Reverse RDP – The Path Not Taken

May 14, 2020

Research by: Eyal Itkin

Overview

During 2019, we published our research on the Reverse RDP Attack: Part 1 (https://research.checkpoint.com/reverse-rdp-attack-code-execution-on-rdp-clients/) and Part 2 (https://research.checkpoint.com/reverse-rdp-the-hyper-v-connection/). In those blog posts, we described how we found numerous critical vulnerabilities in popular Remote Desktop Protocol (RDP) clients. In addition, we focused on a Path-Traversal vulnerability we found in Microsoft's RDP client, a vulnerability that was also applicable as a guest-to-host VM escape in Hyper-V Manager.

When testing the applicability of our findings to Microsoft's RDP client for MacOS, we made an interesting discovery: not only can we bypass Microsoft's patch, we can bypass any path canonicalization check performed according to Microsoft's best practice.

Important Note: The bypass vulnerability for Microsoft's core Path-Traversal check still hasn't been fixed. More details can be found in Microsoft's official response at the end of this blog. Due to the implications of this vulnerability, we urge all software developers and security researchers to be aware of this issue, and make sure their own software projects are manually patched.

Microsoft's Original Patch

In the second part (https://research.checkpoint.com/reverse-rdp-the-hyper-v-connection/) of our Reverse RDP Attack publication, we analyzed in detail the patch (https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2019-0887) that Microsoft issued for our vulnerability (CVE-2019-0887 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-0887)). The patch was supposed to be relatively simple, as the Path-Traversal vulnerability was due to lack of sanitation checks on the file paths that were included inside the incoming `FileGroupDescriptorW` clipboard format. Microsoft followed their own best practice and added a validation check based on the function PathCchCanonicalize (https://docs.microsoft.com/en-us/windows/win32/api/pathcch/nf-pathcch-pathcchcanonicalize), as can be seen in Figure 1:

```
pszFilename = pCurrentFileRecord->szFilename;
status_code = PathCchCanonicalize(&pszPathOut, 0x104ui64, pszFilename);
if ( (status_code & 0x80000000) != 0 )
{
```

Figure 1: `PathCchCanonicalize` is used to calculate the canonicalized form for every filename.

If successful, the canonicalized output is then compared to the original filename, and any mismatch between the two results in an error. This means that if our filename contains strings of the form `.` or `..`, it is changed when converted to the canonicalized form, thus failing the validity check.

We assumed this patch meant the vulnerability was indeed fixed, and we even mentioned it in our previous blog post: "... the fix matches our initial expectations, our Path Traversal vulnerability is now fixed."

MacOS RDP Client

When examining the various RDP clients, we chose to focus on rdesktop (https://www.rdesktop.org/), FreeRDP (https://www.freerdp.com/) and Microsoft's built-in client (`Mstsc.exe`). At the time, we didn't look at any specific RDP client for MacOS, as we heard that MacOS users often use one of the mentioned open-source clients. To our surprise, in October 2019, after our research was marked as "finished", we were contacted by Chris Risvik from Mnemonic. After seeing our presentation at Black Hat (https://www.youtube.com/watch?v=3wncyS-QOBk) and reading more of our research, Chris started playing around with Microsoft's RDP client on his MacOS and found an interesting behavior. This behavior led Chris to believe that CVE-2019-0887 could be applied to the Microsoft RDP client as well, and he decided to contact us.

Surprised to hear about a built-in Microsoft RDP client for MacOS, and intrigued by his findings, we decided it was worth investigating. As we had an existing setup for exploiting the Path-Traversal vulnerability, all we needed was to get a Mac computer and use Microsoft's RDP client for MacOS to connect to our existing malicious RDP server. This is where things got even more interesting.

As it happened, on the day we scheduled for this experiment, our MacOS researcher arrived a bit late to the office. As he is usually the first to arrive, this was a rare stroke of luck. While I was waiting for him to arrive, I decided to update the exploit's configuration on the server. Until now, we used Windows paths (`A\B\C`). Because we were going to traverse a MacOS path, I thought that it would be preferable to use forward-slashes in our exploit instead of backward-slashes: `A/B/C`.

After modifying the exploit's configuration file, we tested the setup locally with 2 Windows machines, to check that everything still works. The original exploit (using `\`) was successfully blocked by Microsoft's patch, resulting in `explorer.exe` getting stuck. Just out of curiosity, we also tested the modified exploit (using `/`) and surprisingly enough, the exploit worked. The simple replacement of `\` to `/` in our malicious RDP server was enough to bypass Microsoft's patch!

Note: When we later tested the MacOS RDP client, we saw that it was implemented with better security; and it isn't vulnerable to CVE-2019-0887. This goes to show that you can never predict where a research lead will take you, and what vulnerabilities you will discover along the way.

Back to PathCchCanonicalize

Intrigued by this unexpected behavior, we decided to compile a simple test program to focus solely on Microsoft's main path canonicalization function: `PathCchCanonicalize`. In Figure 2 we can see the naive check:

```
int main()
{
    wchar_t output_buffer[0x200];
    memset(output_buffer, 0, sizeof(output_buffer));

    int status_code = PathCchCanonicalize(output_buffer, 0x200, L"..\\..\\Evil.bat");
    printf("The canonical string is: %ls, and the status code: %d\n", output_buffer, status_code);
    return 0;
}
```

Figure 2: Testing `PathCchCanonicalize` against a path with backward slashes (\).

As expected, the output was: "The canonical string is: Evil.bat, and the status code: 0."

After that, we performed the same test with a path that uses forward slashes (/), as can be seen in Figure 3:

```
int main()
{
    wchar_t output_buffer[0x200];
    memset(output_buffer, 0, sizeof(output_buffer));

    int status_code = PathCchCanonicalize(output_buffer, 0x200, L"../../Evil.bat");
    printf("The canonical string is: %ls, and the status code: %d\n", output_buffer, status_code);
    return 0;
}
```

Figure 3: Testing `PathCchCanonicalize` against a path with forward slashes (/).

To our surprise, the output was: "The canonical string is: ../../Evil.bat, and the status code: 0."

It seems that `PathCchCanonicalize`, the function that is mentioned in Windows's best practice guide on how to canonicalize a hostile path, ignores forward slash chars! We later verified this behavior by reverse engineering Microsoft's implementation of the function, and saw that it splits the path to parts by searching only for `\` and ignoring `/`.

Just to get a sense of the implications of this vulnerability beyond the scope of RDP, we went looking for how Microsoft recommends handling directory traversal attempts such as `...` The search led us to `PathCcCanonicalize`. This functions claims to do just that in MSDN, but is actually deprecated due to a potential buffer overrun vulnerability:

PathCanonicalizeA function

12/05/2018 • 3 minutes to read

Simplifies a path by removing navigation elements such as "." and ".." to produce a direct, well-formed path.

Note Misuse of this function can lead to a buffer overrun. We recommend the use of the safer **PathCchCanonicalize** or **PathCchCanonicalizeEx** function in its place.

Figure 4: Screenshot from MSDN explaining how to block Path-Traversal attempts using WinAPI.

The official note explicitly directs developers to use `PathCchCanonicalize` instead, which (as we have just shown) can be bypassed by using a forward slash instead of a backward slash.

We presented our senior software developers with the following challenge: Implement a function that receives a file path as an input, and verifies that the file would be written inside a predetermined folder. In other words, use Windows API and best practices to block a Path-Traversal attack. As expected, the common answer we received can be seen in Figure 5:

```
BOOL IsCanonicalSubpath(LPCWSTR pszFile) {
    if (!PathIsRelative(pszFile)) {
        return FALSE;
    }
    WCHAR szConcat[MAX_PATH] = { 0 };
    if (!SUCCEEDED(StringCchCat(szConcat, MAX_PATH, L"C:\\Temp\\Extraction\\Folder"))) {
        return FALSE;
    }
    if (!SUCCEEDED(StringCchCat(szConcat, MAX_PATH, pszFile))) {
        return FALSE;
    }
    WCHAR szCanonical[MAX_PATH] = { 0 };
    if (!SUCCEEDED(PathCchCanonicalize(szCanonical, MAX_PATH, szConcat))) {
        return FALSE;
    }
    return lstrcmpW(szCanonical, szConcat) == 0;
}
```

Figure 5: Sample path sanitation code, always using `PathCchCanonicalize`.

In other words, when following Microsoft's best practices, even senior developers used the vulnerable function, and expected it to handle both `/` and `\` separated paths. This means that there are probably dozens of different projects all around the world that are vulnerable to a Path-Traversal attack, as they rely on Microsoft's vulnerable path canonicalization function.

Windows Paths 101

On Unix-based systems, the used path separator char is a forward slash (`/`), while traditionally on Windows the used char is a backward slash (`\`). However, from the earliest days of Windows, it supported both `/` and `\`. Therefore, performing any file operation (access / read / write) will work with both separator chars as if they were `\`.

The vulnerability that we just found is a classic case of implementing a sanity check using a different logic than the one that is used for the action we wish to protect. Somehow, the developers of the core Windows functionality for sanitizing file paths forgot about Windows's support of `/` chars, and they instead re-implemented the file path logic on their own, supporting only `\` separators.

CVE-2020-0655

We once again contacted Microsoft and disclosed the details of their improper fix for CVE-2019-0887, a fix that could be bypassed due to the new vulnerability we found in `PathCchCanonicalize`. Microsoft acknowledged our findings, and on February's Patch Tuesday (https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0655) issued a patch for it labeled: CVE-2020-0655 (https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2020-0655).

Knowing that `PathCchCanonicalize` is located inside `KernelBase.dll`, we were surprised to find that it wasn't updated as part of this month's Windows Updates. As Figure 6 shows, `kerberos.dll` was updated during February's update, while `KernelBase.dll` remained untouched.

kerberos.dll	11/2/2020 21:11	Application extens...	982 KB
kernel.appcore.dll	15/9/2018 10:28	Application extens...	58 KB
kernel32.dll	10/7/2019 07:59	Application extens...	709 KB
KernelBase.dll	11/12/2019 09:07	Application extens...	2,637 KB
KeyboardSystemToastIcon.contrast-whit...	15/9/2018 10:28	PNG File	1 KB

Figure 6: Timestamps for Windows `.dll` files showing that `KernelBase.dll` wasn't updated.

Not knowing what to expect, we filtered out all the files that were updated as part of the patch, and singled out `mstscax.dll` as a candidate for a potential fix. If our suspicion is true, it is possible that Microsoft only patched the attack scenario for the RDP clients, and neglected the wider use of the vulnerable API function.

In Figure 7, we can see that the function `CFormatDataPacker::ValidateFilePaths`, added as part of the patch for CVE-2019-0887, was modified once again:

```
dwLoopIndex = 0;
if ( dwRecordSize )
{
  dwIndex = 0i64;
  do
  {
    cChar = pszOriginalFilename[dwIndex];
    if ( cChar == '/' )                    // Replacing all '/' chars with '\'
      pszFilename[dwIndex] = '\\';
    else
      pszFilename[dwIndex] = cChar;
    dwIndex = ++dwLoopIndex;
  }
  while ( dwLoopIndex < dwRecordSize );
}
status_code = PathCchCanonicalize(&pszPathOut, 0x104ui64, (PCWSTR)pszFilename);
if ( (status_code & 0x80000000) != 0 )
{
```

Figure 7: Patch for replacing / chars with \ chars inside the RDP client.

Our initial guess was correct. Microsoft fixed their improper patch for CVE-2019-0887 by adding a workaround to the still vulnerable `PathCchCanonicalize` function.

We searched for a published advisory to warn users from using the vulnerable path canonicalization function, but we failed to find one. MSDN (https://docs.microsoft.com/en-us/windows/win32/api/pathcch/nf-pathcch-pathcchcanonicalize) still recommends that users use this function when trying to sanitize malicious input.

We should note, however, that this patch makes sense in a case where the file handling in RDP is uniquely different from the rest of WinAPI. However, having checked the common WinAPI for file handling, such as `CreateFile`, `DeleteFile`, `CreateFolder` etc., we can conclude that they do respect both `/` and `\`, meaning that the RDP case is not unique in any way.

Therefore, in our opinion, `PathCchCanonicalize` is the optimal place for a patch that aims to block potential Path-Traversal attacks originating from the use of `/` separators.

Microsoft's Response

After analyzing Microsoft's patch, we contacted MSRC and presented our findings along with the expected publication date of this research blog. We haven't received their comment thus far and have notified them about publishing the blog in this format.

Conclusion

First and foremost, due to the serious implications of an improper fix to the RDP vulnerability (CVE-2019-0887), we urge all readers to make sure to install Microsoft's patch (https://portal.msrc.microsoft.com/en-US/security-guidance/advisory/CVE-2020-0655).

However, as we've seen when analyzing Microsoft's patch for CVE-2020-0655, this fix does not address the core vulnerability in the `PathCchCanonicalize` function. We fear that just like in the Reverse RDP scenario that we just demonstrated, the implications of a simple bypass to a core Windows path sanitation function may pose a serious risk to many other software products.

We therefore urge all software developers and security researchers to be aware of this vulnerability, and make sure their own software projects are manually patched.

It is still a mystery how a simple Path-Traversal bypass was able to remain unnoticed for so many years in Microsoft's core path sanitation functionality. We suspect that many developers decided to re-invent the wheel on their own and therefore pentests found only bugs in many different proprietary implementations. Otherwise, we would expect a regular pentest performed on a large system that handles file paths in C code, would find this Path-Traversal bypass.

We once again want to thank Chris Risvik from Mnemonic whose work prompted us to take another look at the RDP clients, as we did eventually find a vulnerability, if not one in MacOS RDP clients. As we stated in our previous conclusion, "this research is a good example that shows a researcher can never know if a given research project reached its end."

RELATED ARTICLES

(/research.checkpoint.com/2021/disruption/)

TOOLS

SANDBLAST FILE ANALYSIS (HTTPS://THREATEMULATION.CHECKPOINT.COM/)

URL CATEGORIZATION (HTTPS://WWW.CHECKPOINT.COM/URLCAT/)

INSTANT SECURITY ASSESSMENT (HTTP://WWW.CPCHECKME.COM/CHECKME/)

LIVE THREAT MAP (HTTPS://THREATMAP.CHECKPOINT.COM/THREATPORTAL/LIVEMAP.HTML)

ABOUT US (HTTPS://RESEARCH.CHECKPOINT.COM/ABOUT-US/)

CONTACT US (HTTPS://RESEARCH.CHECKPOINT.COM/CONTACT/)

SUBSCRIBE (HTTPS://RESEARCH.CHECKPOINT.COM/SUBSCRIPTION/)