# Malware Analysis on Process-Injector Malware.

20 May 2024 02:36 PM

**Project Name: Process-Injector**

**Presenter Name: Rishikesh Rahangdale**

**Description**

**In this project, I am trying to perform Malware Analysis with the techniques and skills in my knowledge to find out the workings of this random sample from PMAT Labs which is posted by Husky Hacks.**

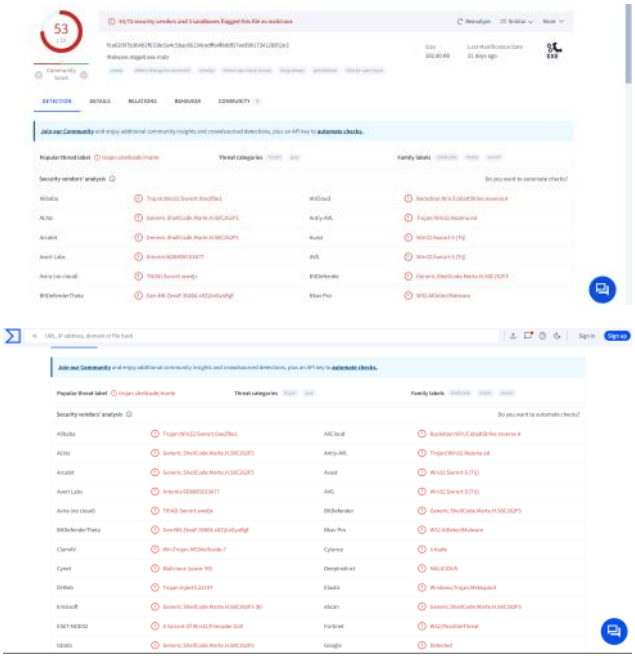
**Project Goals**

- In this project, I am trying to perform Malware Analysis with the techniques and skills in my knowledge to find out the work ings of this random sample from PMAT Labs which is posted by Husky Hacks.
- Analytical report on static and dynamic analysis.
- To get a broader perspective , learn and improve skills to handle malware with safety and utmost precaution.

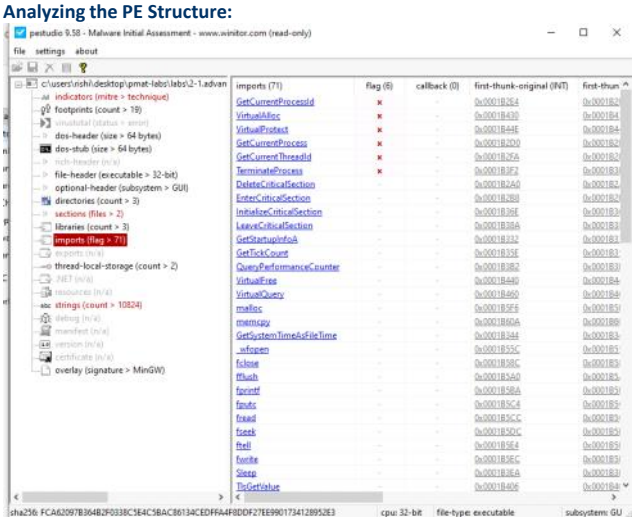
**Basic Static Analysis :**

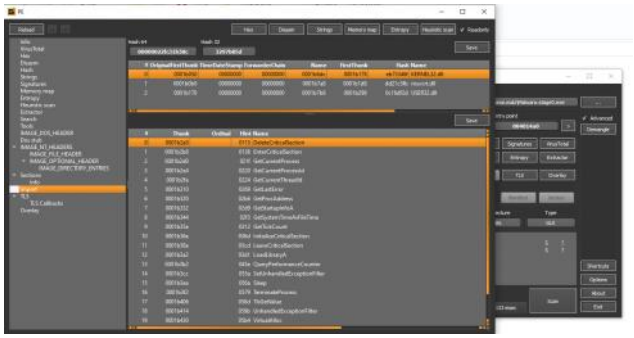
**Resources/Techniques-**

- **Hashing the file : MD5**-6d8895c63a77ebe5e49b656bdefdb822 **| SHA256**-fca62097b364b2f0338c5e4c5bac86134cedffa4f8ddf27ee9901734128952e3
- **Virustotal: in virustotal almost every security vendor had flagged the file as malicious and some interesting findings about this file say it has evading capabilities.**



- **Analyzing the PE Structure:**





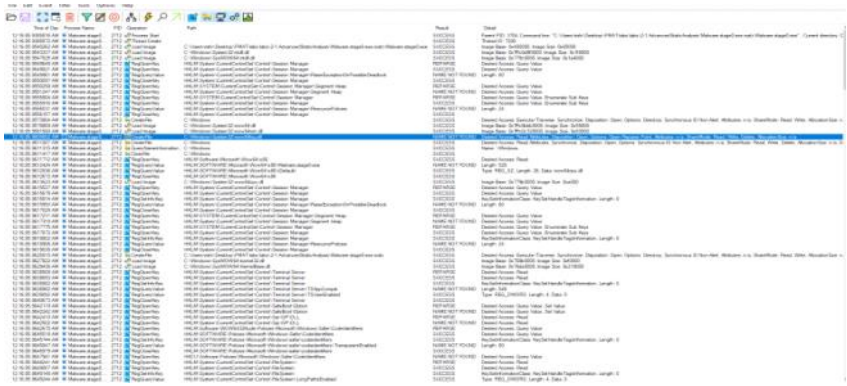
- **Strings/Floss:**
  **After flossing out the executable, we found a lot and lots of strings and most of them included the imports and API calls as given below**
  **Some of the other interesting strings are as follows-**
  C:\Users\Administrator\source\repos\CRTInjectorConsole\Release\CRTInjectorConsole.pdb ( seems like the path of the malware author )
  <?xml version='1.0' encoding='UTF-8' standalone='yes'?>
  <assembly xmlns='urn:schemas-microsoft-com:asm.v1' manifestVersion='1.0'>
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
  <security>
  <requestedPrivileges>
  <requestedExecutionLevel level='asInvoker' uiAccess='false' />
  </requestedPrivileges>
  </security>
  </trustInfo>
  </assembly>
  GNU C99 9.2-win32 20191008 -m32 -mtune=generic -march=pentiumpro -g -O2 -std=gnu99 -fno-PIE
  ./mingw-w64-crt/crt/crtexe.c
  ./build/i686-w64-mingw32-i686-w64-mingw32-crt
  GUID_MIN_POWER_SAVINGS
  GUID_TYPICAL_POWER_SAVINGS
  NO_SUBGROUP_GUID
  ALL_POWERSCHEMES_GUID
  GUID_POWERSCHEME_PERSONALITY
  GUID_ACTIVE_POWERSCHEME
  GUID_IDLE_RESILIENCY_SUBGROUP
  GUID_IDLE_RESILIENCY_PERIOD
  GUID_DISK_COALESCING_POWERDOWN_TIMEOUT
  GUID_EXECUTION_REQUIRED_REQUEST_TIMEOUT
  GUID_VIDEO_SUBGROUP
  GUID_VIDEO_POWERDOWN_TIMEOUT

- **We analyzed some critical imports that the portable executable is importing many of these functions/API calls can be used in malicious ways for example "DeleteCriticalSections" etc, although they might not look harmful as many genuine programs often use these calls as well. We cannot overlook the possibility of the Executable to combine these API calls to perform some malicious activity.**
  GetCurrentProcess
  GetCurrentProcessId
  GetCurrentThreadId
  TerminateProcess
  VirtualAlloc
  VirtualProtect
  DeleteCriticalSection
  EnterCriticalSection
  GetLastError
  GetProcAddress
  GetStartupInfoA
  GetSystemTimeAsFileTime
  GetTickCount
  InitializeCriticalSection

**Basic Dynamic Analysis :**

**Procedures:**

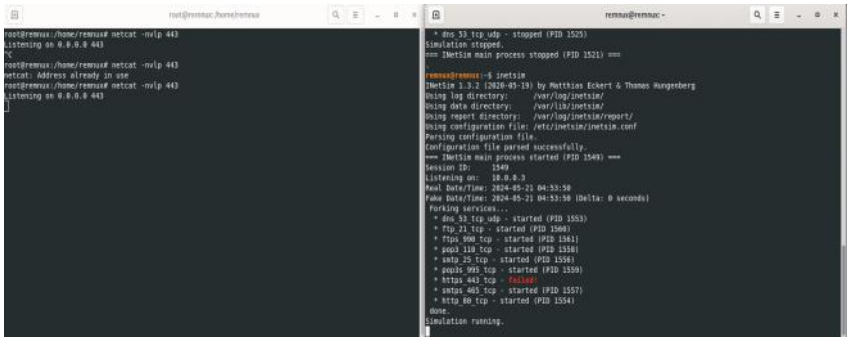
- **We monitored the process of the malware using procmon and found some mysterious registry key modifications which are often used to intentionally harm The system.**



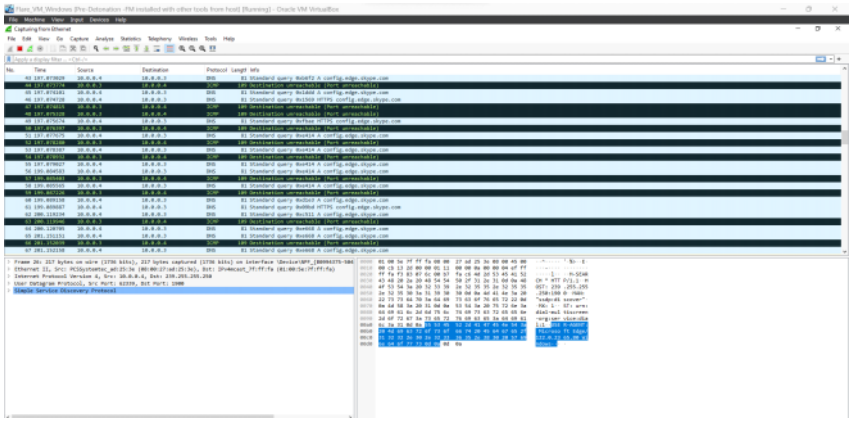
- **We used RegShot to perform and compare the registry key modifications after and before the deployment of the malware.**
  **27 keys are being modified in the second shot and a total 4 keys are being deleted after the comparison which are as follow:**


HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\ServiceInstances
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\ServiceInstances\1930a3c6-96ee-417e-95f7-21facb2ec342
HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Group Policy\ServiceInstances
HKLM\SOFTWARE\WOW6432Node\Microsoft\Windows\CurrentVersion\Group Policy\ServiceInstances\1930a3c6-96ee-417e-95f7-21facb2ec342

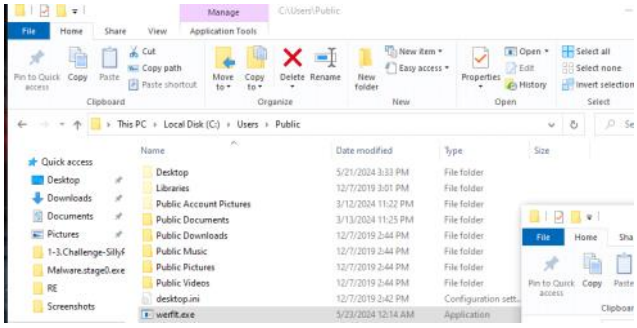
- As we proceed towards the dynamic analysis phase the first step we did was to connect the VM internally in a separate networ k with the remnux machine And started listening on port 443 as we set up the ethernet connection to remnux's IP. But before that, we wanted to check th e network traffic so we Moved on to Wireshark.




- **Wireshark: We found some DNS requests but not malicious on first look.**



- **Sysinternal tool TCP view we found that a Werfault.exe program spawned for a second. Also, A big Indicator of compromise is a new exe file being Created after running the malware in the file system.**

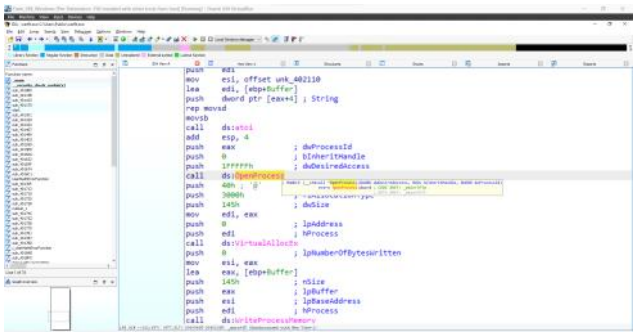



**Advanced Static (Disassembling the code)**

- When we loaded the Werflt.exe in the disassembler we noticed that the program had many functions.

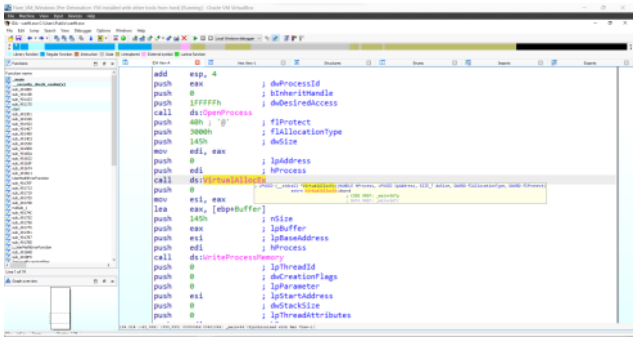
- **We can observe some functions that are being called i.e. "WriteProcessMemory", "CreateRemoteThread"&"CloseHandle". Now since we have some very good resources thanks to MalApi.io we checked it for the corresponding API call and seems It can be used by Malware for Process Injection.**

- **Observations of the Disassembly : Now we know after trying to disassembling the code the actions it is performing :**
  1. **Now we know after trying to disassemble the code the actions it is performing :**
     **We can see that first for the OpenProcess the Process ID is being pushed into the EAX.**

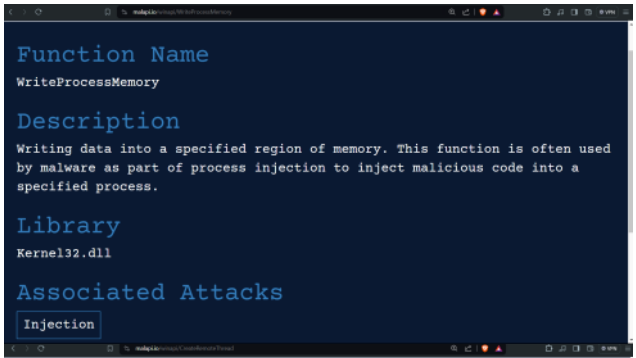


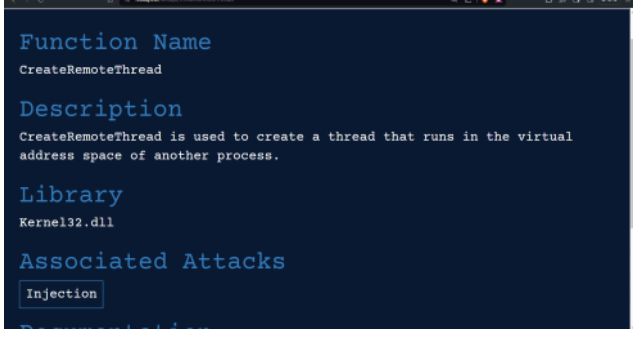
  2. **Next, the arguments are pushed into the stack as an argument call for the OpenProcess handle and after the process is created it Being passed on to another register EDI.**
     **It then creates and allocates virtual memory space for writing its process in that.**
     **Now the API call "WriteProcessMemory" is used for actually writing the process data in allocated memory using the parameters We have already passed.**

  3. **Now in the API call "WriteProcessMemory " is used for actually writing the process data in allocated memory using the parameters We have already passed.**



  4. **Now the program is using the final API call to execute Remote threads which already have all the Perimeters that it needs to be able to create the remote-threat execution.**



- **NOTE: An interesting point to note here is that when doing static analysis before running the Malware we couldn't have got the other Binaries import calls, as when we did the static analysis of Original Program we didn't find these API calls there.**