

Detecting Fileless Malware

Using Large Language Models

Chris, Steven, Arun, Himanshu

Student, Sheridan College

INFO49402: ISS Graduation Capstone Project

August 9th, 2024

Table of Contents

Glossary	2
Abstract	4
Problem Statement	5
Relevant Literature	9
Chosen Solutions	11
Overview of Fileless Malware Detection using SecureBert LLM	11

Key Components of the Solution	11
Step 1: Acquiring Fileless Malware Sample	11
Step 2: Windows 10 VM Setup	14
Step 3: Data Collection Through Memory Dumping:	16
Step 4: Transferring the memory dump	17
Step 5: Data Analysis with Volatility:	18
Step 6: Host.py Script:	21
Step 7: Model Training with SecureBERT:	22
Advantages of the Chosen Solution	26
Conclusion	26
Detailed Justification for the Chosen Solution	26
1. Memory Forensics	26
2. Machine Learning with SecureBERT	27
3. Secure Data Handling with SCP	28
Impact	30
1. Enhanced Detection Capabilities	30
2. Proactive Security Posture	30
3. Support for Incident Response and Threat Hunting	31
Major Activities Carried Out for Solving the Problem	32
1. Problem Analysis and Research	32
2. Setting Up the Virtual Environment	32
3. Executing and Capturing Malware Activity	32
4. Secure Data Transfer	33
5. Forensic Analysis of Memory Dumps	33
6. Training the Machine Learning Model	33
7. Validation and Testing	33
Major deliverables of the project:	34
Software, tools, systems, or other technology employed in the project	36
Results	39
Conclusions and Future Work	41
Bibliography and Acknowledgements	42

Glossary

Term	Definition
Adversaries	People or groups that perform malicious activities.
Binaries	File Format used on Windows to allow for individuals to run programs.
Powershell	Windows operating system tool used to perform automation tasks.
Command Prompt	Windows operating system tool based off of DOS used for troubleshooting, creation, modification of files and folders, etc.
Windows Management Instrumentation (WMI)	Windows Management Instrumentation accessing management information about Operating Systems like applications, networks, devices, etc.
Living off the land	Windows operating system pre-installed and leveraged by state-sponsored threat actors and adversaries to complete their objectives.
Fileless Malware	Authorized software by Windows that threat actors use to perform malicious tasks.
Endpoint Detection and Response (EDR)	Security software that uses machine learning capabilities to detect telemetry on an endpoint.
Large Language Models (LLMs)	Large Language Models use data points from audio, websites, papers, etc. (Examples: ChatGPT4, SecureBert)
Mitre ATT&CK	Framework for detecting threat actors tactics, techniques, and common knowledge of adversaries and state sponsored groups.
Artifact	After an incident changes, modifications, and creations occur on disk.
Token	LLMs process input data, representing it as tokens, which can be either a word, part of a word, or a character, depending on the tokenization method used.

Memory	Random Access Memory Volatile meaning once a computer is rebooted you can't retrieve data in memory.
SecureBERT	Large Language Model specific to Cyber Security.
State Sponsored Actors	Foreign governments are Threat Actors that have ties to communist governments (Russia, China, etc)
Malware Bazaar	Repository of different types of malware samples (exe, powershell, etc)
VMWARE Workstation	Hypervisor allows you to run different operating system in virtualized environment
Ninite	Repository that stores list of applications (7zip, Zoom, etc) (Ninite)
Winpmem	Memory acquisition tool
Pafish	Application that mimics malware detection techniques used by Adversaries to determine if system is a virtualized environment
VmwareHardnerLoader	Tool used to harden your virtual machine against VM detection techniques used by malware by changing vmx file
Office 2021	Suite of tools like (Word, Excel, etc)
SSH	Secure Shell Protocol
Python	Scripting Programming Language
AnyRun	Sandbox environment to upload files, download files, and generate text reports

Abstract

The evolving landscape of cyber threats has seen an increase in adversarial tactics that leverage advanced techniques to exploit vulnerabilities. Fileless malware is becoming increasingly common and poses a significant challenge for detection because it operates primarily in memory, rather than writing files to disk. This memory-based characteristic allows the malware to evade traditional methods of detection, which rely on scanning files found on the system. Previous research into fileless attacks have used methods such as machine learning to help identify and detect signatures found in extracted system memory. However, we feel that large language models have been underutilized in this regard. This project aimed to leverage the SecureBERT large language model that was specifically trained with cybersecurity

information by fine tuning on a set of known malware that employ fileless techniques for identifying samples as either benign or fileless. Our project performed multiple trials resulting in an 87.5% accuracy rating when classifying samples between benign (normal memory) and fileless (malware infected memory). By performing these tests we establish the potential for large language models in this sector of cyber security and avenues for further testing and research.

Problem Statement

Adversaries over the years have been changing their tactics, techniques, and objectives by shifting their focus on prolonging their dwell time on a compromised workstation or server to steal corporate, and classified data about users and companies. Threat actors are developing malicious software that utilizes Windows native operating system tools. For example, PowerShell, Command Prompt, and WMI to name a few. These native operating system tools are known as living off the land binaries and when the lolbins are used with malicious intent, this is known as fileless malware.

The problem that we are currently trying to solve is that security software that exists to detect Fileless malware uses outdated methods, and is reactive, meaning a cyber incident has to occur and artifacts must be dropped on disk before the security solution can detect it. Fileless malware does not drop any executables on disk (Khalid et al., 2023); the malware resides in memory, registry, etc. The legacy methods that security software uses are signature-based detection, static analysis, and dynamic analysis. Signature-based detection is used to determine if a file is classified as malware or not by creating a hash of the file and comparing the hash to the security software predefined database (Khalid et al., 2023). For example, Microsoft defender compares hashes to a predefined database to determine if a file is malicious or not. New sophisticated malware will not have a hash that has already been generated so you can easily evade detection. Static Analysis checks to see if files have been dropped on disk and scans the code of the program to determine what the program is doing. Executables are not dropped on disk and the code for fileless malware is obfuscated using different encoding schemes meaning you can't understand what the program is doing. Dynamic analysis monitors the behaviour of software, but you can't monitor the behaviour of a file if it's only specific to volatile memory which some Fileless malware is. For example, Netwire sample uses powershell. Also, because

fileless malware leverages Microsoft Windows native operating system tools, defender has a difficult time detecting fileless malware because powershell, cmd, doc, docx, xlam, onenote, etc are used to evade detection. Defender can't quarantine powershell because it's native to Windows.

Security vendors over the years have made some improvements in developing security solutions like endpoint detection and response tools to detect telemetry on workstations that can hopefully detect Fileless malware by leveraging artificial intelligence machine learning algorithms. The telemetry that endpoint detection and response security solutions look for is what files have been accessed, network connections, processes, and registry changes that have occurred on the workstation or server (Karantzas & Patsakis, 2021). Endpoint Detection and Response software has a difficult time detecting Fileless malware in memory, andedr software is dependent on whether the machine learning is accurate in detecting file-less attacks by having a accuracy score of 99%(Liu et al., 2024a)

Our topic is important because we want to help the information security community by taking a proactive approach to detecting file-less malware by using the SecureBERT Large Language Model. We have not seen anyone leverage Large Language Models to detect Fileless Malware so this is a new concept.

The current security tools that are being used lack the ability to detect fileless malware. Secure Bert is a large language model that is specific and trained on 5 cyber security threat intelligence. Our goal and objective is to take a proactive approach to this problem by utilizing SecureBERT to determine whether a memory dump is fileless, or benign by using text-classification. Fileless malware is still relevant today. In January, a fileless malware was found in the wild utilizing macros. The threat actors that created the malicious software APT10 who are state sponsored and have ties to China (Room, 2024). A couple of years ago a file-less malware attack using a botnet called Emotet affected 1.6 million workstations (Public Affairs, 2021). Also, ASEC discovered a RAT called Revenge that leveraged native windows operating system tools (ASEC). These articles demonstrate that file-less malware still exists today and will continue to exist in the future.

The capstone contribution table below will provide details on what each group member did when it came to developing the Large Language Model for detecting fileless malware.

Capstone Contribution

Chris	<ul style="list-style-type: none">• Created the Windows 10 Virtual Machine to detonate Malware and Downloaded Remnux.• Downloaded Necessary Tools on Windows 10 and Remnux Virtual Machines(Ninite, Office 2021, Vmware
	<p>Hardener Loader, Pafish)</p> <ul style="list-style-type: none">• Developed python VolatilityAuto.py and commands.yaml code to extract from memory dumps.<ul style="list-style-type: none">◦ Code will loop through the• Added code to the PyAuto_sys.py that will unzip malware with the password of infected using pyzipper.• Created lab guide for Virtual Machines.• Found Large Language Model code edited and added code to fit our needs.• Transferred volatility outputs to csv file that was used to train the Large Language Model.• Ran the Large Language Model locally to train the model.• Created a baseline for determining whether a memory dump was Benign.

Steven	<ul style="list-style-type: none"> Created the PyAuto_sys.py script that performs several key automation tasks: <ul style="list-style-type: none"> Start the script using the command line and enter a number for the folder to be used Script will run and check to see if a zip file is available If password is necessary it will enter the supplied password and unzip the file Script will check for several file types and depending on the type it will either execute or
	<ul style="list-style-type: none"> attempt to open it Once that is completed the script sleeps for a predetermined time ranging from a few minutes to an hour to let malware run completely A file dump will be created using WinPmem That dump will then be transferred to the Remnux VM for processing Created the Host.py script to allow easy reverting to a given snapshot, user can enter command to exit VM

	<ul style="list-style-type: none"> • Debugging and iterations on the scripts <ul style="list-style-type: none"> ◦ Some iterations used task scheduler and windows startup folder to try and achieve full automation ◦ Issues with permissions and timings were found so solution was adjusted • Debugging on the LLM script, created text combining on the script to allow tokenization of multiple categories • Created test bench VMs for running scripts • Testing the LLM locally
Arun	<ul style="list-style-type: none"> • Downloaded Fileless Malware Samples: Acquired relevant samples from online sandboxes like VirusShare and AnyRun. • Configured the Windows VM: Set up a Windows VM with essential tools,
	<p>including Microsoft Office 2021 and Pafish.</p> <ul style="list-style-type: none"> • Installed Forensic Tools: Installed Winpmem for memory dump capture and Volatility for analysis. • Executed Fileless Malware: Successfully detonated malware in the VM, monitoring system behavior closely.

	<ul style="list-style-type: none"> ● Captured and Transferred Memory Dumps: Used Winpmem to create memory dumps and securely transferred them to REMnux VM via SCP. ● Analyzed Memory Dumps: Conducted thorough analysis using Volatility, extracting key indicators of compromise. ● Structured Data for Machine Learning: Organized extracted data into CSV files for machine learning model training. ● Set Up Lab Environment: Configured and maintained the lab environment, optimizing systems for the tasks. ● Troubleshooted and Fixed Errors: Identified and resolved issues during setup and analysis, ensuring smooth operations.
Himanshu	<ul style="list-style-type: none"> ● Procured Fileless Malware Samples: Sourced malware samples from sandboxes like VirusShare and AnyRun.
	<ul style="list-style-type: none"> ● Configured Windows VM: Deployed and equipped the VM with tools like Microsoft Office 2021 and Pafish. ● Deployed Forensic Tools: Set up Winpmem for memory acquisition and Volatility for forensic analysis.

	<ul style="list-style-type: none"> • Executed Malware Payloads: Ran malware in the VM, closely monitoring system activity. • Captured and Transferred Memory Dumps: Generated memory dumps with Winpmem and transferred them securely to REMnux via SCP. • Conducted Forensic Analysis: Used Volatility to analyze memory dumps and extract IOCs. • Prepared Data for ML: Structured the extracted data into CSVs for machine learning training. • Optimized Lab Environment: Configured and fine-tuned the lab environment for optimal performance. • Resolved Technical Issues: Diagnosed and corrected errors during setup and analysis.
--	---

Relevant Literature

The scholarly journal article “A survey on the evolution of fileless attacks and detection techniques” by Liu, et al., 2024 mentions how fileless attacks have grown exponentially over the years from 2018 – 2022 (Liu et al., 2024a). The journal mentions the definition of fileless attacks. The article talks about the different types of fileless attack types and what are the characteristics that define each type. For example, memory only, process injection, registry, etc (Liu et al., 2024b). The rest of the article mentions how you can detect these types of fileless attacks by utilizing various techniques. Also, the article mentions how endpoint detection and response tools do a poor job of detecting file-less malware in memory, and EDR solutions need to be correctly tuned to detect file-less malware attacks (Liu et al., 2024b).

The scholarly journal article “An Insight into the Machine-Learning-Based Fileless Malware Detection” by (Khalid et al., 2023) talks about a research project that Khalid and his

colleagues performed on detecting file-less malware by leveraging multiple machine-learning algorithms. The document mentions how file-less malware evades detection based on security software legacy methods static and dynamic analysis (Khalid et al., 2023). Fileless malware is hard to detect, especially ones that reside in memory current security tools can't detect. The article then talks about their approach to detecting file-less malware attacks by leveraging logistic regression, decision trees, and random forest. The tools that were used with the project were Windows 7 machine, VirtualBox, volatility (Khalid et al., 2023).

The scholarly journal article "An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors" by (Karantzas & Patsakis, 2021) explains how endpoint detection and response software is used to catch malware attacks. The article talks about the telemetry that endpoint detection and response 7 software vendors use to collect on workstations by using agents like file access, modifications, network connections, registry changes (Karantzas & Patsakis, 2021).

The blog post "LodeInfo Fileless Malware Evolves with Anti-Analysis and Remote Code Tricks" by Newsroom talks about how a fileless attack occurred by leveraging macros used on Word documents (Room, 2024) The state-sponsored attackers were backed by the Chinese government and are known as APT-10. Also, known as Advanced Persistent Threat Group 10 (Room, 2024).

As the cybersecurity field continues to expand and evolve, numerous new tools are emerging to enhance the capabilities of cybersecurity professionals in safeguarding sensitive information. Large language models transform how we analyze and interpret data and find patterns to make decisions. Two unique models of potential interest are ChatGP4, one of the most significant, most state-of-the-art large language models available today, and SecureBERT, a model built on top of RoBERTa, which is an offshoot of the popular BERT model adapted to perform text analysis tasks more accurately when dealing with security linguistics. Issues that SecureBert attempts to resolve are semantic in nature. Cyber-specific terms like ransomware, API, and OAuth are not commonly used in everyday English (Aghaei et al., 2022). Additionally, homographs such as virus, handshake, and honeypot may have different meanings when used in the context of cybersecurity compared to their common usage outside this domain (Aghaei et al., 2022). ChatGPT4, on the other hand, is recognized as being domain agnostic, making it adaptable and agile.

Chosen Solutions

The chosen solution for detecting fileless malware is a comprehensive approach that leverages advanced deep learning and generative artificial intelligence techniques, specifically using SecureBERT, a domain-specific Large Language Model (LLM) trained for cybersecurity applications. This solution is designed to address the challenges posed by fileless malware, which typically operates in the system's memory and evades traditional detection methods by using living off the land binaries.

Overview of Fileless Malware Detection using SecureBert LLM

Fileless malware is particularly challenging to detect because it does not rely on traditional files or executables stored on the disk. Instead, it uses the Windows native operating system's tools, like PowerShell, WMI, or other legitimate software components, to carry out malicious activities directly in memory. Traditional security tools often rely on signature-based detection or file analysis, which are ineffective against fileless malware because there are no files to analyze or signatures to match. We will fine tune the SecureBert LLM model by using the task text-classification. The text-classification will help us determine whether a specific memory dump is Fileless or Benign. How we will achieve this is by detonating malware in an isolated environment, acquiring the memory in the isolated environment, transferring memory to another machine and analyzing the memory dump using volatility. (Volatility) The results of the volatility output will be compared to AnyRun sandbox to determine if the sample is leveraging fileless tactic techniques by using living off the land. The volatility output will be transferred to a csv file which will be used to fine tune SecureBertLLM. The SecureBert LLM will make predictions on test split once the data is split using 80% Train and 20% Test.

Key Components of the Solution

Step 1: Acquiring Fileless Malware Sample

To gather fileless malware samples, we utilized the following online platforms known as sandboxes and repositories, where malware is executed in controlled environments to observe its behavior:

1. VirusShare
2. AnyRun
3. PolySwarm
4. Malware Bazaar

After downloading the malware from any of this platform we detonate the malware in the Windows VM (Polser). Before we can analyze the malware, we need a safe environment where the malware can be executed without risking our actual computer. This environment is created using a tool called VMware Workstation Pro.

To set up a virtual environment for malware analysis, first, download and install VMware Workstation Pro. After installation, create an isolated network. The isolated network will use a Host-Only network adapter. Host-Only is a network that is used commonly for detonating malware in an isolated environment. The Host-only network will allow virtual machines to communicate between one another, won't have access to the internet, and packets will not be sent to our physical host. The Screenshot below shows the topology we will be using for our capstone project.

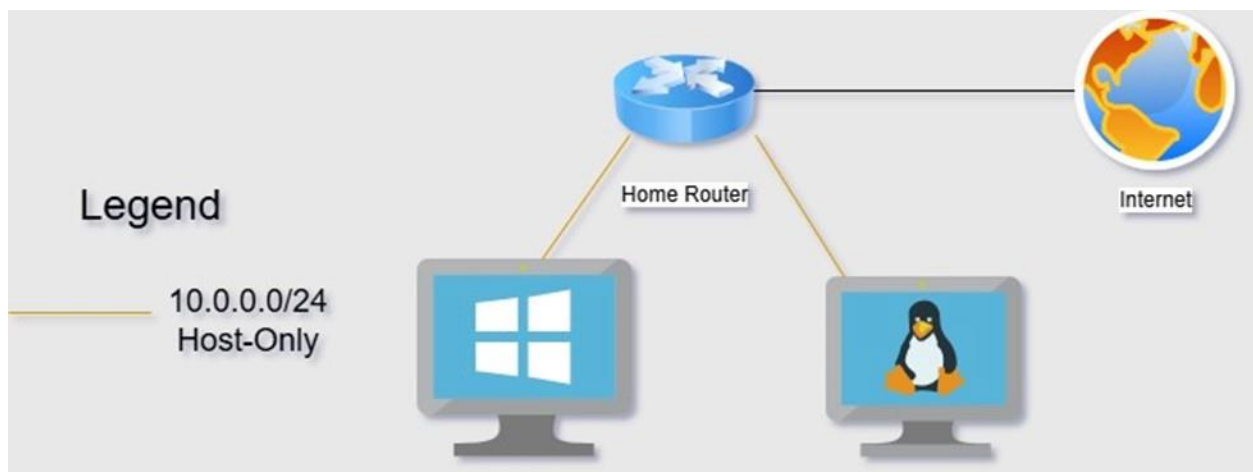


Fig 1 Host-Only Topology (HuskyHacks, 2022)

To create the Host-Only network adapter in VMware Workstation Pro click Edit, Virtual Network Editor, choose Host-Only and edit the Subnet IP to 10.0.0.0 and Subnet Mask to 255.255.255.0

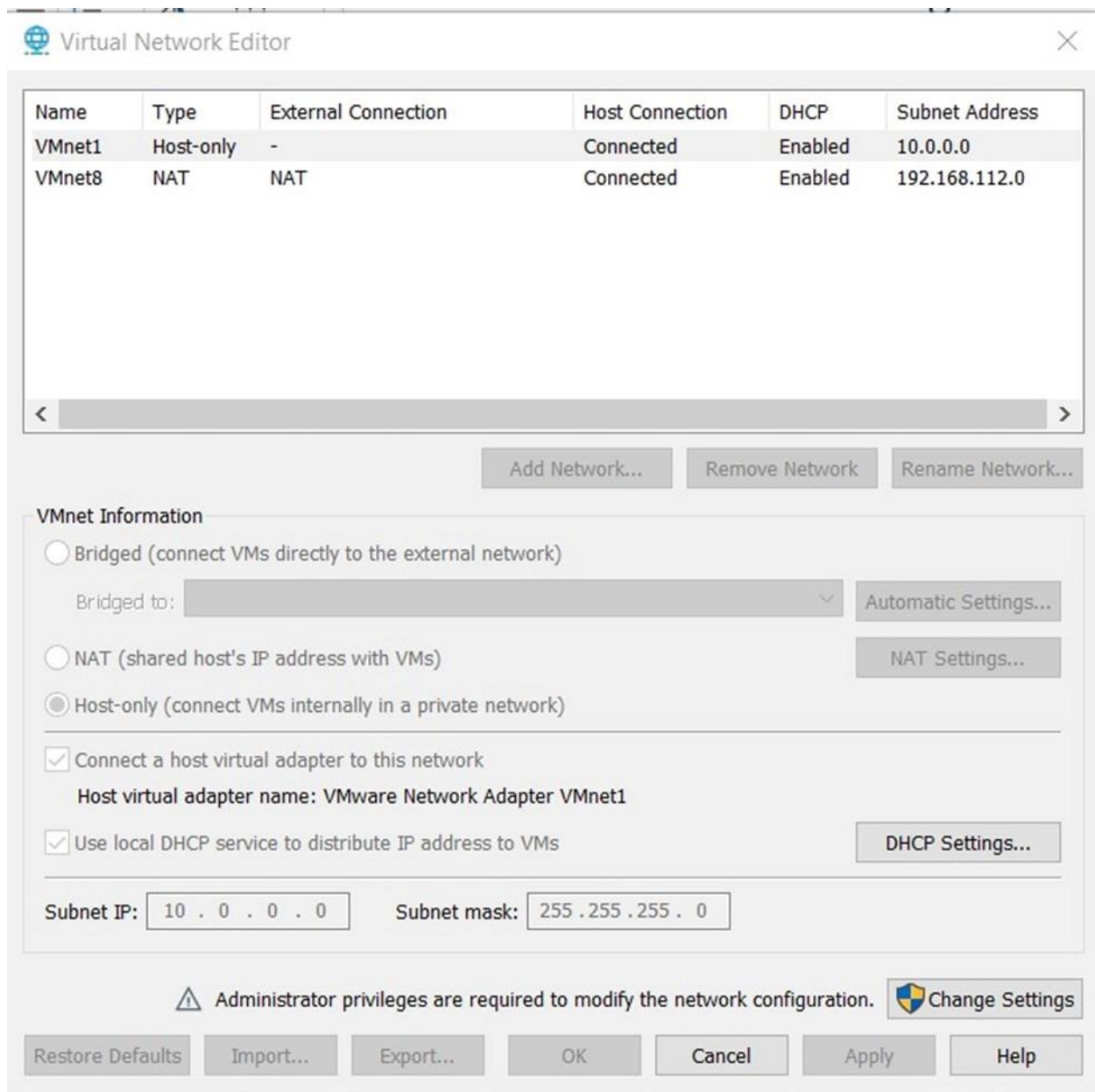


Fig 1.1 Host-Only VMWare Workstation setup

Once the Host-Only network is created you will need to create a new virtual machine by selecting the "Typical" setup option and choosing to install the operating system later. Proceed to install Windows 10 on the VM by selecting "Microsoft Windows" and the appropriate version. Once Windows is fully installed, take a snapshot of the clean system to serve as a baseline for future comparisons after executing malware.

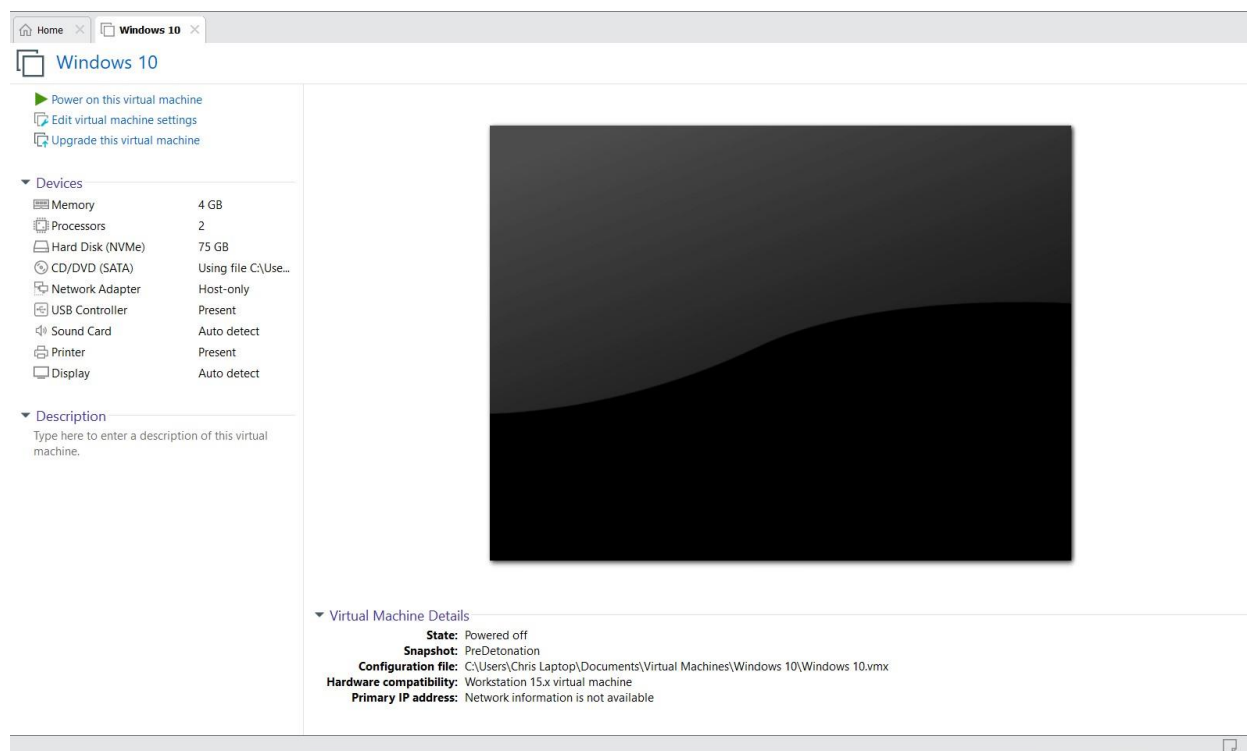


Fig 1.3 Windows 10 Virtual Machine

The Password for Windows 10 VM is: U@ser1122\$

The Password for Remnux is: P@ssw0rd1122

Step 2: Windows 10 VM Setup

- 1. Setup:** The second step is downloading the necessary tools on the windows 10 virtual machine. The tools that we downloaded are winpmem, office2021, pafish, and VmwareHardnerLoader. Winpmem is a command line tool used to capture the full volatile memory(RAM) on a disk (Winpmem, 2024), Office2021 Enterprise is a suite of office tools like Word, Excel, Onenote. (Wisabi Analytics, 2023) Fileless-Malware targets these office products. For example, Emotet uses Excel macros. Pafish is a tool that checks to see if your system is running in a virtualized environment by using known malware virtualization detection techniques. (Labs, 2022) The reason this will help is that when we limit the amount of malware virtualized detection techniques to 0, we will be able to see the full capabilities of the malware being executed in memory.

VmwareHardnerLoader is a tool used to harden the virtual machine against VM detection techniques used by malware by changing up the vmx file of the Virtual Machine, and mac address.(hzzqst, 2018/2024)

Virtual Machine Download Links:

https://sheridanc-my.sharepoint.com/:f/g/personal/mielech_shernet_sheridancollege_ca/Eri6z0x2JzdAik2YaYi3EpABIShHefv8ZTw4B6LqBJ7xvQ?e=lfDPXK

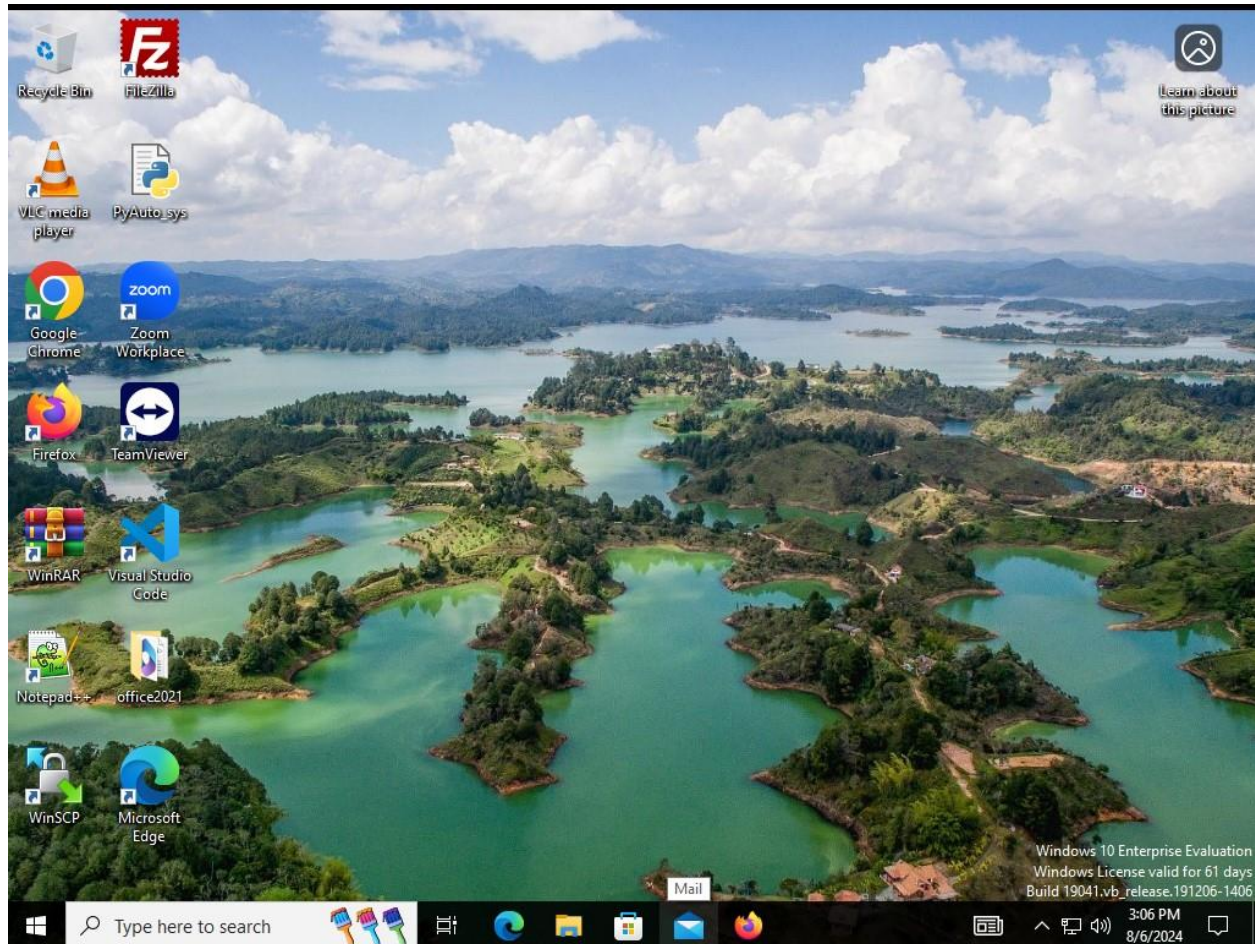
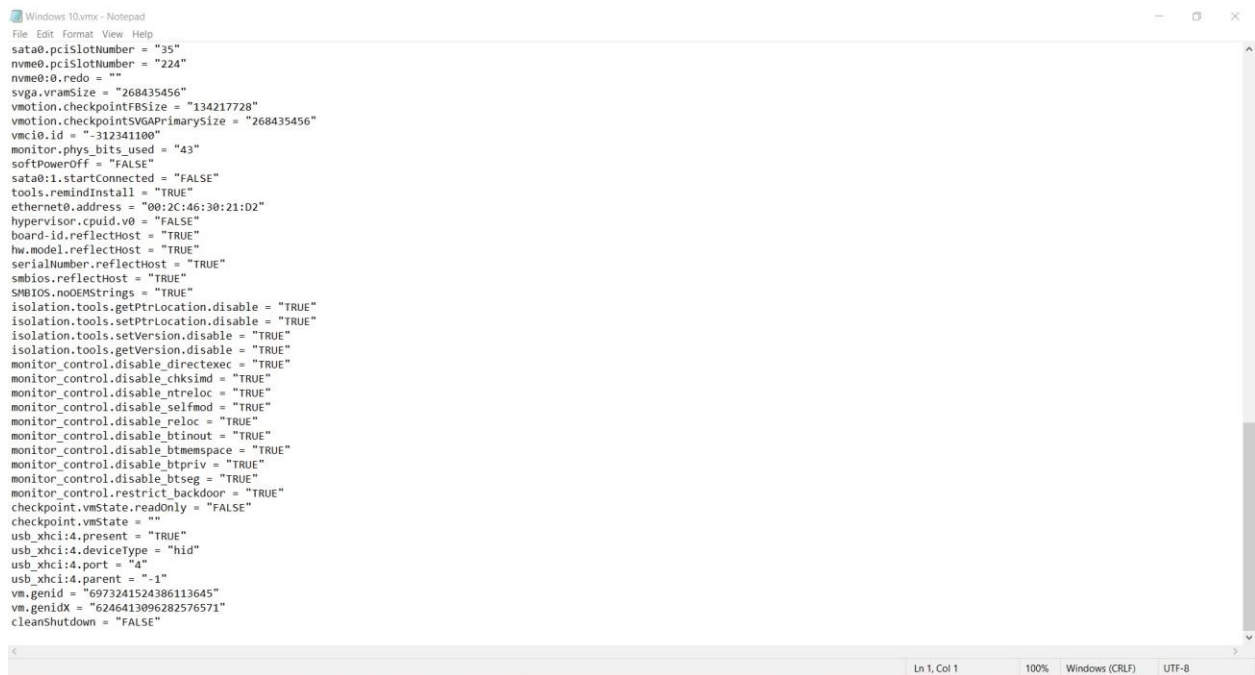


Fig 2 Windows 10 VM Applications



```
File Edit Format View Help
sata0.pciSlotNumber = "35"
nvme0.pciSlotNumber = "224"
nvme0.0.redo = ""
svg8.vramSize = "268435456"
vmotion.checkpointFBSize = "134217728"
vmotion.checkpointSVGAPrimarySize = "268435456"
vmci0.id = "-312341100"
monitor.phys_bits_used = "43"
softPowerOff = "FALSE"
sata0:1.startConnected = "FALSE"
tools.remindInstall = "TRUE"
ethernet0.address = "00:2C:46:30:21:D2"
hypervisor.cpuId.v0 = "FALSE"
board-id.reflectHost = "TRUE"
hw.model.reflectHost = "TRUE"
serialNumber.reflectHost = "TRUE"
smbios.reflectHost = "TRUE"
SMBIOS.noEMStrings = "TRUE"
isolation.tools.getPtrLocation.disable = "TRUE"
isolation.tools.setPtrLocation.disable = "TRUE"
isolation.tools.setVersion.disable = "TRUE"
isolation.tools.getVersion.disable = "TRUE"
monitor_control.disable_directexec = "TRUE"
monitor_control.disable_chksimd = "TRUE"
monitor_control.disable_nrelloc = "TRUE"
monitor_control.disable_selfmod = "TRUE"
monitor_control.disable_reloc = "TRUE"
monitor_control.disable_btinout = "TRUE"
monitor_control.disable_btmemspace = "TRUE"
monitor_control.disable_btpriv = "TRUE"
monitor_control.disable_btseg = "TRUE"
monitor_control.restrict_backdoor = "TRUE"
checkpoint.vmState.readOnly = "FALSE"
checkpoint.vmState = ""
usb_xhci:4.present = "TRUE"
usb_xhci:4.deviceType = "hid"
usb_xhci:4.port = "4"
usb_xhci:4.parent = "-1"
vm.genid = "6973241524386113645"
vm.genidX = "6246413096282576571"
cleanShutdown = "FALSE"
```

Fig 2.1 vmx file edited to include hardened techniques to prevent pafish detection

Step 3: Data Collection Through Memory Dumping:

1. **Memory Dumps:** The second step in detecting fileless malware is to capture the system's memory state at a specific point in time. This is done by creating a memory dump using tools like Winpmem. A memory dump captures the entire contents of a system's RAM, including any malicious activities that may be taking place in memory.

We use **Winpmem** in this project to create memory dumps because it is a powerful and reliable tool specifically designed for capturing the full contents of a system's volatile memory (RAM). This is crucial for analyzing fileless malware, which operates entirely in memory and leaves no traditional file-based traces on disk. By using Winpmem, we can accurately capture the in-memory state of the system at the time of malware execution, enabling detailed forensic analysis and the extraction of key indicators of compromise that are essential for training machine learning models like SecureBERT.

2. **Environment Setup:** To safely execute and analyze fileless malware, a controlled virtual environment (VM Workstation Pro) is used. The malware is executed within this VM, and a memory dump is taken post-execution to capture its behavior.

```
Administrator: Command Prompt
C:\Users\User\Downloads>winpmem_mini_x64_rc2.exe
WinPmem64
WinPmem - A memory imager for windows.
Copyright Michael Cohen (scudette@gmail.com) 2012-2014.

Version 2.0.1 Oct 13 2020
Usage:
  winpmem_mini_x64_rc2.exe [option] [output path]

Option:
  -l Load the driver and exit.
  -u Unload the driver and exit.
  -d [filename]
      Extract driver to this file (Default use random name).
  -h Display this help.
  -w Turn on write mode.
  -0 Use MmMapIoSpace method.
  -1 Use \\Device\\PhysicalMemory method (Default for 32bit OS).
  -2 Use PTE remapping (AMD64 only - Default for 64bit OS).

NOTE: an output filename of - will write the image to STDOUT.

Examples:
winpmem_mini_x64_rc2.exe physmem.raw
Writes an image to physmem.raw

C:\Users\User\Downloads>
```

Fig 3 winpmem command line memory acquisition tool

Step 4: Transferring the memory dump

- In the context of this project, SCP is utilized to transfer memory dumps from the Windows VM, where the fileless malware has been executed, to the REMnux VM(Remnux), which is a specialized environment for malware analysis.

Secure Copy Protocol (SCP) is a command-line tool that allows secure file transfers between two machines over a network, ensuring data encryption during transmission. To send memory dumps from a Windows VM to a REMnux VM, you first need to ensure SCP is installed on both machines. On the Windows VM, prepare the memory dump file, and then use the SCP command from a terminal to initiate the transfer. The command

syntax typically includes specifying the source file, the destination user and host, and the destination path on the REMnux VM.

During the transfer, SCP uses SSH (Secure Shell) for encryption, protecting the data from unauthorized access. Once the command is executed, you'll be prompted for the password of the destination user account on the REMnux VM. After successful authentication, the memory dump file is securely transmitted, and you can verify its presence on the REMnux VM by navigating to the specified directory. This process ensures that sensitive data is securely transferred between different environments.

- **Initiating the Transfer:** On the Windows VM, the memory dump is stored in a specific directory. This file needs to be transferred to the REMnux VM for analysis using Volatility.

Use the SCP command to transfer the file to the REMnux VM. The basic syntax of the SCP command is:

```
scp phymem.raw remnux@10.0.0.130:/home/remnux/Documents/test
```

- In this project, after executing the malware and capturing the memory dump on the Windows VM, the next step is to analyze this memory dump on the REMnux VM.

Step 5: Data Analysis with Volatility:

1. **Volatility Framework:** Once the memory dump is obtained, it is analyzed using the Volatility framework. Volatility is a powerful open-source tool for memory forensics that can extract detailed information about the system's state, such as running processes, open network connections, loaded modules, and more.
2. **Extracting Key Indicators:** Through various Volatility plugins, key indicators of compromise (IOCs) are extracted from the memory dump. These include lists of active processes, details of network connections, and any other suspicious activities that may indicate the presence of fileless malware.
3. **Data Structuring:** The extracted information is structured into CSV files, which serve as a dataset for training machine learning models. This step is crucial as it converts raw memory data into a format that can be effectively used for model training. The table below gives a brief overview of the malware that was detonated, malware family, and file extension

Memory Dump Name	Malware Family Name	File Extension
physmem.raw and physmem3.raw	WannaCry	.exe
physmem5.raw and physmem7.raw	Kovter	.exe
physmem9.raw and physmem11.raw	Grandcrab	.exe
physmem13.raw and physmem15.raw	Netwire	.exe
physmem17.raw and physmem19.raw	Grandcrab	.doc
physmem20.raw and physmem22.raw	Netwire	.ps1
physmem24.raw and physmem26.raw	Novter	.exe
physmem28.raw and physmem30.raw	Nodster	.js
physmem32.raw and physmem34.raw	PowDow	.xlam
physmem36.raw and physmem38.raw	Fin7	.doc

physmem40.raw and physmem42.raw	Jeki	.doc
physmem44.raw and physmem46.raw	SysJoker	.exe
physmem48.raw and physmem50.raw	SysJoker	.exe
physmem52.raw and physmem54.raw	Revil	.exe
physmem56.raw and physmem58.raw	Lockbit	.exe

The Link below is the Google Docs which has the Malware Samples we analyzed for our Capstone Project, Analysis of memory dumps using volatility.

Google Docs:

[Fileless Malware - Google Docs](#)

Analyzed Malware:

[Malware Raw Files](#)

```
import yaml
import subprocess

#Output Banner
def Banner():
    print("Volatility Automation Script")

#Load yaml file
def yaml_load(file_path):
    with open(file_path, 'r') as file:
        return yaml.safe_load(file)

#Execute Subprocess commands
def execute_command(command):
    subprocess.run(command, shell=True, check=True, text=True, stdout=subprocess.PIPE, stderr=subprocess.PIPE)

#Main function
def main():
    config_file = yaml_load('commands.yaml')
    commands = config_file.get('commands', [])
    for i in commands:
        execute_command(i)

if __name__ == "__main__":
    Banner()
    main()
```

Fig 5 Volatility Automation Script

commands.yaml

```
1  commands:
2  - vol.py -f physmem.raw --profile=Win10x64_19041 pslist > physmem.pslist.csv
3  - vol.py -f physmem.raw --profile=Win10x64_19041 pstree > physmem.pstree.csv
4  - vol.py -f physmem.raw --profile=Win10x64_19041 psxview > physmem.psxview.csv
5  - vol.py -f physmem.raw --profile=Win10x64_19041 dlllist > physmem.dlllist.csv
6  - vol.py -f physmem.raw --profile=Win10x64_19041 handles > physmem.handles.csv
7  - vol.py -f physmem.raw --profile=Win10x64_19041 ldrmodules > physmem.ldrmodules.csv
8  - vol.py -f physmem.raw --profile=Win10x64_19041 svcsan > physmem.svcsan.csv
9  - vol.py -f physmem.raw --profile=Win10x64_19041 cmdline > physmem.cmdline.csv
```

Fig 5.1 volatility plugins being sent to csv

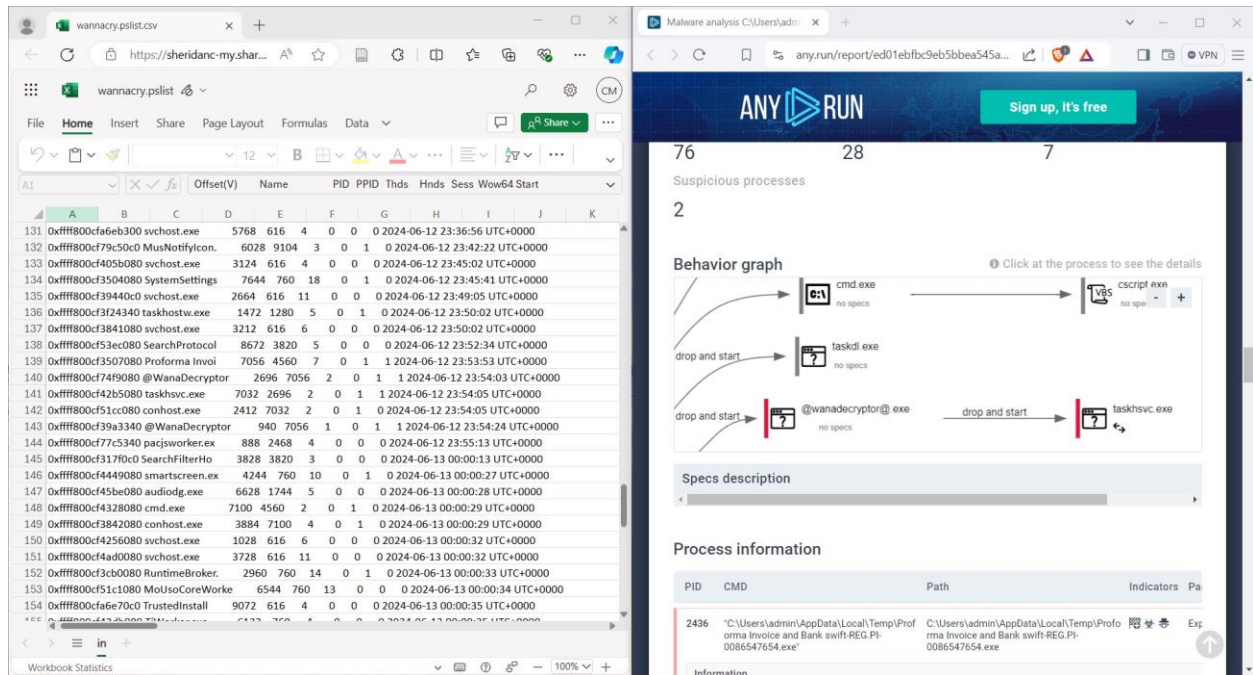


Fig 5.2 Comparing Results to AnyRun

<https://any.run/report/ed01ebfbc9eb5bbea545af4d01bf5f1071661840480439c6e5babe8e080e41aa/b28a4f68-c06b-40dc-8d8a-8b0df1ab75a3>

Step 6: Host.py Script:

Host.py script when run will look for vmrun, look for Windows 10.vmx file, and revert Windows 10 vmx file to a predetonation snapshot. Also, the code will allow you to stop vm by entering the number "1" in terminal.

```

45
46 #Variable Setup
47 vmrun_path = "C:\\Program Files (x86)\\VMware\\VMware Workstation\\vmrun.exe"
48 vm_path = "C:\\Users\\Chris Laptop\\Documents\\Virtual Machines\\Windows 10\\Windows
49 snapshot_name = "PreDetonation" #Pre Detonation
50
51 #Check if vmrun_path is correct
52 if not os.path.isfile(vmrun_path):
53     print(f"vmrun executable not found at: {vmrun_path}")
54     sys.exit(1)
55 #Check if vm_path is correct
56 if not os.path.isfile(vm_path):
57     print(f"VMX file not found at: {vm_path}")
58     sys.exit(1)
59

```

Fig 6 Host.py code snippet

Step 7: Model Training with SecureBERT:

- Training Process:** The CSV files generated from the Volatility analysis are fed into SecureBERT for training. The model learns to recognize patterns, tactics, techniques, and behaviors associated with fileless malware. It uses these learned patterns to detect similar activities in future datasets.
- Specialized LLM:** SecureBERT is a specialized Large Language Model designed to process and analyze cybersecurity-related data. Unlike general-purpose models, SecureBERT is trained on a vast corpus of cybersecurity texts, including threat reports, security blogs, and technical documentation, making it particularly adept at understanding the context and nuances of cybersecurity threats.

Name	pslist	pstree	psxview	dlllist	handles	ldrmodules
phymem.raw	proforma invoice and bank swift-reg.pi-0086547654.exe	The parent process is proforma invoice and bank swift-reg.pi-0086547654.exe	proforma invoice and bank swift-reg.pi-0086547654.exe @WanaDecryptor.exe taskhsvc.exe taskse.exe	ADVAPI32.dll KERNEL32.dll MSVCRT.dll USER32.dll	proforma invoice and bank swift-reg.pi-0086547654.exe @WanaDecryptor.exe taskhsvc.exe taskse.exe	ADVAPI32.dll KERNEL32.dll MSVCRT.dll USER32.dll
	swift-reg.pi-0086547654.exe	The child process of proforma invoice and bank swift-reg.pi-0086547654.exe is @WanaDecryptor.exe				
	@WanaDecryptor.exe	The child process of proforma invoice and bank swift-reg.pi-0086547654.exe is taskse.exe				
	@WanaDecryptor.exe	The child process of process is proforma invoice and bank swift-reg.pi-0086547654.exe is taskhsvc.exe				
	taskhsvc.exe	The parent process is cmd.exe				
	cmd.exe	The child process of cmd.exe is taskhsvc.exe				

svcsan	cmdline	Category
vssvc.exe	proforma invoice and bank swift-reg.pi-0086547654.exe executed the Command line : "C:\\Users\\User\\Downloads\\Proforma Invoice and Bank swift-REG.PI-0086547654.exe"	Fileless
wbengine.exe	@WanaDecryptor executed the Commnad line : @WanaDecryptor@.exe co	
	taskhsvc.exe executed the Command line : TaskData\\Tor\\taskhsvc.exe	
	conhost.exe executed the Command line : \\??C:\\Windows\\system32\\conhost.exe 0x4	
	\\??C:	
	@WanaDecryptor executed the Command line : @WanaDecryptor@.exe	

physmem2.raw	explorer.exe	System smss.exe wininit.exe RuntimeBroker.exe taskhostw.exe winlogon.exe csrss.exe service.exe svchost.exe lsass.exe The parent process is System The child process of System is smss.exe The parent process is smss.exe The child process of smss.exe is wininit.exe The parent process is svchost.exe The child process of svchost.exe is RuntimeBroker.exe The parent process is svchost.exe The child process of svchost.exe is taskhostw.exe The parent process is smss.exe The child process of smss.exe is winlogon.exe The parent process is smss.exe The child process of smss.exe is csrss.exe The parent process is service.exe The child process of service.exe is svchost.exe The parent process is wininit.exe The child process of wininit.exe is lsass.exe	System smss.exe wininit.exe RuntimeBroker.exe taskhostw.exe winlogon.exe csrss.exe service.exe svchost.exe lsass.exe explorer.exe The parent process is System The child process of System is smss.exe The parent process is smss.exe The child process of smss.exe is wininit.exe The parent process is svchost.exe The child process of svchost.exe is RuntimeBroker.exe The parent process is svchost.exe The child process of svchost.exe is taskhostw.exe The parent process is smss.exe The child process of smss.exe is winlogon.exe The parent process is smss.exe The child process of smss.exe is csrss.exe The parent process is service.exe The child process of service.exe is svchost.exe The parent process is wininit.exe The child process of wininit.exe is lsass.exe	ADVAPI32.dll KERNEL32.dll USER32.dll ntdll.dll	System smss.exe wininit.exe RuntimeBroker.exe taskhostw.exe winlogon.exe csrss.exe service.exe svchost.exe lsass.exe explorer.exe The parent process is System The child process of System is smss.exe The parent process is smss.exe The child process of smss.exe is wininit.exe The parent process is svchost.exe The child process of svchost.exe is RuntimeBroker.exe The parent process is svchost.exe The child process of svchost.exe is taskhostw.exe The parent process is smss.exe The child process of smss.exe is winlogon.exe The parent process is smss.exe The child process of smss.exe is csrss.exe The parent process is service.exe The child process of service.exe is svchost.exe The parent process is wininit.exe The child process of wininit.exe is lsass.exe	ADVAPI32.dll KERNEL32.dll USER32.dll ntdll.dll
SamSs	KeyIso	smss.exe executed the Command line : \SystemRoot\smss.exe wininit.exe executed the Command line : wininit.exe RuntimeBroker executed the Command line : C:\Windows\System32\RuntimeBroker.exe -Embedding taskhostw.exe executed the Command line : taskhostw.exe {222A245B-E637-4AE9-A93F-A59CA119A75E} winlogon.exe executed the Command line : winlogon.exe csrss.exe executed the Command line : %SystemRoot%\system32\csrss.exe ObjectDirectory=\Windows SharedSection=1024 services.exe executed the Command line : C:\Windows\system32\services.exe svchost.exe executed the Command line : C:\Windows\system32\svchost.exe -k DcomLaunch -p lsass.exe executed the Command line : C:\Windows\system32\lsass.exe explorer.exe executed the Command line : C:\Windows\Explorer.EXE				Benign

Fig 7 csv file results

Excel Download Link: [Fileless-Results2.xlsx](#)

The baseline we used was finding the pre detonation process, child process, dlls, psxview, handlers, ldrmodules, svcscan, cmdline.

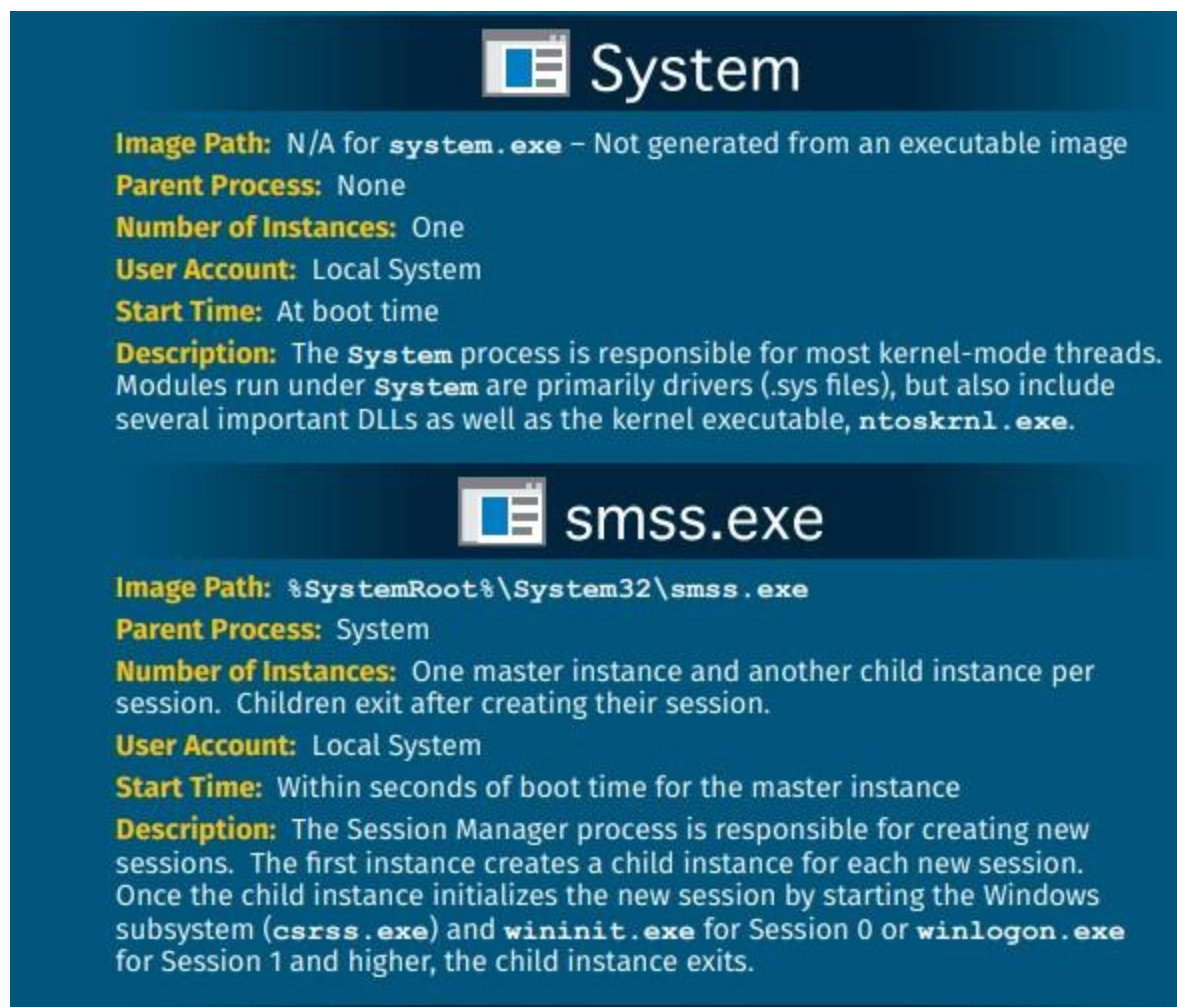


Fig 7.1 Processes Sans Hunt Evil (SANS, 2024)

```
CSV FILE SETUP

csv = "Fileless-ResultsNew2.csv" #CSV File Name Has Data On It
text_column_name = "Name" #Column Name
text_column_name_1 = "pslist" #Column pslist
text_column_name_2 = "pstree" #Column pstree
text_column_name_3 = "psxview" #Column psxview
text_column_name_4 = "dlllist" #Column dlllist
text_column_name_5 = "handles" #Column handles
text_column_name_6 = "ldrmodules" #Column ldrmodules
text_column_name_7 = "svcsan" #Column svcsan
text_column_name_8 = "cmdline" #Column cmdline
label = "Category" #Label Column
model_name = "ehsanaghaei/SecureBERT" #Model Chosen Hugging Face
test_size = 0.2 #Test Size Split 20% Test Train 80%
num_labels = 2 #Fileless and Benign
```

Fig 7.2 Screenshot shows CSV File Setup. CSV file we used, columns, Label Category, Large Language Model Name, Data Split 80% Train and 20% Test, and the labels are Fileless and Benign(Code in a Jiffy, 2023)

```

tokenizer = AutoTokenizer.from_pretrained(model_name)

def preprocess_function(examples):
    combined_texts = []
    for Name,pslist,pstree,psxview,dllist,handles,ldrmodules,svcsan,cmdline in zip(
        examples["Name"], examples["pslist"], examples["pstree"], examples["psxview"], examples["dllist"], examples["handles"], examples["ldrmodules"], examples["svcsan"],
    ):
        combined_text = f"{Name} {pslist} {pstree} {psxview} {dllist} {handles} {ldrmodules} {svcsan} {cmdline} "
        combined_texts.append(combined_text)

    encoding = tokenizer(
        combined_texts,
        padding = 'max_length',
        truncation = True,
        return_tensors = None,
        return_attention_mask=True,
        return_token_type_ids=True
    )

    return {
        'input_ids': encoding['input_ids'],
        'attention_mask': encoding['attention_mask'],
        'token_type_ids': encoding.get('token_type_ids', [None] * len(combined_texts))
    }

```

Fig 7.3 Columns that are AutoTokenized taking plain text and converting to vectors

```

training_args = TrainingArguments(
    output_dir="./result",
    learning_rate=2e-5,
    per_device_train_batch_size=12,
    per_device_eval_batch_size=12,
    num_train_epochs=5,
    weight_decay=0.01,
    evaluation_strategy = "epoch",
    logging_strategy="epoch",
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics
)

```

Fig 7.4 Shows LLM FineTuning Training Rate

Learning_rate = 2e-5 How fast model can learn from data smaller the rate better result

per_device_train_batch_size = 12 GPU process 12 samples from data at a time at once

per_device_eval_batch_size = 12 GPU will process 12 samples from evaluation at

once num_train_epochs = 5 number of times entire data will be trained weight_decay =

0.01 Prevents overfitting by adding errors

Also, if you have trouble running the script locally and receive “ImportError Using the Trainer with PyTorch requires accelerate>=0.20.1 you must restart the kernel for jupyter notebook.

(Transformers, 2023)

Step 7: Continuous Improvement:

- **Iterative Training:** The solution is designed to be continuously improved. As new malware samples are collected and new techniques are observed, the model can be retrained to maintain its effectiveness against evolving threats.

Advantages of the Chosen Solution

- **Enhanced Detection:** By focusing on memory analysis and using a model specifically trained for cybersecurity, the solution can detect fileless malware that would otherwise go unnoticed by traditional security tools.
- **Context-Aware Analysis:** SecureBERT's ability to understand the context of threats and map them to the MITRE ATT&CK framework provides deeper insights into the nature of the threats, aiding in more effective response strategies.
- **Proactive Security Posture:** This solution allows organizations to adopt a more proactive security posture by detecting and mitigating threats before they can cause significant damage.

Conclusion

The chosen solution leverages advanced machine learning, specifically SecureBERT, to tackle the sophisticated threat of fileless malware. By combining memory forensics with the power of LLMs, this approach offers a robust and scalable method for detecting and understanding fileless malware, thereby enhancing the overall security posture of organizations.

Detailed Justification for the Chosen Solution

The chosen solution for detecting fileless malware involves integrating memory forensics with advanced machine learning techniques. This approach is carefully designed to address the unique challenges posed by fileless malware, which traditional detection methods often fail to identify. Below is a detailed justification for each component of the solution:

1. Memory Forensics

Tools Used: Winpmem and Volatility

- **Nature of Fileless Malware:** Fileless malware operates entirely in the system's memory

(RAM) without leaving traditional file-based artifacts on disk, making it invisible to many conventional antivirus tools that rely on signature-based detection. Since this type of malware leverages legitimate system processes, it can evade detection by standard security measures that do not inspect in-memory activities.

- **Why Winpmem?**

- **Full Memory Capture:** Winpmem is a robust tool specifically designed to capture the entire contents of a system's volatile memory. This includes all active processes, system states, and any malicious code residing in RAM at the time of the capture. For fileless malware detection, capturing this data is critical because the malware's footprint exists solely in memory.
- **Forensic Integrity:** Winpmem provides a forensically sound method of capturing memory, ensuring that the data can be analyzed without being altered. This is essential for maintaining the accuracy and reliability of the forensic investigation.

- **Why Volatility?**

- **Comprehensive Analysis:** Volatility is a powerful open-source framework for memory forensics. It allows for the extraction of detailed information from memory dumps, including running processes, open network connections, loaded DLLs, and more. This granular level of analysis is necessary to detect the subtle signs of fileless malware, which may not be immediately obvious.
- **Support for Multiple Plugins:** Volatility supports a wide range of plugins, each designed to extract specific types of information from memory dumps. This modularity enables a comprehensive analysis of the memory state, helping to identify all possible indicators of compromise (IOCs) associated with fileless malware.

2. Machine Learning with SecureBERT

Tool Used: SecureBERT

- **Limitations of Traditional Detection Methods:**

- **Signature-Based Detection:** Traditional antivirus solutions rely on signature-based detection, which matches known patterns (signatures) of malware against files on the system. However, fileless malware does not create traditional files, so there are no signatures to match, rendering this method ineffective.

- **Behavioral Analysis Limitations:** While some advanced security tools use behavioral analysis to detect malware based on its actions, these methods can be circumvented by fileless malware that mimics legitimate processes or hides within them.
- **Why SecureBERT?**
 - **Cybersecurity-Specific Training:** SecureBERT is a Large Language Model (LLM) specifically trained on cybersecurity-related data, such as threat intelligence reports, technical documentation, and security blogs. This training gives SecureBERT a deep understanding of cybersecurity terminology, tactics, and techniques, making it highly effective at analyzing and detecting threats in this domain.
 - **Pattern Recognition:** SecureBERT excels at recognizing complex patterns in data, which is crucial for detecting fileless malware. By analyzing the structured data extracted from memory dumps, SecureBERT can identify patterns of behavior and techniques associated with known malware, even when those behaviors are subtle or obfuscated.
 - **Adaptability and Learning:** Unlike traditional detection methods that rely on predefined rules or signatures, SecureBERT learns from data. This means it can be continuously retrained and updated with new information about emerging threats, ensuring that it remains effective against the latest techniques used by cyber attackers.
- **Integration with MITRE ATT&CK:**
 - **Tactics and Techniques Mapping:** SecureBERT's integration with the MITRE ATT&CK framework allows it to map detected behaviors to known adversarial tactics and techniques. This not only helps in detecting the malware but also provides context on how the malware operates, which is invaluable for incident response and threat hunting teams.

3. Secure Data Handling with SCP Tool Used: Secure Copy Protocol (SCP)

- **Importance of Secure Data Transfer:**
 - **Sensitivity of Memory Dumps:** Memory dumps contain all the data stored in a system's RAM, which can include sensitive information such as encryption keys, passwords, and active session data. It is critical to ensure that this data is transferred securely to prevent unauthorized access.

- **Integrity and Confidentiality:** During transfer, it is essential to maintain the integrity and confidentiality of the memory dumps. Any tampering or exposure during transfer could compromise the forensic analysis or lead to a data breach.
- **Why SCP?**
 - **Encryption:** SCP uses SSH (Secure Shell) to encrypt the data being transferred. This ensures that the memory dump is protected against interception or unauthorized access during transit, which is crucial given the sensitivity of the data.
 - **Reliability and Simplicity:** SCP is a reliable tool that is easy to use, especially in environments where SSH is already implemented. Its simplicity makes it an ideal choice for securely transferring files between different systems, such as from a Windows VM to a REMnux VM.
 - **Cross-Platform Compatibility:** SCP works seamlessly across different operating systems, making it versatile for use in diverse environments. This compatibility is important in a project where data needs to be transferred between different types of virtual machines.

4. Python Automation

Tools Used: Python, vmrun utility, Volatility, SCP

- **Importance:** Certain actions in the project were automated using python to speed up the detonation and analysis of the malware samples
- **Time Efficiency:** Since we were working on a limited timeframe and the difficulty of locating a pre existing database of fileless malware we needed a way to expedite the information gathering process
- **Flexibility:** Tools such as Volatility, SCP, and vmrun can be automated using Python scripts by executing them as command-line commands on Windows

Impact

The chosen solution for detecting fileless malware, which integrates memory forensics and machine learning through SecureBERT, has several far-reaching impacts on the cybersecurity landscape. This detailed impact analysis explores how this solution enhances security, improves detection accuracy, supports incident response, and contributes to the broader field of cybersecurity.

1. Enhanced Detection Capabilities

- **Detecting Stealthy Threats:**

- Traditional security tools often struggle to detect fileless malware due to its lack of file-based artifacts and its ability to blend with legitimate system processes. By focusing on memory forensics, this solution captures the volatile state of the system, including in-memory activities where fileless malware operates. The use of Winpmem ensures that all memory data is captured, while Volatility provides the means to analyze this data for signs of malicious activity.
- SecureBERT, trained specifically on cybersecurity data, enhances the detection of subtle and complex patterns that are indicative of fileless malware. It can recognize unusual behaviors and tactics that would be missed by signature-based detection methods, thereby significantly improving the chances of early detection.

- **Improved Accuracy and Reduced False Positives/Negatives:**

- The traditional methods often generate false positives, where legitimate activities are flagged as malicious, or false negatives, where actual threats are missed.

SecureBERT's advanced pattern recognition capabilities reduce these errors by providing context-aware analysis. This means that the model not only looks at isolated indicators but also understands the broader context of system behavior, leading to more accurate threat detection.

2. Proactive Security Posture

- **Early Threat Detection:**

- By analyzing memory dumps and detecting threats based on behavioral patterns rather than static signatures, the solution allows organizations to identify fileless malware at an early stage. This early detection is crucial for preventing the malware from achieving its objectives, such as exfiltrating data or establishing persistent access.
- The proactive nature of this detection method shifts the organization's security posture from reactive (responding to incidents after they occur) to proactive (identifying and mitigating threats before they can cause harm).

- **Continuous Learning and Adaptation:**

- SecureBERT can be continuously retrained with new data and emerging threats, ensuring that the detection capabilities evolve alongside the threat landscape.

This adaptability is critical in the cybersecurity field, where new attack vectors and techniques are constantly being developed by adversaries.

- The ability to update and refine the model means that organizations using this solution are better equipped to handle not just current threats but also future, more sophisticated attacks.

3. Support for Incident Response and Threat Hunting

- **Detailed Forensic Insights:**

- The forensic analysis provided by Volatility offers deep insights into the system's memory state at the time of the malware's execution. This includes information about running processes, network connections, and other in-memory activities. These insights are invaluable for incident response teams who need to understand the scope and impact of an attack.
- By mapping the detected behaviors to the MITRE ATT&CK framework, SecureBERT provides a clear picture of the tactics, techniques, and procedures (TTPs) used by the malware. This detailed understanding helps incident responders and threat hunters develop targeted remediation strategies and prevents future attacks using similar methods.

- **Improving Response Times and Effectiveness:**

- With a clear understanding of the malware's behavior and the techniques it uses, security teams can respond more quickly and effectively to incidents. This reduces the time it takes to contain and eradicate the threat, minimizing potential damage.
- The solution also enables security teams to prioritize their efforts by identifying the most critical threats and focusing resources where they are needed most.

Major Activities Carried Out for Solving the Problem

The process of solving the problem of detecting fileless malware involved several key activities, each essential for ensuring a comprehensive and effective solution. Here's an overview of the major activities, tools used, and the implementations carried out during this project:

1. Problem Analysis and Research

- **Objective:** Understand the unique challenges posed by fileless malware, which operates entirely in system memory and evades traditional detection methods.
- **Research:** Studied existing literature and methodologies on fileless malware detection, focusing on the limitations of traditional tools and the potential of using machine learning models like SecureBERT for detection.
- **Outcome:** Formulated a clear strategy that leverages memory forensics and machine learning to detect fileless malware.

2. Setting Up the Virtual Environment

- **Tools Used:** VMware Workstation Pro, Windows 10 VM, REMnux VM.
- **Implementation:**
 - Created isolated virtual environments where malware could be safely executed without risking the host system.
 - Set up a clean Windows 10 virtual machine for executing fileless malware.
 - Configured a REMnux VM to serve as the analysis platform.
- **Outcome:** A controlled environment ready for malware detonation and forensic analysis.

3. Executing and Capturing Malware Activity

- **Tools Used:** Winpmem, Python script for simulating user activity.
- **Implementation:**
 - Imported and executed fileless malware samples in the Windows VM to trigger their in-memory activities.
 - Ran a Python script to simulate mouse movement, ensuring the system remained active during malware execution.
 - Captured the system's memory state by creating raw memory dumps using Winpmem.
- **Outcome:** Obtained memory dumps that contain the operational footprint of the fileless malware.

4. Secure Data Transfer

- **Tools Used:** SCP (Secure Copy Protocol).
- **Implementation:**

- Transferred the captured memory dumps from the Windows VM to the REMnux VM using SCP.
- Ensured that the transfer process was secure and the integrity of the data was maintained during transit.
- **Outcome:** Securely transferred memory dumps ready for detailed analysis on REMnux.

5. Forensic Analysis of Memory Dumps

- **Tools Used:** Volatility Framework.
- **Implementation:**
 - Analyzed the memory dumps using Volatility to extract key indicators of compromise, such as running processes, network connections, and other in-memory activities.
 - Structured the extracted data into CSV files for further processing.
- **Outcome:** Detailed forensic data in CSV format, capturing the behavioral footprint of the fileless malware.

6. Training the Machine Learning Model

- **Tools Used:** SecureBERT, Python.
- **Implementation:**
 - Used the CSV files generated from the Volatility analysis to train SecureBERT, a machine learning model specifically designed for cybersecurity tasks.
 - The model was trained to recognize patterns and behaviors typical of fileless malware based on the data extracted from the memory dumps.
- **Outcome:** A trained SecureBERT model capable of detecting fileless malware by analyzing similar memory dumps in future scenarios.

7. Validation and Testing

- **Tools Used:** Test datasets, SecureBERT.
- **Implementation:**
 - Validated the performance of SecureBERT using test datasets to ensure its accuracy in detecting fileless malware.
 - Fine-tuned the model to reduce false positives and negatives, improving its reliability and effectiveness.

- **Outcome:** A validated and optimized model ready for deployment in real-world scenarios to enhance cybersecurity defenses.

Major deliverables of the project:

1. **Datasets:** One of the major deliverables of the project is Datasets, We've used multiple different malware and detonated them in a safe virtual environment. The datasets are the main component of our project as those will help us in training our LLM and testing it. The testing dataset, drawn from sources like VirusShare, AnyRun, and PolySwarm, contains samples representing both malware and non-malware instances. It serves to evaluate the machine learning model's ability to distinguish between these classes, assessing metrics like accuracy, precision, and recall before deployment.

Meanwhile, the training dataset, also sourced from these platforms, is balanced to ensure a comprehensive representation of fileless malware and benign files. Through extensive training on this dataset, the model learns the nuances of fileless malware, enhancing its detection capabilities and enabling effective generalization to new, unseen instances. Together, these datasets provide a robust foundation for developing and validating a reliable fileless malware detection system.

The creation of the dataset is majorly dependent on the nature of the malware, As with fileless malware it's difficult to get just the fileless malware there are samples included for the malware that depict the characteristics of fileless malware which not only help with increasing the datasets but also in providing more wide variety of data. Once we have enough data to train and test the model it'll help in improving the accuracy of LLM.

2. **Finalized Output Data:** Once we have created the datasets, the next major deliverable is output data generated from the testing of LLM or the final result. As with any other LLM solution, the more data you test it on the better results will be. In our case as well, once we started training the LLM with the low amount of data we were getting a very low number of accuracy, As we started increasing the datasets the accuracy number also started improving.

The given nature of fileless malware and its not having its classification makes it very hard to gather samples of just fileless malware. In our effort to improve the accuracy score, we trained LLM on more data and got better accuracy than in the past.

When this LLM is used on the new fileless malware with the training data we used it has a good chance of detecting whether the sample that is provided is benign or malware. With this deliverable, we'll be providing the accuracy rate with which the LLM can detect whether the sample is malware or benign.

3. **Automation Scripts:** There were several automation scripts created for different purposes in the project. The host script used the vmrun command-line utility in workstation pro. This command allows for the starting and stopping of the VM through scripting and can reset a virtual machine using a specified snapshot.

The python script that runs on the virtual machine itself is responsible for the process between locating the malware sample to sending the generated memory dump file to REMnux for analysis. The user runs the script through the command line and enters a number that is used when searching through a predefined folder structure. The script checks the folder and checks for a compressed file type. If one is found then the next step is to attempt to unzip the compressed file. Each file is compressed with the same password so that is defined in the script. The file is decompressed using the password resulting in the file containing the malware. The script retrieves the file type (e.g., .ps1, .xlsx, .docx) and then performs different functions based on whether the file is executable , runnable or needs to be opened with specific software. The malware sample is now running on the VM and a timer can be set to allow the sample enough time to conduct its malicious activities. Once this time has elapsed the file is sent to the REMnux machine.

On the REMnux machine another script is used to run several volatility commands to retrieve the information needed to populate the .csv file. Once enough samples are gathered we are able to begin using this file to finetune the Large Language Model.

4. **Final Report (Week 13):** Finalizing and generating the solution report we received from our Large Learning Model in terms of True Positive Rate(TPR) and False Positive Rate(FPR). With this report, we'll be including all the information in regards to our project

outlining the process starting from setting up the lab to getting the results from LLM. This report will also feature the issues that we got while finishing up the project and how we overcame those issues.

This report will be a great stepping stone for everyone as a start on the detection of fileless malware and also work as a great asset in resolving some of the issues that we had to deal with, without having anything to reference. It'll also help with providing a step-by-step guide to simple stuff like setting up the lab environment, getting the malware samples, and much more.

Software, tools, systems, or other technology employed in the project

Multiple Tools were employed throughout the course of this project which helped us in getting results. We'll be mentioning all the tools and if there's any specific ones that are required for using it.

1. **Vm Workstation Pro:** When working with malware it's really important to keep the host machine safe. We used Vm Workstation Pro, which not only helps with keeping the host safe but also provides the ability to snapshot the virtual machines. Snapshot helps us with the ability to go back to the same state of the virtual machine where we started.

Another big factor to keep in mind is while we're using a VM workstation there is still malware with the ability to escape the VM so make sure we're not using the malware which has the capability of escaping. VM workstation also provides us with an easy way to change between network configurations making it easier for us to maintain different network types like keeping an internet-connected interface just for analysis of VM.

2. **Windows VM:** We used the latest Windows version for exploiting the malware and created a memory dump file of the VM. This helped us gather all the necessary footprints of the malware and things that are changed by its nature like Registry keys, processes, etc.

3. **Remnux VM:** After getting the memory dump from Windows VM to analyze it rather than using our host machine we used another VM with Linux-based operating system Remnux running on it. We wanted to minimize all the interaction between the Windows VM and the host machine, this VM gives us a good opportunity to perform analyses on raw files without having any communication with the host machine.

3. **WinPmem:** After getting the VM setup another major tool required is to get the memory acquisition of the complete machine. For this, we're using the WinPmem which is a command line open-source tool used to get memory acquisition.

4. **SCP(Secure Copy Protocol):** Transferred the captured memory dumps from the Windows VM to the REMnux VM using SCP. Ensured that the transfer process was secure and the integrity of the data was maintained during transit.

5. **Volatility Framework:** The Volatility Framework is an effective open-source tool for memory forensics and analysis. Memory dumps can be analyzed to detect significant indicators of compromise (IoCs) within a system. In this case, Volatility was used to analyze memory dumps and extract essential information such as ongoing processes, network connections, and other in-memory activity.

After extracting the relevant data with Volatility, the material was arranged into CSV files for further processing and analysis. The CSV format is especially beneficial for the Large Language Models to detect patterns and anomalies. This systematic method not only facilitates complete investigations but also promotes ongoing monitoring and preemptive defensive tactics.

6. **SecureBERT:** SecureBERT is a machine learning model designed for cybersecurity applications, specifically to identify complex threats such as fileless malware. The model can identify patterns and behaviors that indicate harmful activity by using CSV data obtained by Volatility analysis. Volatility, a well-known memory forensics program, pulls critical information from memory dumps, resulting in a full snapshot of the system's state at any given time. This data includes a variety of artifacts, such as running processes, network connections, and loaded modules, all of which are critical for understanding the context and behavior of possible threats.

By training SecureBERT on this rich dataset, the model learns to distinguish between normal and malicious actions, improving its detection skills.

SecureBERT is trained using vast data from Volatility's memory dump analysis, which allows it to understand the various intricacies of fileless malware behavior. Unlike typical malware, fileless malware does not rely on files on disk, making it more difficult to detect with regular antivirus technologies. Instead, malware works directly in the system's memory, exploiting weaknesses and using legitimate system tools. By focusing on memory-resident indications of infiltration, SecureBERT may detect abnormalities and suspicious patterns common to fileless malware. This proactive strategy allows cybersecurity experts to better detect and mitigate risks, offering a strong defense against advanced persistent threats and other complex cyber assaults.

7. Python Scripting: Python scripting has been used throughout our project to automate various tasks and processes to save on time. We've used Python scripting to automate several key areas of the process which include handling virtual machine operations, the detonation of malware samples, the creation of raw memory dumps and the transfer of those raw files to REMnux. Another big function is in training and testing the LLM. We've used Python scripting which gives us an easier way to have our data preprocessed, ingested and tested. Python scripting gives us a path to choose the columns, train the data and make any changes effectively and efficiently.

Results

For the results, we ran ten tests and generated results as seen below. Each iteration was trained separately so that the results could be as accurate as possible. Our dataset would be split into two distinct sets one for training and one for testing. In our tests we made changes to which labels and categories would be available to the model. The training set had the fileless/benign tag for guided training and the test set and it was removed so that predictions would be made on the data rather than specific columns. Data leakage can happen when too much information is presented to the large language model during training.

	Correct	Incorrect	# of sucessfull fileless guessed	# of successful benign guessed	Test Data (Samples in Dataset)
Trial #1	11	1	5	6	phymem21, phymem42, phymem35, phymem46, phymem5, phymem55, phymem56, phymem30, phymem43, phymem40, phymem49, phymem57
Trial #2	10	2	4	6	physmem38, physmem16, physmem5, physmem23, physmem24, physmem27, physmem31, physmem9, physmem52, physmem14, physmem8, physmem28
Trial #3	11	1	5	6	physmem13, physmem52, physmem7, physmem43, physmem29, physmem14, physmem36, physmem21, physmem2, physmem9, physmem45, physmem42
Trial #4	12	0	6	6	physmem32, physmem43, physmem34, physmem22, physmem27, physmem31, physmem16, physmem52, physmem59, physmem7, physmem4, physmem26
Trial #5	11	1	5	6	physmem4, physmem44, physmem14, physmem11, physmem37, physmem15, physmem29, physmem22, physmem10, physmem30, physmem, 45, physmem28
Trial #6	7	5	1	6	physmem15, physmem46, physmem6, physmem23, physmem38, physmem16, physmem17, physmem8, physmem29, physmem33, physmem36, physmem20
Trial #7	11	1	5	6	physmem3, physmem47, physmem10, physmem8, physmem,38, physmem59, physmem16, physmem51, physmem20, physmem13, physmem9, physmem46
Trial #8	9	3	3	6	physmem44, physmem19, physmem2, physmem8, physmem17, physmem25, physmem42, physmem30, physmem54, physmem45, physmem16, physmem57
Trial #9	12	0	6	6	physmem58, physmem7, physmem53, physmem12, physmem13, physmem39, physmem11, physmem4, physmem47, physmem48, physmem43, physmem26
Trial #10	11	1	5	6	physmem14, physmem35, physmem10, physmem46, physmem50, physmem56, physmem16, physmem8, physmem22, physmem26, physmem23, physmem54
Total Tests	Total Correct	Total Incorrect	Ratio		
120	105	15	87.50%		

Ten trials to gauge the accurate classification by SecureBERT.

Our tests showed that on average only a few test samples would be incorrectly labelled entirely stemming from fileless malware being labelled as benign. The model was excellent at spotting benign samples largely attributable to the consistency in processes and DLLs. Each trial has a test dataset containing twelve unique samples and when splitting from the main dataset it is stratified to ensure that the both sets share the same ratio of fileless malware and benign samples. There was a disturbance in the sixth trial conducted where the model labelled most of the samples as benign which dropped the average results. Despite further testing and modifications, the cause of the reduced accuracy remains unidentified. However there were trials where all samples were classified accurately.

The average was calculated from the successful and failed prediction. The total correct count was 105 and the total incorrect count was 15 resulting in an average accuracy of 87.50%. While the results demonstrate a positive outcome, a more comprehensive assessment of the LLMs

capabilities would require a larger dataset and additional testing. However, these results demonstrate that LLMs have potential in detecting fileless malware and can set a new precedent in this domain.

Conclusions and Future Work

In conclusion our findings determined that detecting fileless malware using memory dumps is possible using large language model securebert(RobertaSequenceClassification). The task we used was text-classification to determine whether a memory dump was fileless or benign. The volatility plugins we used were pslist, pstree, psxview, dlllist, handles, ldrmodules, svcsan, and cmdline. The output of the plugins were compared to anyrun to see if we were able to detect the fileless tactics and techniques that were occurring using living off the land binaries. The data was copied to excel document and exported to csv file where the large language model was fine tuned on our data. Large language model made predictions on the test set and was 87.50% accurate on 10 tries.

What we learned was how to fine tune a hugging face large language model, how to analyze memory dumps using volatility, how to create a python script to automatically revert to a predetonation snapshot, how to create a python script to unzip a malware with password of infected and transfer to remnux using scp.

The future work we would like to implement is adding additional volatility plugins hivelist, netscan, mutantscan, modules, etc. We would like to add volatility plugins that hopefully will be created in the future that focuses on powershell, and WMI. We would like to create a sandbox to detonate malware that isn't virtualized. Also, add mitre att&ck tactics and techniques to data to be trained.

Bibliography and Acknowledgements

- [1] Aghaei, E., Niu, X., Shadid, W., & Al-Shaer, E. (2022). SecureBERT: A DomainSpecific Language Model for Cybersecurity. arXiv.Org.
<https://doi.org/10.48550/arxiv.2204.02685>

- [2] Aghaei, E. (2024). Ehsanaghaei/SecureBERT [Computer software].
<https://github.com/ehsanaghaei/SecureBERT> (Original work published 2022)

- [3] Code In a Jiffy (Director). (2023, January 14). *Build a Text Classifier with Transformers in 5 minutes* [Video recording].
<https://www.youtube.com/watch?v=8yrD0hR8OY8>
- [4] HuskyHacks (Director). (2022, August 22). *Malware Analysis In 5+ Hours—Full Course—Learn Practical Malware Analysis!* [Video recording].
<https://www.youtube.com/watch?v=qA0YcYMRWyl>
- [5] hzqst. (2024). *Hqzst/VmwareHardenedLoader* [C].
<https://github.com/hzqst/VmwareHardenedLoader> (Original work published 2018)
- [6] Karantzas, G., & Patsakis, C. (2021). An Empirical Assessment of Endpoint Detection and Response Systems against Advanced Persistent Threats Attack Vectors. *Journal of Cybersecurity and Privacy*, 1(3), 387. <https://doi.org/10.3390/jcp1030021>
- [7] Khalid, O., Ullah, S., Ahmad, T., Saeed, S., Alabbad, D. A., Aslam, M., Buriro, A., & Ahmad, R. (2023). An Insight into the Machine-Learning-Based Fileless Malware Detection. *Sensors*, 23(2), Article 2. <https://doi.org/10.3390/s23020612>
- [8] Labs, Vmr. (2022, March 31). *Pafish: How to Test your Sandbox Against Virtualization Detection*. VMRay. <https://www.vmray.com/a-pafish-primer/>
- [9] Liu, S., Peng, G., Zeng, H., & Fu, J. (2024b). A survey on the evolution of fileless attacks and detection techniques. *Computers & Security*, 137, 103653.
<https://doi.org/10.1016/j.cose.2023.103653>
- [10] Ninite, N. (n.d.). *Ninite—Install or Update Multiple Apps at Once*. Retrieved July 31, 2024, from <https://ninite.com/>
- [11] Polsen, H. (n.d.). *Accessing trials and kits for Windows*.

TECHCOMMUNITY.MICROSOFT.COM. Retrieved July 31, 2024, from <https://techcommunity.microsoft.com/t5/windows-11/accessing-trials-and-kits-for-windows/m-p/3361125>

[12] Public Affairs, O. of. (2021, January 28). Office of Public Affairs | Emotet Botnet Disrupted in International Cyber Operation | United States Department of Justice. <https://www.justice.gov/opa/pr/emotet-botnet-disrupted-international-cyberoperation>

[13] Remnux, R. (n.d.). *REMnux: A Linux Toolkit for Malware Analysts*. Retrieved July 31, 2024, from <https://remnux.org/>

[14] Room, N. (2024, January 25). LODEINFO Fileless Malware Evolves with AntiAnalysis and Remote Code Tricks. The Hacker News. <https://thehackernews.com/2024/01/lodeinfo-fileless-malware-evolves-with.html> 18

[15] SANS, S. (2024, June 10). *Hunt Evil* | SANS Poster. <https://www.sans.org/posters/hunt-evil/>

[16] Transformers, H. F. (n.d.). *ImportError: Using the Trainer with PyTorch requires accelerate>=0.20.1 · Issue #28191 · huggingface/transformers*. GitHub. Retrieved August 9, 2024, from <https://github.com/huggingface/transformers/issues/28191>

[17] Törnberg, P. (2023). How to use LLMs for Text Analysis. arXiv.Org. <https://doi.org/10.48550/arxiv.2307.1310>

[18] Volatility, V. (n.d.). *The Volatility Foundation—Promoting Accessible Memory Analysis Tools Within the Memory Forensics Community*. The Volatility Foundation Promoting Accessible Memory Analysis Tools Within the Memory Forensics Community.

Retrieved July 31, 2024, from <https://volatilityfoundation.org/>

[19] WinPmem, V. (2024). *Velocidex/WinPmem* [C]. Velocidex.

<https://github.com/Velocidex/WinPmem> (Original work published 2019)

[20] Wisabi Analytics (Director). (2023, March 14). *How to Install and Activate Microsoft Office 2021 for Free—Step by Step Guide* [Video recording].

<https://www.youtube.com/watch?v=c9EVh3QtrG4>