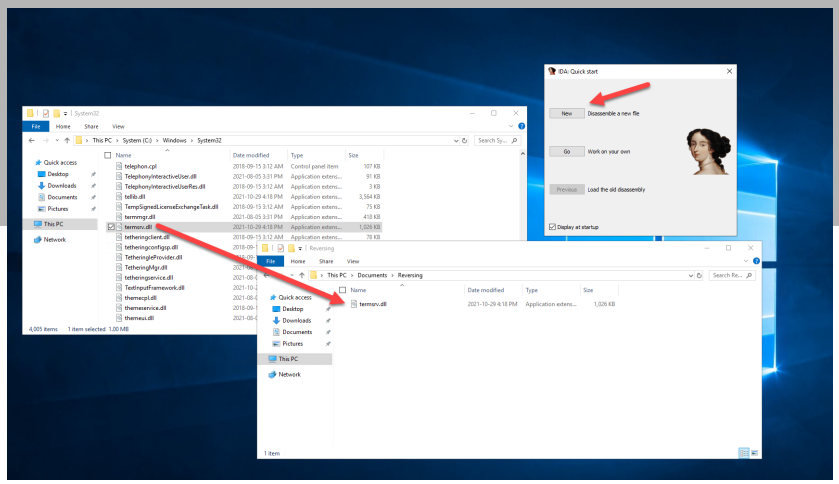
How you choose the goal doesn't matter, but I picked this one because I am familiar with H.264 in RDP through my work in the FreeRDP project.

The Microsoft RDP server is handled by the **Remote Desktop Services** system service, which is called **TermService** and uses termsrv.dll as its entry point:

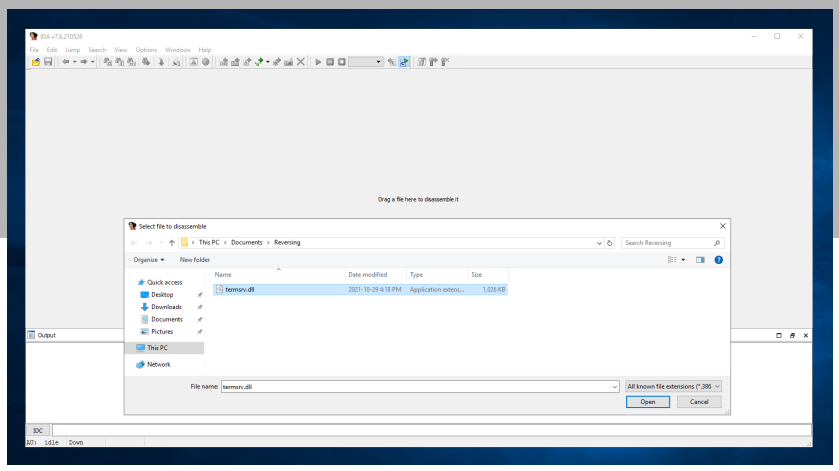


Getting Started

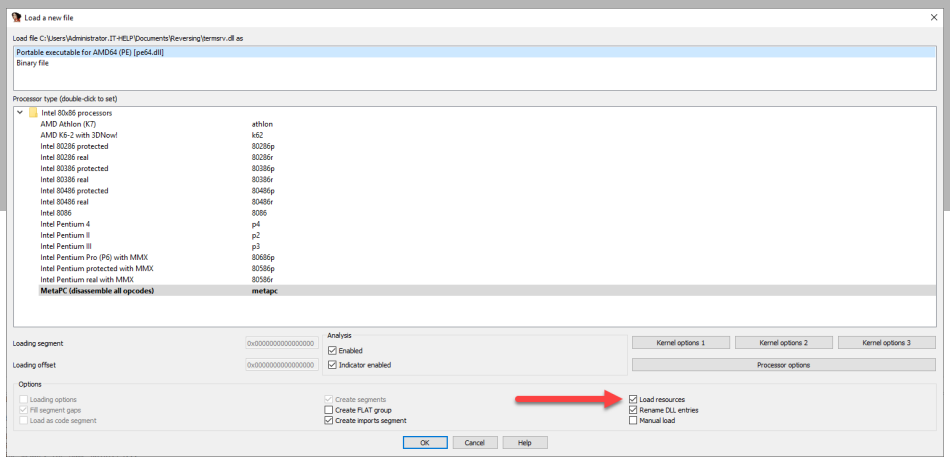
Create a new directory called «Reversing» in «My Documents» and copy «termsrv.dll» from «C:\Windows\System32» into it. Launch IDA and click **New** in the **Quick Start** dialog:



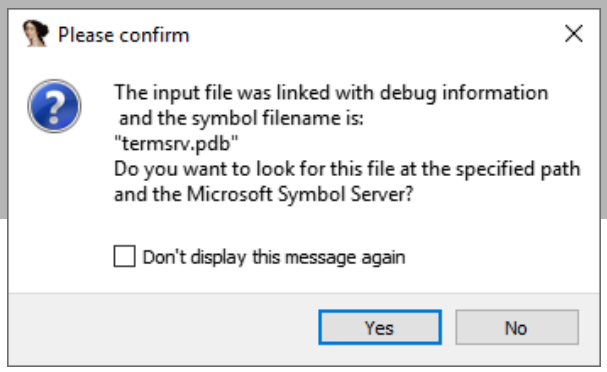
Browse to the «Reversing» directory, select «termsrv.dll» and then click **Open**:



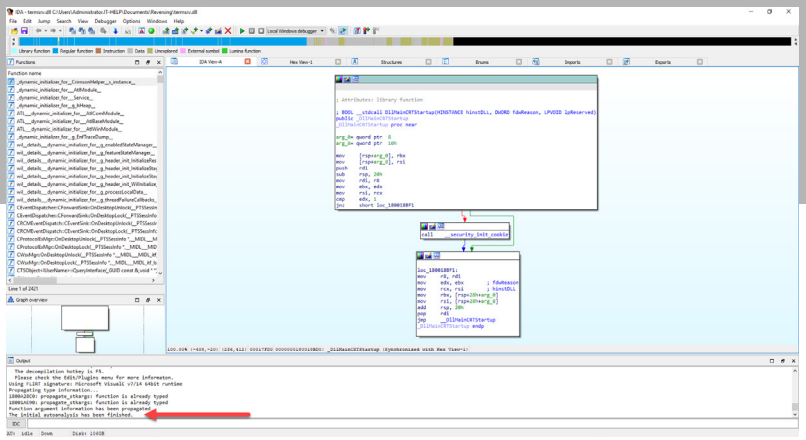
In the **Load a new file** dialog, check «Load resources» and then click **OK**. This option is not mandatory, but resource segments can sometimes contain valuable information.



Confirm that you want to load the matching debug symbols by clicking **Yes** when prompted:



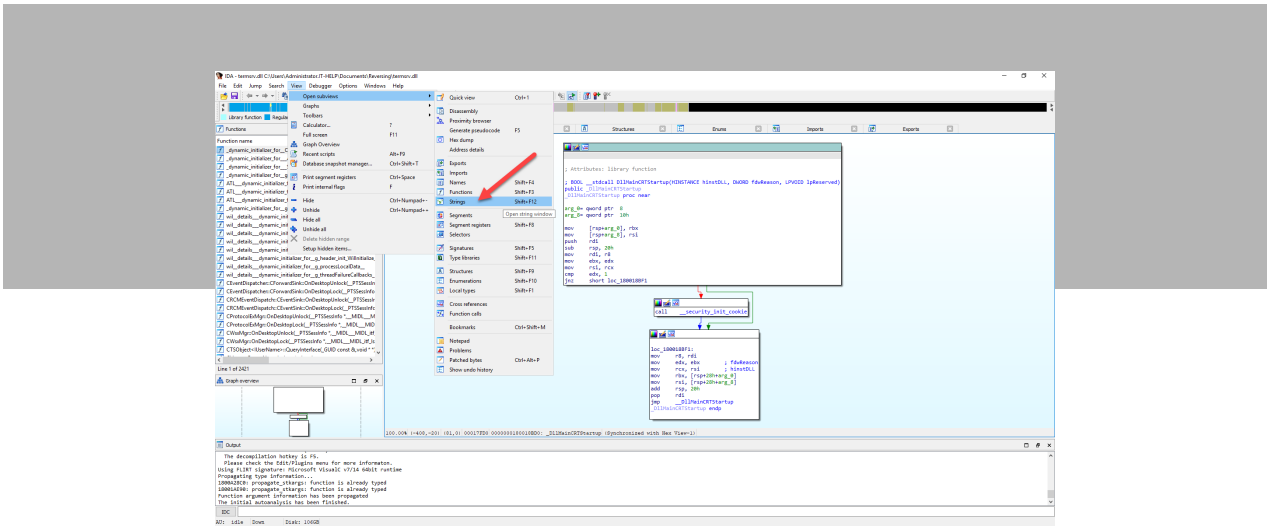
Once the main IDA interface appears, wait for the **Output** window at the bottom to say «the initial autoanalysis has been finished». This can take a few minutes and varies a lot depending on the size of the binaries.



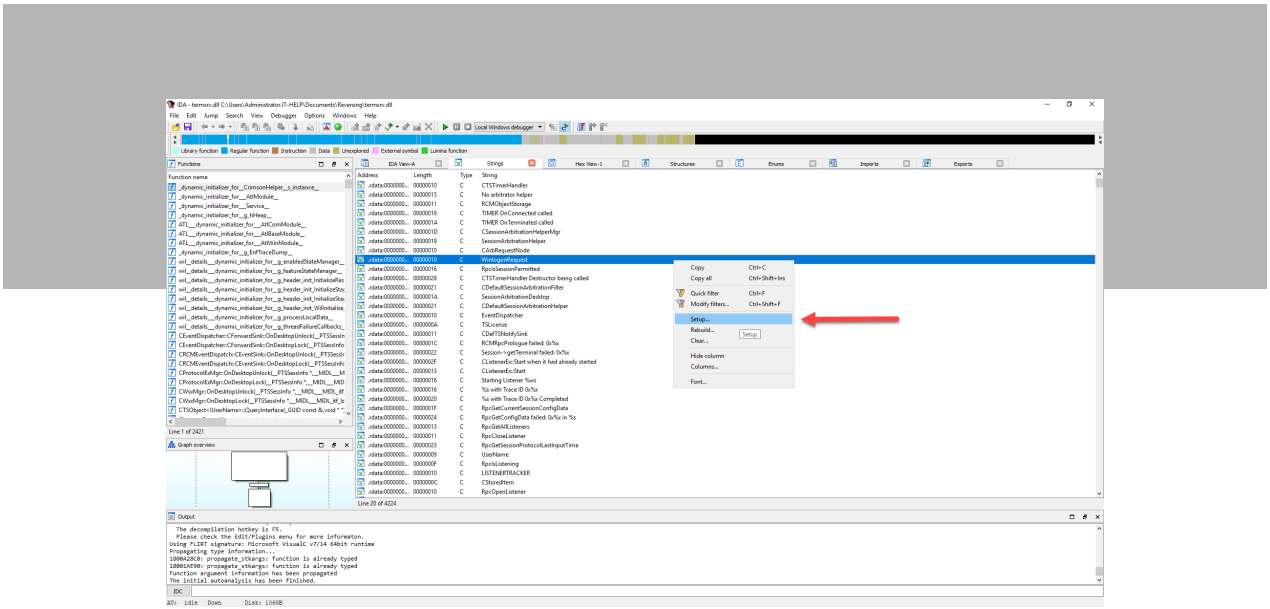
That's it! You are now ready to begin investigating the contents of «termsrv.dll». Refer to this procedure to create IDA projects for new binaries in the future.

No Strings Attached

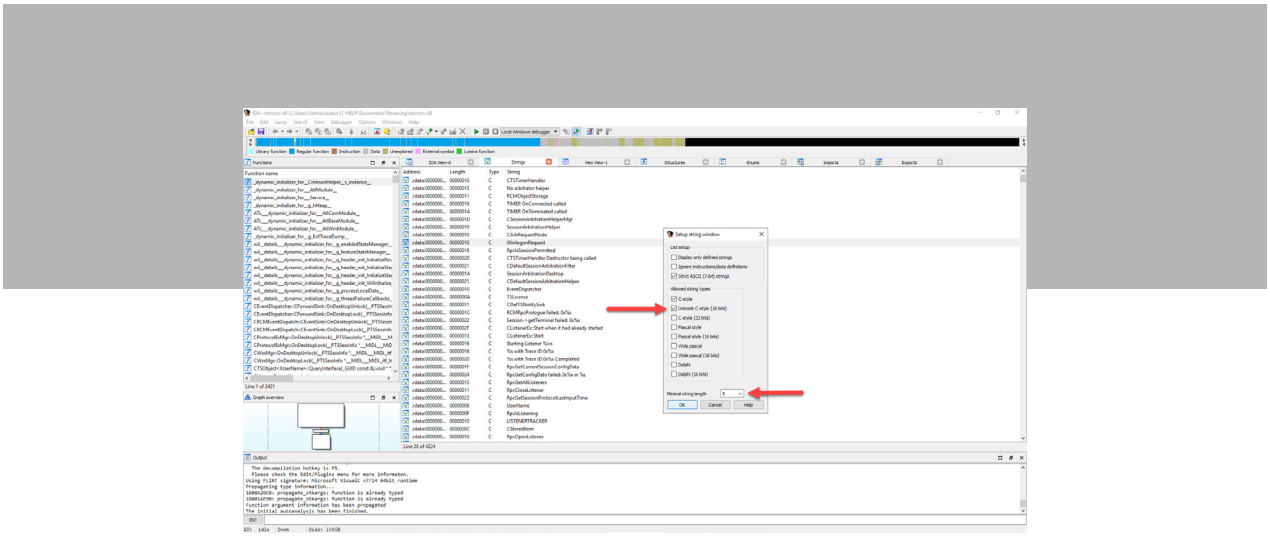
Strings is the most useful view in IDA, but it is unfortunately not present in the default configuration. On the **View** menu, navigate to **Open subviews** then select **Strings**:



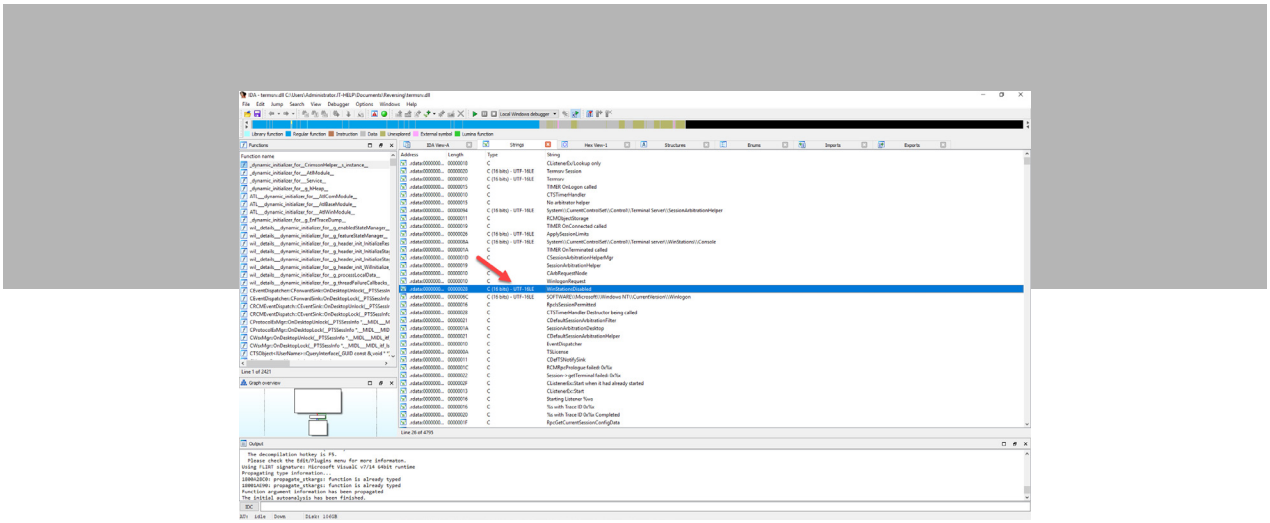
The new **Strings** view will be added, and show textual strings found anywhere inside the binary. Right-click anywhere inside the view to open the contextual menu, then select **Setup...**:



Check **Unicode C-style (16 bits)** to enable Unicode UTF-16 literals that are frequently used on Windows. The minimal string length can be changed if desired (5 is the default). Click **OK** to apply the changes:



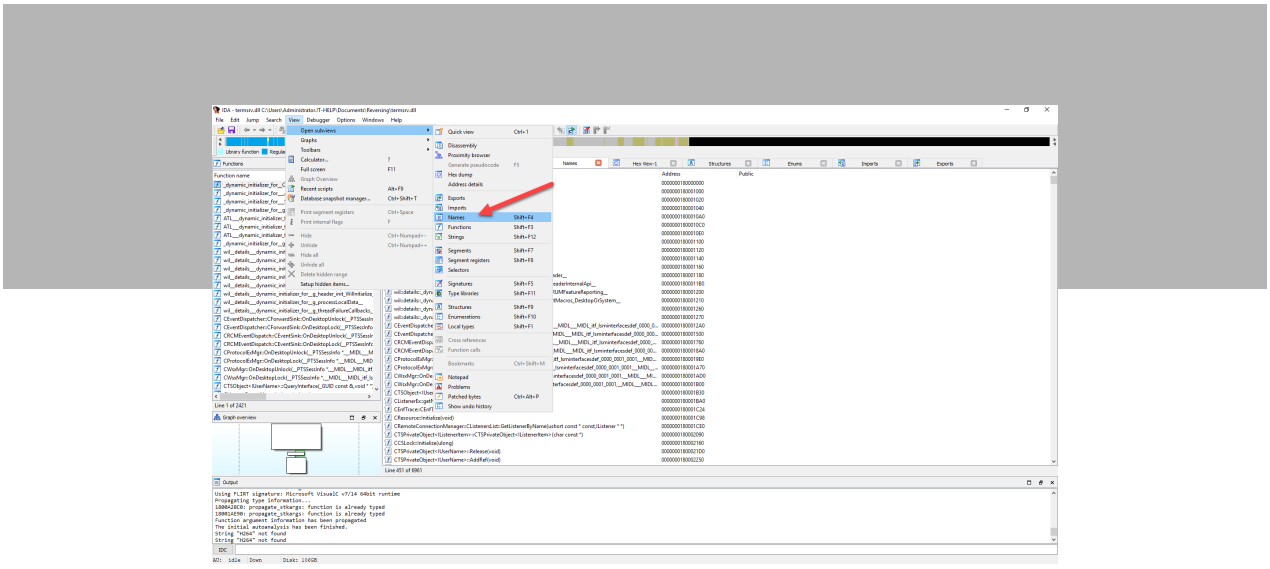
The **Strings** view now shows a lot more strings than before. It took me years before I realized that IDA didn't look for UTF-16 strings by default, and it would have saved me a lot of time!



Press **Alt+T**, then type «H264» and press **Enter** to search for that specific substring in all of the strings in termsrv.dll. The **Output** window at the bottom should say «String H264 not found».

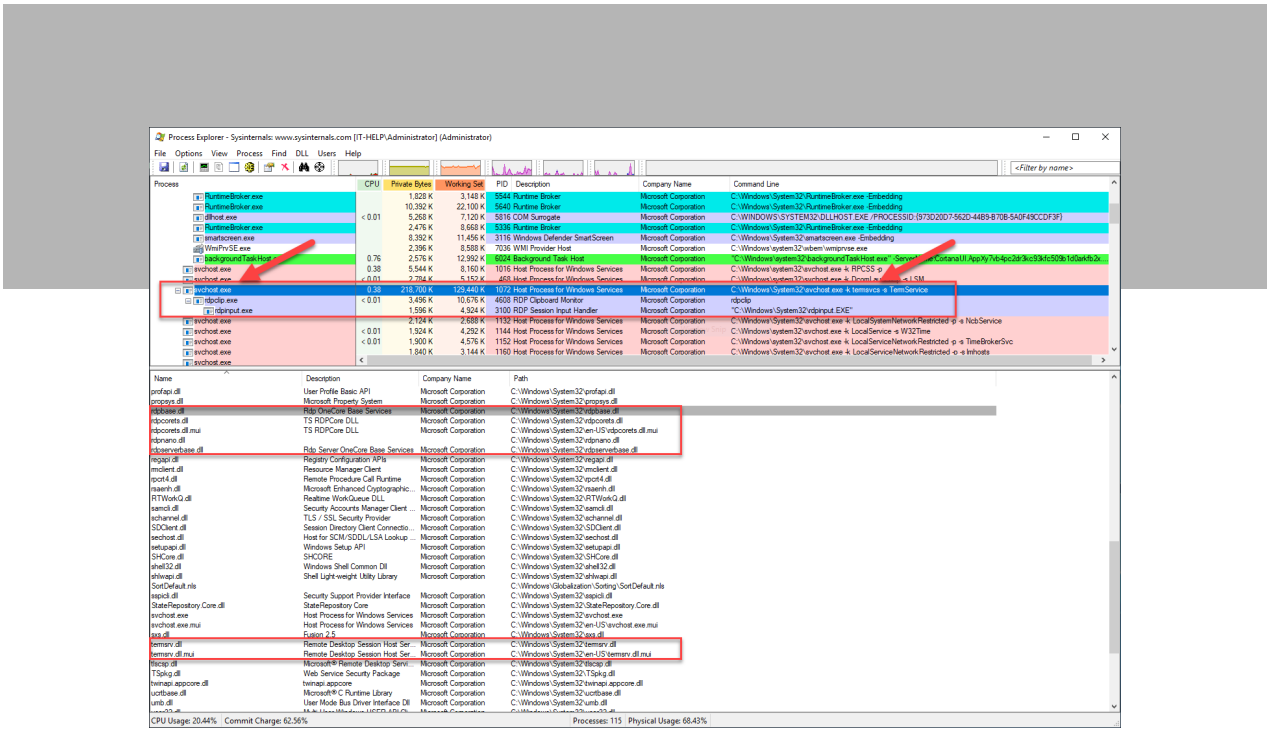
What's Your Name?

So «H264» as a string is not present in termsrv.dll, but maybe it can be found in function names or symbols? On the **View** menu, navigate to **Open subviews** then select **Names**:



The **Names** view functions in the same way as the **Strings** view. Press **Alt+T**, then type «H264» and press **Enter** to search for the keyword in all symbol names. Unfortunately, we still have no results for «H264», indicating that termsrv.dll may not be the right place to look.

Is this the end of the road? Not at all! Reversing is a lot like fishing, where it can take time to catch one fish but it's exciting when it finally happens. Let's try a different type of bait: Process Explorer from the Sysinternals Suite. Download and launch the tool on the RDP server, then look for a process called «svchost.exe» with the «rdpclip.exe» and «rdpinput.exe» subprocesses:

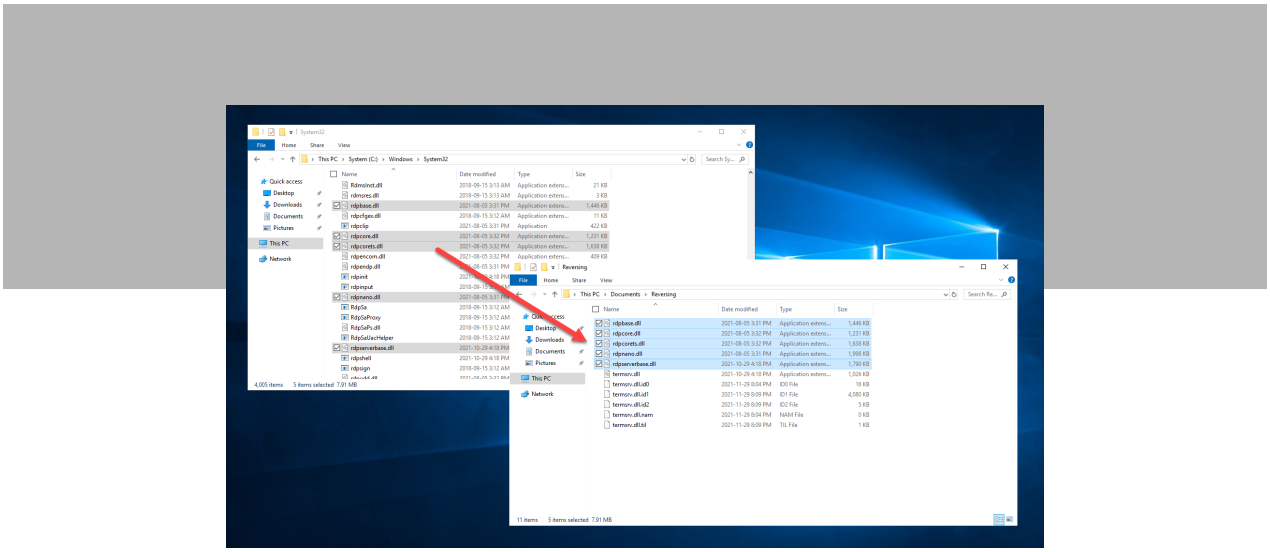


Windows system services all run within «service host» (svchost) processes, which makes it harder to tell them apart. The Process Explorer **Command Line** column is quite helpful, otherwise the Get-CimInstance PowerShell cmdlet can be used to find the process id for a given service name:

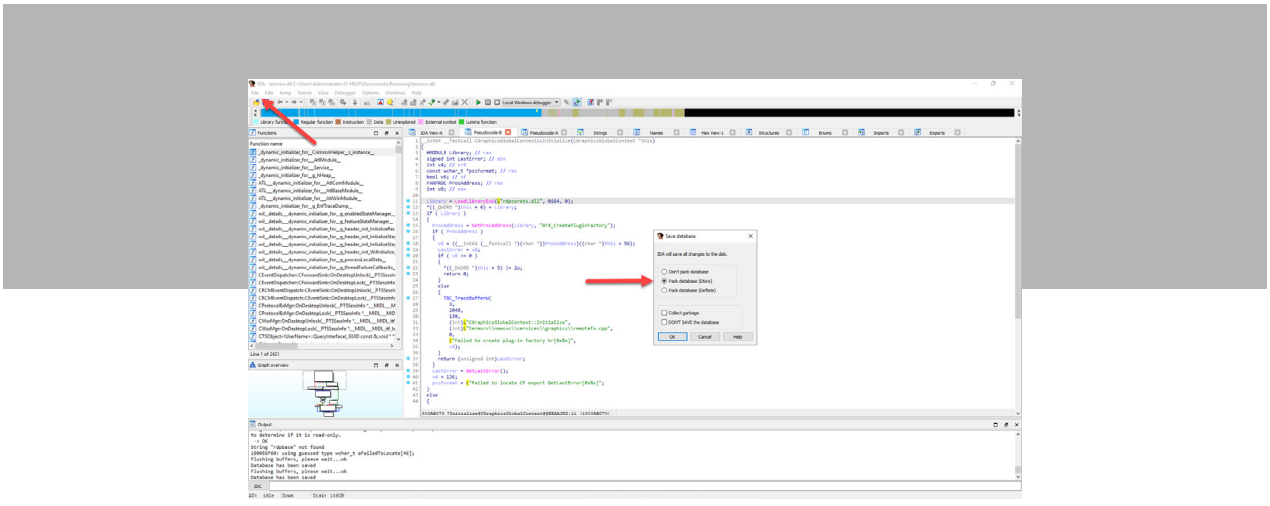
```
Get-CimInstance -Class Win32_Service -Filter "Name LIKE 'TermService'"

ProcessId Name        StartMode State   Status ExitCode
--------- ----        --------- -----   ------ --------
1072      TermService Manual    Running OK     0
```

What matters is that we can see a list of DLLs loaded in the RDP server alongside termsrv.dll. Let's expand our search area to include a few DLLs that begin with «rdp» and copy them into our «Reversing» project directory:



While it is possible to have multiple IDA instances open at the same time, I would not recommend it, as it is easy to get lost. On the **File** menu, select **Close**. In the **Save database** dialog, select **Pack database (Store)** and then click **OK**:



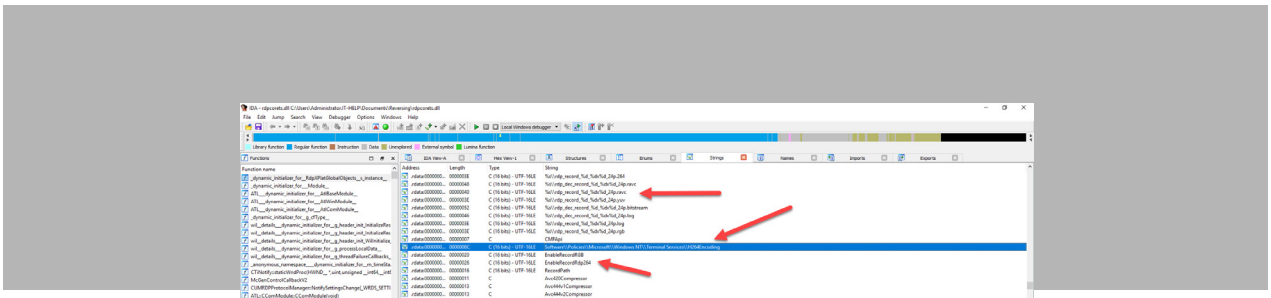
We now have new files to look at, which I have decided to investigate in the following order:

- rdpserverbase.dll
- rdpbase.dll
- rdpcorets.dll
- rdpcore.dll
- rdpnano.dll

For each of those files, I repeat the entire process:

- Create new IDA project
- Search for «H264» string

The string «H264» is found a few times in rdpserverbase.dll and rdpbase.dll, but it's nothing significant. We finally hit the jackpot when searching for «H264» in rdpcorets.dll:



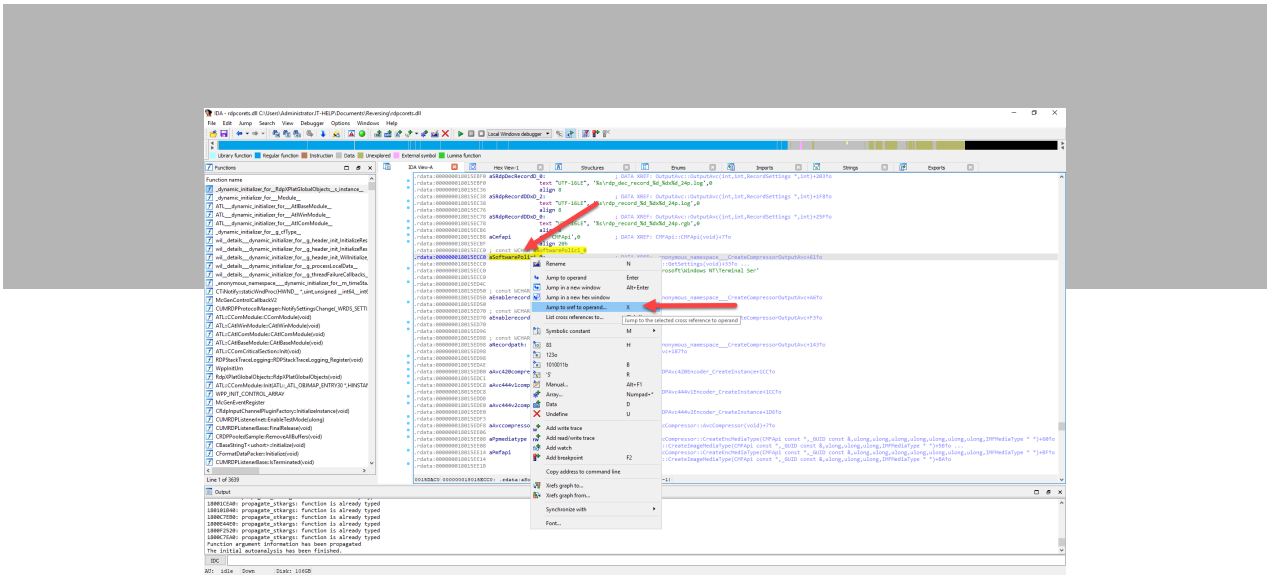
Not only do we see a lot of references to «H264», but some strings clearly hint at registry keys. We decide to focus on rdpcorets.dll and close all other IDA projects.

Searching for Functions

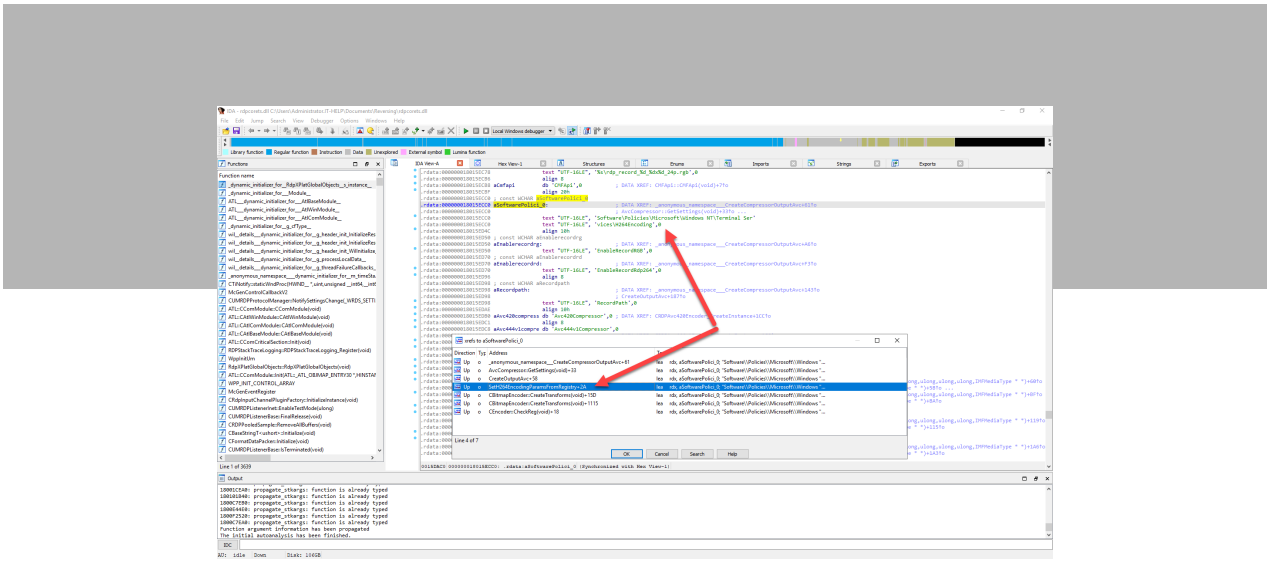
Now that we've found the strings, we can start searching for the functions that use them. Double-click on the string that appears to be a registry key path named «H264Encoding»:

Software\Policies\Microsoft\Windows NT\Terminal Services\H264Encoding

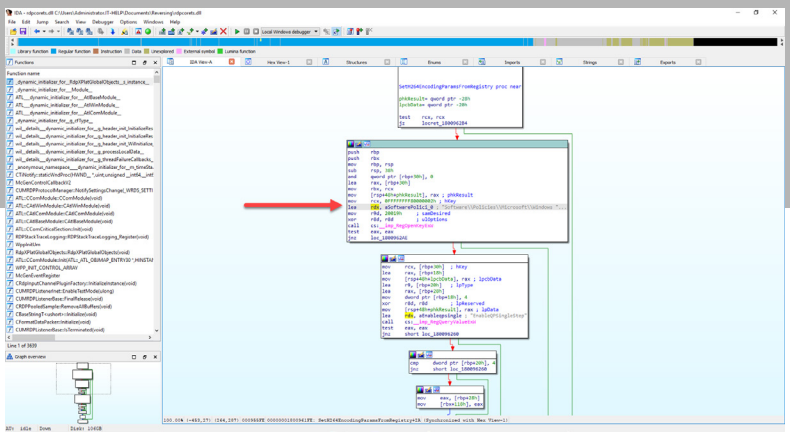
The corresponding symbol is automatically selected and shown in the **IDA View-A** tab. A lot of times, strings do not have a proper name in the binary, so IDA generates a name for them. In this case, our string is called «aSoftwarePolici_0». Right-click on the symbol name, then select **Jump to xref to operand...**:



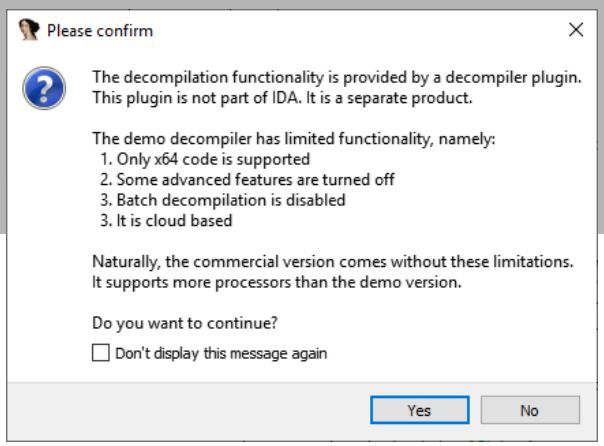
A list of functions using this specific string symbol appears. «SetH264EncodingParametersFromRegistry» looks like a good choice, so double-click on it:



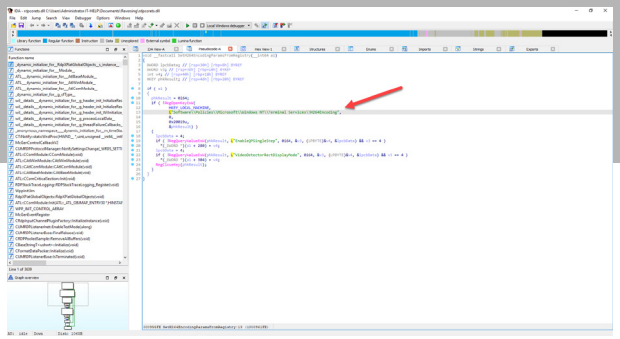
The IDA view now presents the disassembled «SetH264EncodingParametersFromRegistry» function, with commented assembly instructions:



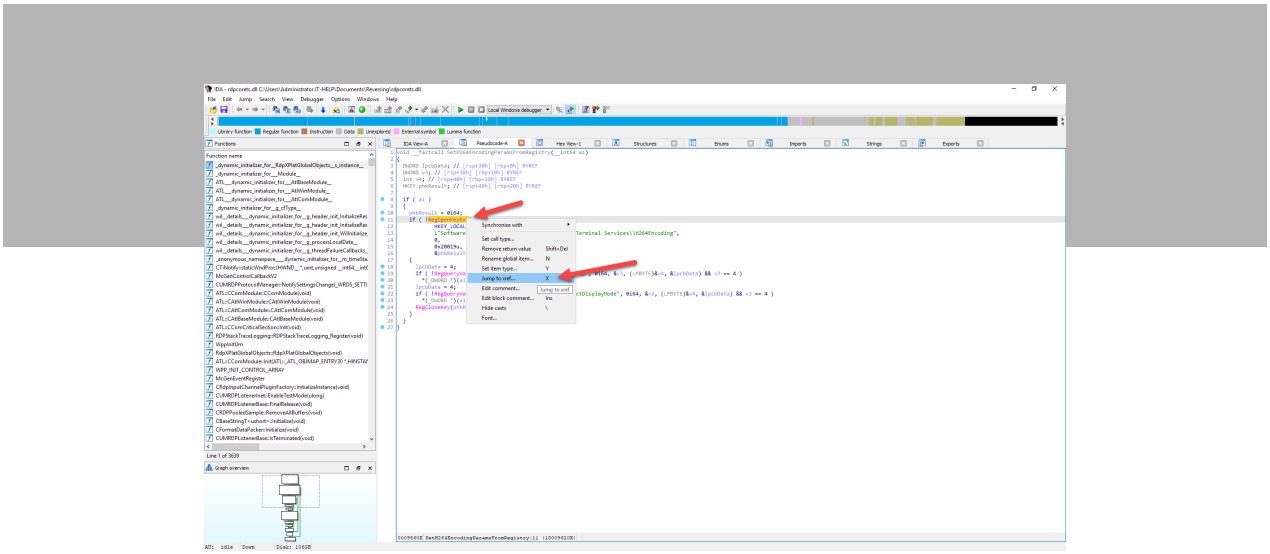
While some advanced reverse engineers like this kind of view, it is quite difficult to read, especially without knowledge of assembly language. Press F5 to decompile the function into readable pseudocode. When IDA prompts for confirmation, click **Yes** to proceed:



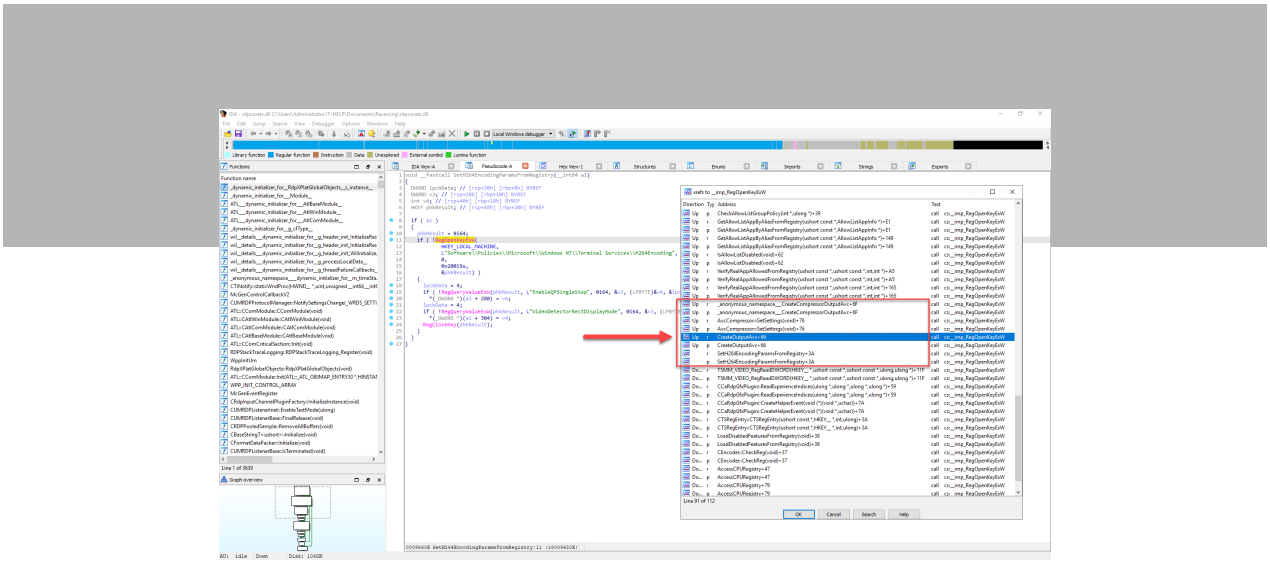
The decompiled function is now shown in all of its glory. Please bear in mind that since a lot of information is lost at compile time, IDA can only reconstruct automatically certain things, and make an educated guess at the rest. This particular example is surprisingly well decompiled, but most functions will have a lot of inaccuracies in them.



Decompiling a function is exciting, but did we find the right one? «EnableQPSingleStep» and «VideoDetectorRectDisplayMode» are clearly registry keys, but they have names that don't mean much except to someone well versed in H.264 codec internals. You can right-click on a function name or string constant in the pseudocode to find more cross-references to them. Let's try it on the «RegOpenKeyExW» function and see where we land:

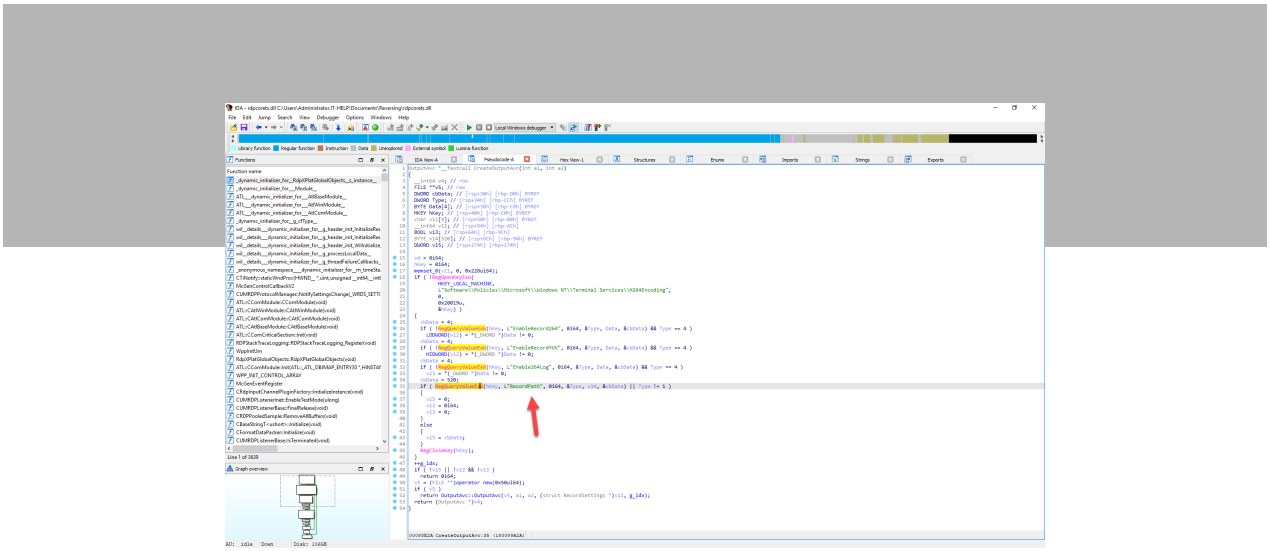


As expected, «RegOpenKeyExW» is used in a lot of other places that deal with registry keys, most of which are unrelated to H.264. However, we can spot a few of the functions from the previous list of cross-references on the «H264Encoding» string, so let's move on to the «CreateOutputAvc» function:



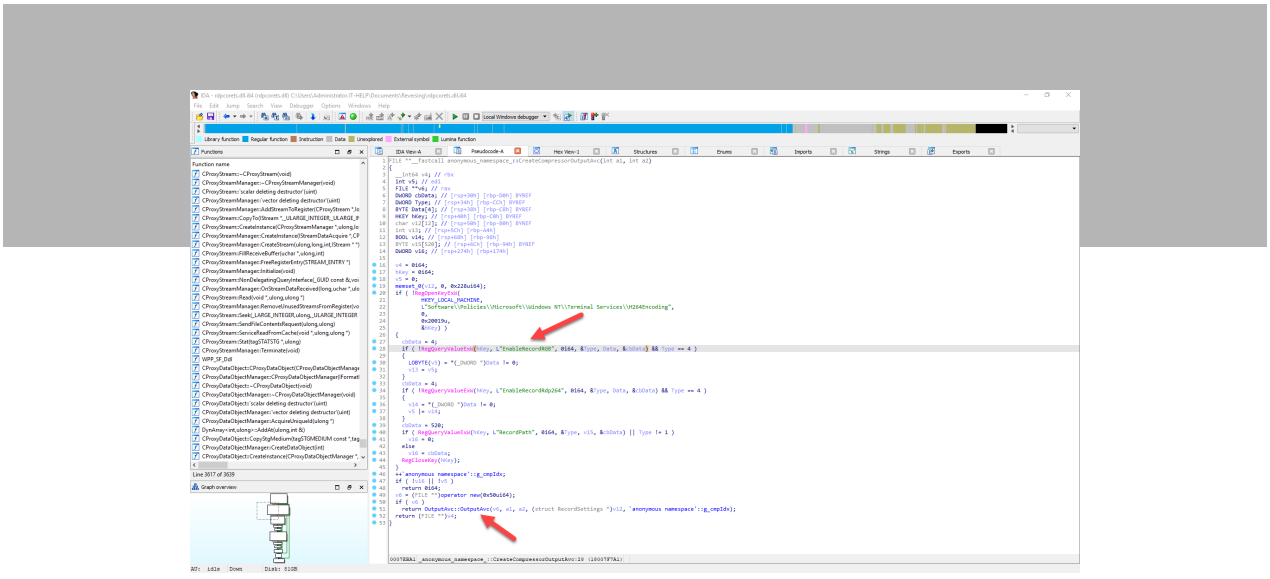
Now that's what I'm talking about! We can see the following registry key names that hint at H.264 recording capabilities in the RDP server:

- RecordPath
- EnableRecord264
- EnableRecordYUV

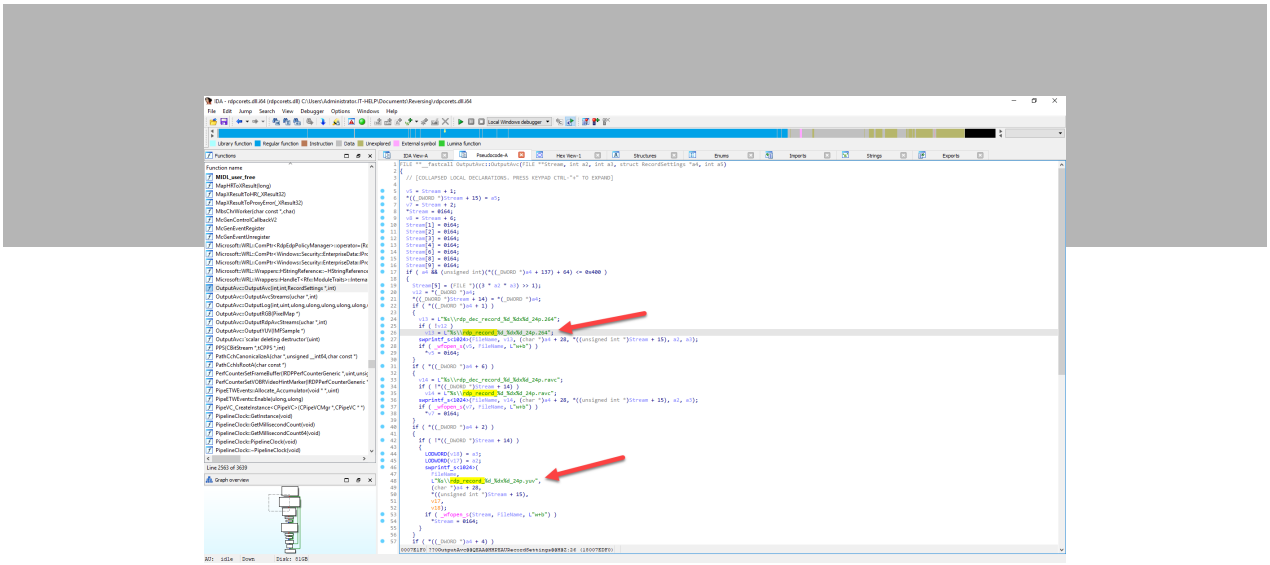


Let's try the «CreateCompressorOutputAvc» function to see if we can find more. It looks similar to the previous function, but with a few different registry keys:

- RecordPath
- EnableRecordRGB
- EnableRecordRdp264



We are definitely onto something here, but we still haven't seen a function that makes use of the values from the registry keys. Class constructions or initialization functions are usually great spots, so let's try «OutputAvc::OutputAvc»:



The decompiled output from this OutputAvc constructor is more complex and contains more inaccuracies, but we can still see that it creates files with a specific name formatting. We could continue this for hours, so let's stop and see if we can act on the information we already have.

Collecting Information

Once you get started with IDA, one issue most people face is finding a lot more information than they can process. You should see your primary goal as the «main quest» of the game, and everything else as «side quests». If you start taking too many tangents, you will lose track of your initial goal.

The best way to collect information is to use your favorite text editor to paste all sorts of clues as you find them. Do not try to organize them too much it's not worth it. You often learn the true value of individual pieces of information once they have been linked to something else.

Let's go back to our functions of interest for the RDP H.264 registry keys, and make a proper list. We have the names, but not the types, so let's look at the API documentation for RegQueryValueExW.

The lpType output parameter contains the type as defined in Registry Value Types, but unfortunately we do not have the numerical values for each. This is a frequent problem that can be fixed by using a copy of the Windows SDK headers. In this case, only two types are used **REG_SZ** (1) and **REG_DWORD** (4).

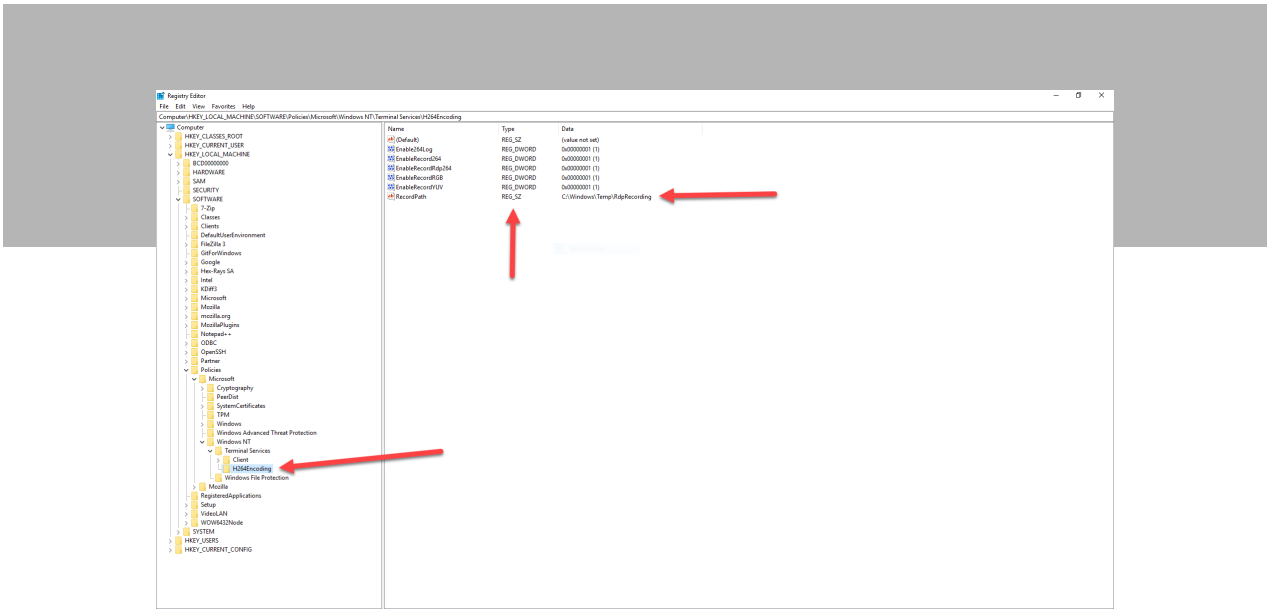
The only string type (REG_SZ) is «RecordPath» all other registry keys are numbers (REG_DWORD) with a value of 1 or 0. Please note that since «RecordPath» is a «REG_SZ» type and not a «REG_EXPAND_SZ» type, it cannot contain environment variables like «%SystemRoot%». We end up with the following list of registry keys with comments:

HKEY_LOCAL_MACHINE\Software\Policies\Microsoft\Windows NT\Terminal Services\H264Encoding

- **RecordPath:** full path to an output directory for recording
- **Enable264Log:** enables H.264 logging, must be set to 1
- **EnableRecordYUV:** enables raw YUV capture dump (very large!)
- **EnableRecordRGB:** enables raw RGB capture dump (very large!)
- **EnableRecord264:** enables raw H.264 bitstream dump
- **EnableRecordRdp264:** enables raw RDP H.264 stream dump

Testing It Out

Now that we have collected valuable information, let's try using the registry keys to confirm that they actually work. Open the registry editor (regedit.exe), then create the «H264Encoding» registry key. Create the «C:\Windows\Temp\RdpRecording» directory, then set the «RecordPath» key value accordingly. Create and set all of the other registry key values to 1 to enable all types of recording.



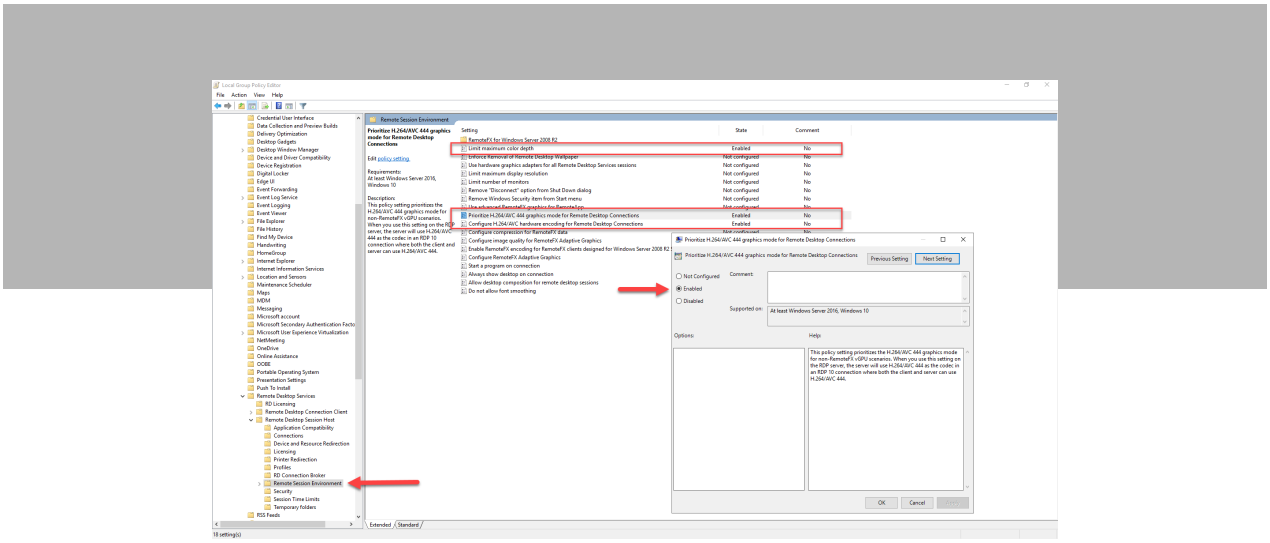
RDP H.264 registry keys are only going to be used if H.264 is used for the RDP connection, so let's tweak the RDP server configuration. Open the group policy editor (gpedit.msc) then browse to the following section under **Computer Configuration**:

Computer Configuration > Administrative Templates > Windows Components > Remote Desktop Services > Remote Desktop Session Host > Remote Session Environment

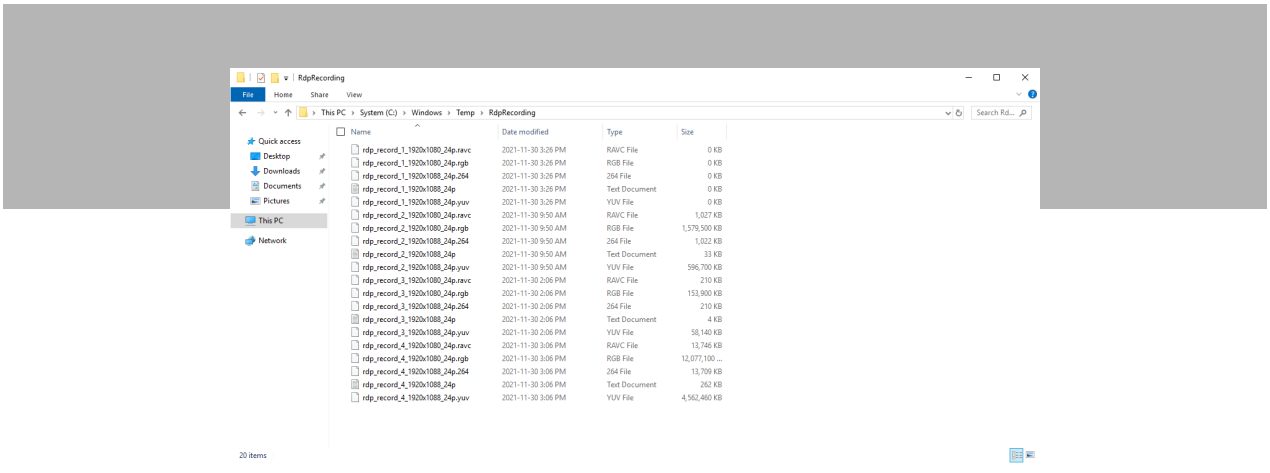
Enable the following policies:

- Prioritize H.264/AVC 444 graphics mode for remote desktop connections
- Configure H.264/AVC hardware encoding for remote desktop connections

Enable and set the **Limit maximum color depth** policy to **32 bit** as it can affect codec negotiation as well.



Reboot the RDP server to apply the changes, then connect with RDP, do a few things to create image updates, and then sign out. Reconnect with RDP and then open the «C:\Windows\Temp\RdpRecording» directory to see if it worked:



Empty files are usually the ones currently used by the RDP server; they are flushed to the disk only when the session is terminated, which is why a full sign out is required. Since these are raw data dumps used for internal debugging, they require a bit of transformation.

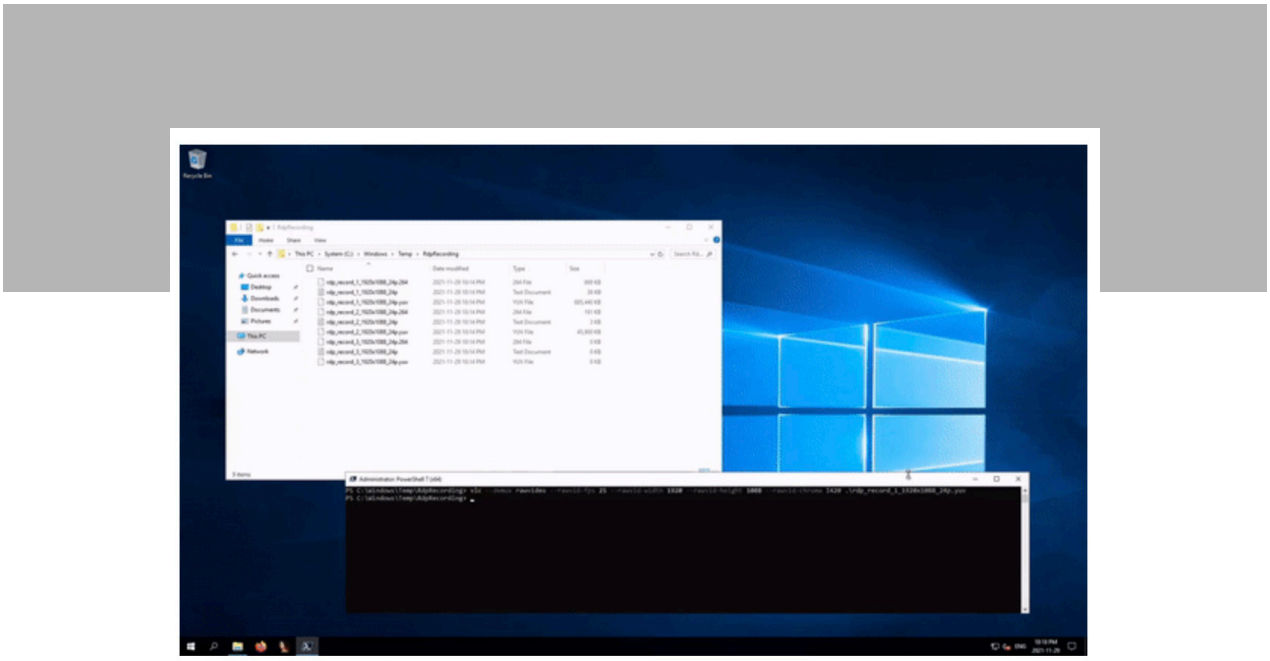
The uncompressed YUV pixels can be played directly using VLC and the right command-line options:

```
choco install vlc
cd "C:\Windows\Temp\RdpRecording"
$Env:Path += ";$Env:ProgramFiles\VideoLAN\VLC"
vlc --rawvid-fps 24 --rawvid-width 1920 --rawvid-height 1088 --rawvid-c
```

Alternatively, the raw H.264 bitstream can be embedded into an .mp4 video file using ffmpeg:

```
choco install ffmpeg
cd "C:\Windows\Temp\RdpRecording"
ffmpeg -i rdp_record_0_1920x1088_24p.264 -codec copy rdp_record_0_1920x
```

The actual file names and parameters will vary, so adjust them accordingly. Here is what it should look like when it works:



Just like most «secret» registry keys, they weren't hidden, but they weren't meant to be used or relied upon. There is also no guarantee that they will be supported in the future.

Closing Thoughts

Without access to the source code, one can still gain insight into closed-source binaries used on millions of devices. There is a certain adrenaline rush associated with finding ways to achieve useful things that are not normally supported. While advanced reverse engineering takes a lot more work, one has to start somewhere. If you are a beginner in reverse engineering, did you find this blog post useful? If so, what would you like to learn next?