LINGI2144: Secured System Engineering
Tutorial 9: Dynamic Analysis

Academic year : 2020 - 2021

Teacher : Legay Axel
Course : LINGI2144
Collaborators :
Crochet Christophe
Duchene Fabien
Given-Wilson Thomas
Strebelle Sebastien

1 Prerequisite

Working directory: `~/SecurityClass/Tutorial-09`

There is a new image for this tutorial, it is highly recommended you download this new image to save a lot of effort! Also note that due to lots of files and sharing between parts of the tutorial, the tutorial files have not been split up according to section.

If you downloaded the new image and made on changes to your VM configuration then you can skip this section. If you made changes to your VM configuration it is HIGHLY recommended you keep the memory size small (or change it back to a small amount).

To run the tutorial today you will need to download some files these are:

1. `LiMe`

   ```
   git clone https://github.com/504ensicsLabs/LiME.git
   ```

2. `volatility`

   ```
   git clone https://github.com/volatilityfoundation/volatility.git
   ```

3. A wannacry memory dump (available from):

   ```
   https://mega.nz/#!Au5xlCAS!KX5ZJKYzQgDHSa72lPFwqKL6CsZS7oQGbyyQrMTH9XY
   ```

   the password to extract this is in the Tutorial files in password.txt.

4. You will also need some libraries to make things work, you should run:

   ```
   apt-get install memdump dwarfdump libdistorm3-3 libjansson4 libyara3
       python-distorm3 python-jdcal python-openpyxl python-py python-pytest
       python-yara volatility volatility-tools
   ```

After downloaded and installing the above, you should reboot the VM.

2 Exercise

2.1 Memory Dumping with `LiMe`

Let's build a `LiMe` memory profile:

```
cd LiME/src
make
```

this should build your kernel module, e.g. `lime-5.3.0-kali2-686-pae.ko` You can dump the system memory using the `LiMe` module you built with (**take long time**):

```
sudo insmod lime-5.3.0-kali2-686-pae.ko
    "path=/home/admin/SecurityClass/Tutorial-09/memory.dump format=lime"
```

After the dump has completed you will need to remove the `LiMe` module with

```
sudo rmmod lime
```

The above command (or variations) will be assumed to be used for dumping memory for the rest of this tutorial (but not repeated).

2.2 Configuring volatility

NOTE: Details from:

`https://github.com/volatilityfoundation/volatility/wiki/Installation#getting-volatility`

but the main steps for Kali Linux 32-bit are described below, you should not need the above link unless your system is different or something goes wrong.

The parts below are for your own information and configuration, you should not need to do these on the downloaded image. If the last line of this section works for you, then skip the configuration here. The you will need to build a profile. Details are here

`https://github.com/volatilityfoundation/volatility/wiki/Linux`

but the quick version that worked for me was and should work on your image is:

```
cd volatility/tools/linux
make
cd ../..
sudo zip ./volatility/plugins/overlays/linux/Kali-5.3.0-686.zip
    ./tools/linux/module.dwarf /boot/System.map-5.3.0-kali2-686-pae
```

you can now find the name of the profile you created with

```
python vol.py --info | grep -i kal
```

Now we can use this `LiMe` memory dump and volatility to see which processes were running when we took the memory dump:

```
python vol.py -f /home/admin/SecurityClass/Tutorial-09/memory.dump
    --profile=LinuxKali-5_3_0-686x86 linux_pslist
```

2.3 Memory Dumping Basics

NOTE: The "`build.sh`" script will rebuild the "malware" for this tutorial, you should not need to do this unless you made major changes to your VM environment.

In the `Tutorial-09/bin` directory you will fond various programs. Run the program `malware_1` with

```
./bin/malware_1
```

which will print `"I am evil!!!"` and then pause. During this pause dump the memory of the system using `LiMe`.

Once you have done the memory dump you can kill `malware_1` (it is set to sleep for a long time so you have time to do the dump).

Now we can see which processes were running when we dumped the memory with

```
python vol.py -f /home/admin/SecurityClass/Tutorial-09/memory.dump
    --profile=LinuxKali-5_3_0-686x86 linux_pslist
```

and we should see that malware 1 was running and what process ID it had.

Now we can look inside this process to see that the string "I am evil" is present with

```
python vol.py -f /home/admin/SecurityClass/Tutorial-09/memory.dump
    --profile=LinuxKali-5_3_0-686x86 -p 2452 linux_yarascan -Y "I am evil"
```

where 2452 is the process ID of `malware_1` in the memory dump.

2.4 Memory Dumping Again

There are four more malware programs in the bin directory. Observe that only malware 1 is detected as malware by a simple YARA rule:

```
yara evil.yar bin
```

However, the others also print `"I am evil!!!"`.

NOTE: You can test your YARA rules from last week on these malware if you want, some may be detected (or detectable) with YARA.

Run the other programs and perform memory dumps to confirm that you can observe the malicious behaviour string in the memory.

2.5 WannaCry

WARNING: This is malware we are working with, do not use this in any malicious way! Be very careful how you handle these files!

Extract the dump into a directory, the rest of this section will assume they are in `Tutorial-09/wcry`.

First let's use volatility to determine information about the image:

```
python volatility/vol.py -f wcry/wcry.raw imageinfo
```

Now like before we can look at the processes running in the image:

```
python volatility/vol.py -f wcry/wcry.raw pslist
```

Notice in this list a process `"@WanaDecryptor@"` that probably has something to do with wannacry.

Let's do a scan of the processes and their parents

```
python volatility/vol.py -f wcry/wcry.raw psscan
```

Here we can see that the wannacry components are being run by the task scheduler `"tasksche.exe"`.

We can all see what dynamically-linked libraries both the decryptor and the task scheduler are using:

```
python volatility/vol.py -f wcry/wcry.raw dlllist -p 740
python volatility/vol.py -f wcry/wcry.raw dlllist -p 1940
```

Observe that both are running from the same directory, and one that does not look like a normal OS directory.

We can also inspect the "handles" (used in Windows APIs for interaction) used by the code:

```
python volatility/vol.py -f wcry/wcry.raw handles -p 1940
```

Here we can see some Mutexes which are interesting as these are often used to ensure that infection does not occur multiple times.

If we search online for these we will quickly find results that indicate that

```
MsWinZonesCacheCounterMutexA
```

is used by wannacry.

Now let's extract the network connections that were open during the time wannacry was running:

```
bulk_extractor -E net -o pcap wcry/wcry.raw
```

This will create several files in the "pcap" directory for us to use.

Now we can find the IP addresses that the system connected to with

```
tshark -T fields -e ip.src -r pcap/packets.pcap | sort -u
```

This can be useful in later analysis and other parts of program hacking...

Now let's find the files in memory that we care about (running from the malware's directory):

```
python volatility/vol.py -f wcry/wcry.raw filescan
    | grep ivecuqmanpnirkt615
```

Let's make a diectory for our file and memory dumps:

```
mkdir filedump
mkdir memdump
```

Now we can see the
les we care about we can dry and dump them with commands like:

```
python volatility/vol.py -f wcry/wcry.raw dumpfiles -Q 0x00000000022ec718
    -D filedump/
```

We can extract the files we're interested in here. We can also dump the memory of the processes we care about:

```
python volatility/vol.py -f wcry/wcry.raw memdump -p1940,740 -D memdump
```

Now we have the files and the memory when the malware was running. We can use "strings" to look for interesting strings, and of course explore further to find more about how the program works and to reverse engineer the code.

BONUS: Look for interesting strings in the files, can you find the bitcoin wallet, the IP addresses from the connections, the script sent to the system, etc?