International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

Unveiling the Veiled: Unmasking Fileless Malware through Memory Forensics and Machine Learning

Jyoshna Bejjam[1*]
Associate Professor, CSE dept. Keshav Memorial Institute Of Technology
Hyderabad, India
e-mail: drjyoshnabejjam@gmail.com

Bhuvanagiri[2,] **Sai Devansh,**
Student**,** CSE dept. Keshav Memorial Institute Of Technology
Hyderabad, India
e-mail: saidevansh023@gmail.com

Recharla Divya Reddy[3,]
Student, CSE dept. Keshav Memorial Institute Of Technology
Hyderabad, India
e-mail: recharladivyareddy@gmail.com

Mandadi Vaishnavi[4]**,**
Student, CSE dept. Keshav Memorial Institute Of Technology
Hyderabad, India
e-mail: vaishnavi272012@gmail.com

Sravya Ravulakolla[5]**,**
Student, CSE dept. Keshav Memorial Institute Of Technology
Hyderabad, India
e-mail: ravulakollasravya@gmail.com

Abstract-In recent times, significant advancements within the realm of malware development have dramatically reshaped the entire landscape. The reasons for targeting a system have undergone a complete transformation, shifting from file-based to fileless malware.Fileless malware poses a significant cybersecurity threat, challenging traditional detection methods. This research introduces an innovative approach that combines memory forensics and machine learning to effectively detect and mitigate fileless malware. By analyzing volatile memory and leveraging machine learning algorithms, our system automates detection.We employ virtual machines to capture memory snapshots and conduct thorough analysis using the Volatility framework. Among various algorithms, we have determined that the Random Forest algorithm is the most effective, achieving an impressive overall accuracy rate of 93.33%. Specifically, it demonstrates a True Positive Rate (TPR) of 87.5% while maintaining a zero False Positive Rate (FPR) when applied to fileless malware obtained from HatchingTriage, AnyRun, VirusShare, PolySwarm, and JoESandbox datasets. To enhance user interaction, a user-friendly graphical interface is provided, and scalability and processing capabilities are optimized through Amazon Web Services.Experimental evaluations demonstrate high accuracy and efficiency in detecting fileless malware. This framework contributes to the advancement of cybersecurity, providing practical tools for detecting against evolving fileless malware threats.

***Keywords**-Fileless malware, Cyber Security, machine learning, volatility.*

I. INTRODUCTION*(HEADING 1)*

The concept of detection raises when there is something inappropriate happens with the existing resources, Likewise we have come up with a detection tool to find the inappropriate behavior happening in a system. The reason behind this unusual behavior could be "MALWARE"

Malware: It refers to deliberately crafted computer programs designed to disrupt pre-existing computer applications. Malicious software may be employed for purposes such as pilfering sensitive information, disrupting normal operations, and inflicting harm on computer systems.

In the swiftly changing cybersecurity environment of today, the increasing sophistication of cyberattacks is surpassing the effectiveness of traditional malware detection methods. Fileless malware, in particular, poses a significant challenge due to its stealthy nature and ability to evade detection by residing solely in memory. Detecting and mitigating fileless malware requires innovative approaches that go beyond signature-based detection methods and delve into the realm of memory forensics and machine learning.

In recent years, the prevalence of fileless malware attacks has escalated, leading to substantial financial losses,

reputational damage, and compromised data security for organizations across various sectors. Traditional malware detection mechanisms, such as signature-based antivirus software, struggle to identify fileless malware due to its minimal footprint on disk and its reliance on exploiting existing legitimate processes. As a result, fileless malware has emerged as a potent weapon in the arsenal of cybercriminals, enabling them to infiltrate systems, extract sensitive information, and maintain persistence.

To address this critical issue, our research focuses on leveraging memory forensics and machine learning techniques to detect and mitigate fileless malware effectively. By examining the volatile memory of a compromised system, we can identify indicators of fileless malware attacks that may not be visible through traditional disk-based analysis. Furthermore, by harnessing the power of machine learning algorithms, we aim to automate the detection process, enabling real-time identification and response to fileless malware incidents.

Fileless malware is specifically crafted to function without leaving any discernible traces on the infected system's hard drive. Instead, it resides solely within volatile memory (RAM) and leverages the functionality of legitimate tools and features integrated into the operating system. This makes it exceptionally difficult to detect using conventional methods. Indications of fileless malware include slow performance, pop-up ads, unexpected browser behavior, high network traffic, unwanted toolbars, and unexpected command prompt running.

To understand the unique challenges posed by fileless malware, it is essential to differentiate it from file-based malware, the more familiar form of malicious software.

File-Based Malware: File-based malware constitutes a category of malicious software distributed through various file formats, including executables, documents, or scripts. These malware types typically rely on signature-based detection methods and are typically stored on a hard disk, making them susceptible to detection by antivirus (AV) and firewall systems, file sandboxes, host-based security systems, as well as anomaly detection mechanisms. These solutions generally use a combination of signatures for files, called hashes, and URLs or IPs of known malicious or compromised systems.

FILELESS Malware: Malicious software that operates without leaving traces of files on the infected system’s hard drive. Instead of relying on traditional file-based methods, fileless malware resides in the system’s volatile memory (RAM). It is designed to bypass the conventional defenses mentioned above.

These fundamental differences between file-based malware and fileless malware necessitate innovative detection methods. Traditional approaches that rely on scanning files and comparing them against known signatures are ineffective against fileless malware, which resides solely in memory. Therefore, a paradigm shift is required to address this challenge, shifting the focus toward memory forensics and machine learning techniques. By examining the volatile memory of a compromised system, we can identify indicators of fileless malware attacks that may not be visible through traditional disk-based analysis. Furthermore, by harnessing the power of machine learning algorithms, we aim to automate the detection process, enabling real-time identification and response to fileless malware incidents.

To overcome the limitations of traditional detection mechanisms, our research proposes a comprehensive framework that combines memory forensics and machine learning. By analyzing the volatile memory of a compromised system using tools like Volatility, we can extract essential features and patterns indicative of fileless malware. These features serve as input data for machine learning algorithms, which have proven successful in various domains, including fileless malware detection. The Random Forest classifier algorithm is employed to classify the obtained test data, enabling the prediction of malware or benign behavior.

To enhance usability and facilitate seamless interaction with our memory forensics system, we develop an intuitive graphical user interface (GUI) using PyQt. This GUI empowers security analysts and researchers to navigate the complex landscape of memory forensics, visualize the results of the analysis, and make informed decisions in a user-friendly manner.

Moreover, our research leverages the scalability and processing capabilities of cloud computing through Amazon Web Services (AWS). By utilizing services such as Flask, Amazon S3, and Amazon EC2, we optimize file transfer, model deployment, and memory analysis tasks. This cloud-based approach ensures efficient resource utilization and enhances the scalability of our system to handle large-scale memory dumps. Through rigorous experimentation and evaluation, we demonstrate the efficacy of our approach in detecting fileless malware. Our results showcase improved detection rates, reduced false positives, and enhanced response times, thereby empowering organizations to protect their digital assets against fileless malware attacks.

By utilizing Virtual machines, we establish a secure testing environment that enables the capture and analysis of memory snapshots. Through the DumpIt tool, we obtain memory dumps, which serve as the basis for our subsequent memory forensic analysis using the Volatility framework. This allows us to extract essential features and patterns indicative of fileless malware, enhancing our detection capabilities.

In summary, this research proposed work aims to contribute to the field of cybersecurity by presenting a comprehensive framework for detecting fileless malware using memory forensics and machine learning. By combining theoretical knowledge, practical experimentation, and cutting-edge technologies, we provide a unique approach to address the challenges posed by fileless malware and offer practical tools for enhancing the security posture of organizations in an increasingly sophisticated threat landscape.

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

2. Literature Survey

2.1 Fileless Malware

Fileless malware represents a novel category within the realm of memory-resident malware. It distinguishes itself by its ability to infiltrate and compromise a targeted system without leaving any discernible traces on the target's file system or secondary memory storage [1].

Fileless malware uses the tools and libraries built into the infected device's platform to carry out its harmful objectives. In other words, fileless malware manipulates benign applications and software libraries to achieve the attack's goals. Major corporations have already fallen victim to fileless malware attacks, with 140 enterprises experiencing such attacks in 2017, according to Kaspersky Lab's Global Research and Analysis Team. The Ponemon Institute reported that fileless tactics were used in 77% of attacks against businesses. Additionally, there are indications that ransomware attacks are adopting fileless techniques, as described in Magnusardottir's research in 2018 [2].

By altering the Windows registry or establishing particular services that download or generate the malware as soon as the operating system starts, it can carry out persistence-related actions. Concealment techniques have evolved significantly in recent years, including the use of polymorphic and metamorphic methodologies to evade antivirus signature-based detection. Fileless malware has further complicated analysis since it resides solely in volatile memory, rendering traditional disk analysis ineffective [3].

Malware creators employ various methods to achieve persistence on a compromised system. This includes manipulating the Windows registry or establishing specific services that download or generate malware during the operating system's startup. The malware's code is often encrypted within the registry to avoid detection, and it may access the operating system's thumbnail cache to ensure persistence. Once its harmful mission is complete, the file typically deletes itself [4].

One common technique used by malware involves injecting DLLs into victims' systems. Initially, the malware employs the "VirtualAllocEx" API to allocate memory in the victim's address space. It then uses the "WriteProcessMemory" API to write the DLL path into the allocated memory. Subsequently, the malware uses "LoadLibrary" to load the DLL. To create threads within the target process, the malware calls functions like "CreateRemoteThread," "NtCreateThreadEx," or "RtlCreateUserThread." These techniques allow the malware to execute its malicious code while evading detection by antivirus software [5].

2.2 Memory Samples

Memory forensics [6]is a technique utilized to uncover unusual and unauthorized activities on a targeted computer or server. This method often involves employing specialized software designed to capture the current state of the system's memory, generating a snapshot file commonly known as a memory dump. This captured file can then be transferred to an external location for examination by experts in digital forensics. Among the leading memory forensic tools are Belkasoft RAM Capture, Process Hacker, Volatility Suite, Rekall, and Helix ISO.Several approaches to memory acquisition encompass:

1. Pagefile: A file that mirrors data present in the system's RAM.
2. VMware snapshot: A frozen instance of a virtual machine's precise state at the time of snapshot creation.
3. RAW format: Extracted directly from an active cyber environment.
4. Crash dump: Information gathered by the operating system when system crashes occur.
5. Hibernation file: A retained snapshot intended for the operating system's hibernation recovery.

Memory analysis [7] has gained significant traction in recent years as an effective and accurate method within the realm of malware analysis. This technique is appealing to malware analysts due to its ability to comprehensively dissect malware, including its actions beyond the usual functional boundaries.

The process of memory forensics involves two pivotal stages:

- Memory Acquisition: This phase entails the extraction of the target machine's memory to create a memory image. Specialized tools like Memoryze, FastDump, and DumpIt are commonly employed for this purpose.

- Memory Analysis: Subsequently, the obtained memory image undergoes analysis to identify potentially malicious activities. Experts typically rely on tools like Volatility and Rekall for this memory analysis.

Forensic Toolkit Imager (FTK Imager) [10] is an invaluable freeware forensics solution created by AccessData, specifically designed to aid researchers in conducting computer forensic examinations. Its primary function is to acquire precise forensic images, encompassing both physical memory and logical data, providing a comprehensive view of the digital evidence. This tool's versatility extends to reading these acquired forensic images, decrypting encrypted data, and generating detailed reports, making it an indispensable asset in the realm of digital forensics.

2.3 Volatility tool

In[11] order to retrieve distinctive attributes from a memory dump, whether it's from a malicious or non-malicious source, and then store these attributes in a CSV file for future use in training and testing machine learning models, we employ the Volatility Framework tool. The Volatility tool boasts an extensive array of over 70 plugins, each designed to scrutinize diverse aspects of system memory. This tool is compatible with both 32-bit and 64-bit operating systems, encompassing a wide range of platforms, including Windows, Linux, and macOS variations.

To dissect the memory dump effectively, the initial step involves configuring the correct profile. This profile is essential as it helps Volatility identify the underlying operating system of the memory dump. Once the profile is properly established, we can employ various Volatility plugins to extract critical information. This includes details like active processes, loaded

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

DLLs within those processes, running services, network connections, registry hive listings, and more. [11]

Numerous[12] tools have been developed to facilitate memory analysis, enabling users to sift through memory dumps for valuable data. Within the realm of open-source solutions, Volatility and Rekall stand out, while commercial offerings include Cellebrite Inspector, FireEye Redline, Magnet AXIOM, and WindowsSCOPE. Notably, the majority of research approaches incorporate Volatility software, and many commercial products either expose or leverage Volatility's capabilities.

When delving into the field of volatile memory forensic tools, it is essential to highlight Volatility—a Python-based, open-source framework tailored for the examination of memory dumps. [12].

Volatility[13] possesses the capability to examine memory dumps originating from a variety of platforms, including Windows, Linux, and Macintosh systems. It's versatile in handling various file dump formats and boasts an extensible Application Programming Interface (API). Additionally, Volatility excels in creating features and is known for its efficient implementation.

When working with Volatility, it offers a range of modes and commands, but a standard usage scenario typically follows this pattern:

python3 vol.py -f windows.pslist

In this illustration, Volatility is employed to extract the list of processes active in a Windows system at the moment when the memory image was captured.[13]

2.4 Dataset

In 2022, the Canadian Institute for Cybersecurity made available the dataset known as CIC-MalMem-2022, which was used in this study. It is a balanced dataset with 58,596 records, equally divided between benign and malicious samples. Within the dataset, you can find three distinct categories of malware: Ransomware, Trojan and Spyware. Malicious memory dumps were created by executing VirusTotal samples on a 2GB memory VM, while normal behavior memory dumps were generated by running applications on the machine. The study involved classifying hidden malware families through memory analysis[14]

The obfuscated malware dataset aims to evaluate methods for detecting hidden malware that evades detection. It closely resembles real-world scenarios and includes prevalent malware samples. The dataset is balanced, consisting of an equal split of 50% malicious memory dumps and 50% benign memory dumps. It comprises a total of 58,596 records, with 29,298 benign and 29,298 malicious samples.[15]

The process of creating the dataset encompassed four primary phases: research, extraction of memory dumps, transfer of memory dumps, and extraction of features.

In research phase, we gathered a range of 100 to 200 malware samples from five distinct families spanning three malware categories: Trojan Horse, Ransomware, and Spyware. Our objective was to replicate real-world scenarios while excluding malware designed for obsolete systems.

In the memory dumping step, memory snapshots were captured using the VirtualBox virtual machine management system, ensuring a clean environment without additional processes. The dumps were taken from a Windows 10 system to closely resemble real-world scenarios.

To incorporate benign processes in the creation of malicious memory dumps, we simultaneously ran various applications in the Windows virtual machine alongside the execution of malware samples. This deliberate approach aimed to prevent the classifier from solely depending on benign processes for distinguishing between malicious and non-malicious data. Each execution of a malware sample generated 10 memory dumps, spaced 15 seconds apart, resulting in a total of 29,298 malicious memory dumps. For benign dumps, we captured normal user behavior by running different applications, and we balanced the dataset through oversampling using the SMOTE algorithm.

Subsequently, the process entailed moving the memory dump files to a Kali Linux machine, where we performed feature extraction utilizing the VolMemLyzer tool. This process included 26 new features targeting malware obfuscation. Finally, the memory dump files underwent feature extraction using VolMemLyzer, and a combined CSV file was created for all the tested memory dumps. This file was intended for use in the ensemble learning system.[16]

CIC-MalMem-2022 serves as an obfuscated malware dataset designed to assess the effectiveness of memory-based obfuscated malware detection algorithms. Its primary objective is to replicate authentic situations by incorporating widely recognized malware. The dataset consists of an equal quantity of malicious and benign memory dumps, totaling 58,596 samples. It offers 56 distinct features for the application of machine learning algorithms. To mitigate the challenges posed by high dimensionality, it is strongly recommended to employ feature selection methods before implementing machine learning techniques on the CIC-MalMem-2022 dataset.[17]

CIC-Malmem-2022 dataset has received significant attention in recent literature. In a specific research investigation [18], the authors tackled the issue of class imbalance within their dataset through the application of oversampling and the utilization of XGBoost as preprocessing methodologies. Following a comprehensive assessment of multiple algorithms, they observed that Random Forest and Multilayer Perceptron (MLP) surpassed other models, achieving an accuracy rate of nearly 100%, particularly in the context of detection.[19]

2.6 Machine learning

Binary class classification, involves using machine learning techniques to categorize data into two distinct groups: malware and non-malware (benign). To effectively build and evaluate a classifier, it is essential to divide the available data into two parts: a training set and a testing set. The training set is employed to "teach" the classifier by adjusting its model parameters to fit the data, while the testing set is employed to assess how well the classifier performs on new, unseen data.

For the task of fileless malware detection, several state-of-the-art machine learning algorithms, including Random Forest (RF), Decision Tree (DT), Support Vector Machine (SVM), Logistic Regression (LR), K-Nearest Neighbor (KNN), XGBoost (XGB), and Gradient Boosting (GB) are used. These

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

algorithms are widely recognized for their ability to effectively categorize data into the desired binary classes. Each algorithm has its unique strengths and weaknesses, and the choice of which one to use for a specific task depends on the characteristics of the data and the specific requirements of the problem at hand.[20]

In all experimental configurations, the Random Forest (RF) algorithm consistently delivered an accuracy rate of 93.33%, achieving a True Positive Rate (TPR) of 87.5% while maintaining a False Positive Rate (FPR) of 0%.[20] Notably, it is important to highlight that the Support Vector Machine (SVM) approach, even with optimized parameters, achieved a similar TPR of 87.5% as RF; however, it exhibited significantly higher FPR of 14.28% compared to RF's 0%.[20]

They conducted feature extraction[21]from decompiled malicious binaries and employed the Random Forest algorithm for malware classification based on these derived features. Utilizing a dataset encompassing 10,260 instances of malware, their findings revealed an accuracy rate of up to 99.21%.

2.5 Automation

AWS delivers computing resources that are readily available when needed, following a billing model where users pay based on their actual usage. This empowers both individuals and organizations to access virtual clusters of computers via the Internet. Amazon Elastic Compute Cloud (EC2) is a popular service that provides continuous access to virtual PCs, complete with CPU, GPU, RAM, storage, and pre-loaded software. AWS excels in Infrastructure as a Service (IaaS) compared to Azure and Google Cloud. However, it may be relatively more expensive, and its security may not be as strong as that of Azure and Google Cloud.[22]

As reported in a recent ZDNet article, in January 2012, EC2 comprised a substantial network of 454,600 servers. When factoring in other AWS services, the aggregate count of Amazon's systems devoted to cloud computing is even more extensive.EC2 is a web service that allows easy launching of application instances across various operating systems. Simple Storage System (S3) is a storage service tailor-made for the purpose of accommodating large objects. AWS services can be accessed through the AWS Management Console, command-line tools, SDK libraries, or raw REST requests. S3 stores data in buckets, and CloudWatch supports application monitoring and automatic migrations. To access AWS, users need to create an account and use the provided tools and services, such as EC2 instances. DNS is used to map human-friendly names to IP addresses, and the Java API is available through the AWS SDK.[23]

AWS provides readily available computing resources, follows a flexible pay-as-you-use billing approach, caters to a diverse clientele including individuals and various types of organizations such as public, private, and government entities. Some of its notable clients include the US Navy, Unilever, Kellogg's, and Siemens. AWS offers a comprehensive suite of cloud services, including Software as a Service (SaaS), Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), with its Infrastructure as a Service (IaaS) solution being widely acclaimed on a global scale.[24]

AWS Lambda is a platform suitable for executing High Throughput Computing jobs. Although it provides a user-friendly computing environment for general-purpose applications that adhere to its limitations, such as maximum execution time, RAM, and disk space, it demonstrates diverse performance characteristics that could potentially restrict its suitability for closely interconnected computing tasks. AWS Lambda is recognized as a pioneer in serverless computing, supporting stateless functions written in languages like Node.js, Java, C#, Python, and Go. It allows for massive-scale execution, handling up to 3000 parallel invocations in response to events.[25]

3. PROPOSED WORK

This work proposes a five-phase approach for automating memory dump analysis using machine learning techniques. It aims to streamline the process of analyzing memory dumps, extract valuable insights, and build predictive models for system behavior.

The five phases include collecting memory dumps, Dataset selection, building a machine-learning model, extracting features, and automating the entire workflow. Each stage serves a pivotal function in attaining the project's objectives and adds to the overall efficiency and effectiveness of the analysis procedure.

3.1 Memory Acquisition

We focused on the collection and acquisition of memory dumps as a vital step in the analysis of fileless malware. Our objective was to gather a diverse collection of memory dumps with extensions such as .vmem, .mem, .raw, and .dmp. These memory dumps were obtained using various tools capable of capturing different memory dump formats.

The data for our research consisted of memory dumps obtained from different sources. These sources included controlled experiments conducted within our virtual machine environment, where we captured memory snapshots using tools capable of extracting memory dumps. We also accessed publicly available memory dump repositories, which provided additional real-world memory dumps for analysis.

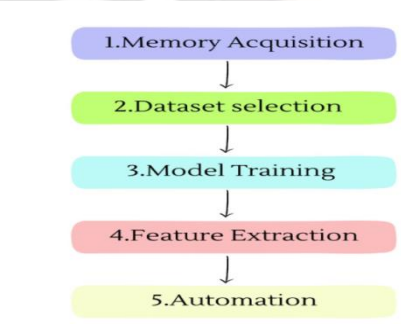

Fig 1: Five-phase approach

To collect the memory dumps, we employed various tools specific to each file extension. We obtained .vmem files using VMware Workstation Player 17 Pro, capturing the memory

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

snapshots from virtual machines running within the virtualization platform. In the case of .raw files, we employed DumpIt, a well-known tool recognized for creating raw memory dump files from physical systems. The .mem files were acquired through FTK Imager, a widely used tool for capturing memory dumps from non-virtualized environments. Lastly, .dmp files were obtained using WinDbg, a Windows debugging tool capable of generating memory dump files during system crashes or blue screen errors.

These memory dumps served as the foundation for our subsequent analysis and evaluation of fileless malware detection techniques.

3.2 Dataset selection and preparation

The dataset selected for our work on fileless malware detection is the Obfuscated MalMem dataset, obtained from the Cybersecurity and Infrastructure Security Agency (CIC). This dataset provides valuable insights into the challenges faced in detecting obfuscated malware instances in real-world scenarios.

The dataset Obfuscated MalMem consists of a wide-ranging assortment of memory samples obtained from real instances of malware. It contains a total of 58596 samples, consisting of both malware and benign instances, making it suitable for training and evaluating machine learning models.

To prepare the dataset for analysis, we performed necessary preprocessing steps. This involved removing irrelevant columns, handling missing values, and addressing any inconsistencies present in the data. Additionally, data normalization techniques were applied to standardize the features and ensure effective model training and evaluation.

In summary, the Obfuscated MalMem dataset offers an exclusive and representative assortment of memory samples to train and assess machine learning models for detecting malware. Below is a glimpse of all the attributes of the dataset.

Table 1: Dataset attributes [14]

S.No	Name of the Feature	Description
1	Category	Type of Malware
2	pslist.nproc	The complete count of processes.
3	pslist.nppid	The complete count of parent processes.
4	pslist.avg_threads	The mean count of threads for the processes.
5	pslist.nprocs64bit	The overall count of processes operating in a 64-bit environment.
6	pslist.avg_handlers	The mean count of handlers.
7	dllist.ndlls	The combined number of loaded libraries for each individual process.
8	dllist.avg_dlls_per_proc	Mean number of libraries loaded per process on average.
9	handles.nhandles	The complete count of handles that have been opened.
10	handles.avg_handles_per_proc	The mean count of handles per process.
11	handles.nport	The overall count of port handles.
12	handles.nfile	The overall count of file handles.
13	handles.nevent	The overall count of event handles.
14	handles.ndesktop	The overall count of desktop handles.
15	handles.nkey	The overall count of key handles.
16	handles.nthread	The overall count of thread handles.
17	handles.ndirectory	The overall count of directory handles
18	handles.nsemaphore	The overall count of semaphore handles
19	handles.ntimer	The overall count of timer handles
20	handles.nsection	The overall count of section handles
21	handles. Nmutant	The overall count of mutant handles
22	ldrmodules.not_in_load	The overall count of modules missing from the load list
23	ldrmodules.not_in_init	The overall count of modules missing from the init list
24	ldrmodules.not_in_mem	The overall count of modules missing from the memory list
25	ldrmodules.not_in_load_avg	The mean quantity of absent modules from the loading list.
26	ldrmodules.not_in_init_avg	The mean quantity of absent modules from the init list
27	ldrmodules.not_in_mem_avg	The The mean quantity of absent modules from the memory
28	malfind.ninjections	The cumulative count of hidden code injections.
29	malfind.commitCharge	The cumulative count of Commit Charges
30	malfind.protection	The cumulative count of protection
31	malfind.uniqueInjections	The cumulative count of unique injections
32	psxview.not_in_pslist	The overall count of processes that are absent in the pslist.
33	psxview.not_in_eprocess_pool	The overall count of processes that are absent in the psscan
34	psxview.not_in_ethread_pool	The overall count of processes that are absent in the thrdproc
35	psxview.not_in_pspcid_list	The overall count of processes that are absent in the pspcid
36	psxview.not_in_csrss_handles	The overall count of processes that are absent in the csrss
37	psxview. not_in_session	The overall count of processes that are absent in the session
38	psxview. not_in_deskthrd	The overall count of processes that are absent in thedesktrd
39	psxview.not_in_pslist_false_avg	The mean false ratio of the process list.
40	psxview.not_in_eprocess_pool_false_avg	The mean false ratio of the process scan
41	psxview.not_in_ethread_pool_false_avg	Average rate of processes not found in the ethread pool in psxview.
42	psxview.not_in_pspcid_list_false_avg	The average rate of processes not present in the pspcid list in psxview.

43	psxview.not_in_csrss_handles_false_avg	The average rate of processes not included in the csrss handles list in psxview.
44	psxview.not_in_session_false_avg	The average rate of processes not found in the session list in psxview.
45	psxview.not_in_deskthrd_false_avg	The average rate of processes not found in the deskthrd list in psxview.
46	modules.nmodules	Count of modules
47	svcscan.nservices	Count of services
48	svcscan.kernel_drivers	Count of kernel drivers
49	svcscan.fs_drivers	Count of file system drivers
50	svcscan.process_services	Count of Windows 32 owned processes
51	svcscan.shared_process_services	Count of Windows 32 shared processes
52	svcscan.interactive_process_services	Count of interactive service processes
53	svcscan.nactive	Count of actively running service processes
54	callbacks.ncallbacks	Count of callbacks
55	callbacks.nanonymous	Count of unknown processes
56	callbacks.ngeneric	Count of generic processes
57	Class	Malicious or Benign

These attributes represent various characteristics and properties of the memory samples in the dataset. [14]

3.3 Model Training and Evaluation

3.3.1 Model Training

The Random Forest algorithm was employed to build the machine learning model dedicated to detecting fileless malware. This model underwent training using the gathered dataset, which included memory dumps from diverse sources.

The dataset was divided into two components: the features (X) and the corresponding labels (Y). The features represented extracted characteristics from the memory dumps, while the labels indicated whether each sample was classified as malware or benign. Subsequently, the Random Forest algorithm was employed to adapt the model to the provided training dataset.

Throughout the training procedure, the Random Forest algorithm constructed a collection of decision trees, with each tree being trained on distinct portions of the training dataset.This ensemble approach allowed the model to capture complex relationships between the features and the corresponding malware or benign labels.

3.3.2 Performance Metrics

Precision, recall, accuracy, F1-score, and the area under the ROC curve (AUC) were calculated as part of evaluating the performance of the trained model.Each metric provided insights into different aspects of the model's performance in detecting fileless malware instances.

Table 2: The Performance metrics for various algorithms

Algorithm	Accuracy	Precision	Recall	F1_score
Logistic Regression	1.0	1.0	1.0	1.0
SVM	0.99	0.98	0.99	0.99
KNN	1.0	1.0	1.0	1.0
Random forest	1.0	1.0	1	1.0

Accuracy represented the overall correctness of the model's predictions, while precision measured the proportion of correctly predicted malware instances among all instances predicted as malware. Recall, which is also referred to as sensitivity or the true positive rate, assessed the model's capability to accurately detect instances of malware. The F1-score, on the other hand, integrated both precision and recall to offer a well-rounded evaluation of the model's performance.

3.3.3 Model Evaluation

The model's performance was assessed using an independent test dataset that was not utilized during the training process. This approach enabled us to gauge the model's capacity to generalize and provide accurate predictions for unseen data.

Test dataset, consisting of memory dumps not employed in the training phase, was input into the trained Random Forest model to generate predictions for each sample. These predicted labels were subsequently compared to the actual labels to compute the evaluation metrics.

The evaluation metrics, such as accuracy, recall, precision and F1-score were computed using predicted labels and the true labels in the test dataset. These metrics offered an unbiased evaluation of the model's ability to detect fileless malware.

3.3.4 Results and Discussion

The results obtained from the evaluation of the trained Random Forest model demonstrated its effectiveness in fileless malware detection. The model obtained an accuracy rating of 99.98%, a precision score of 99.96%, a recall rate of 100%, and an F1-score of 99.98%.

These performance metrics confirmed the model's capability to accurately classify memory dumps as either malware or benign. The high accuracy indicated the model's overall ability to make correct predictions, while the precision score reflected its capability to accurately identify malware instances. The recall score demonstrated the model's effectiveness in correctly detecting actual malware instances. The F1-score offered a holistic assessment of the model's performance, as it amalgamated both precision and recall.

The results validate the efficacy of the Random Forest algorithm for fileless malware detection using memory forensics. The model's performance metrics illustrate its capacity to differentiate between malicious and benign instances, thereby enhancing cybersecurity measures against fileless malware attacks.

In summary, the trained Random Forest model exhibited promising performance in fileless malware detection. The assessment criteria, encompassing accuracy, precision, recall, and F1-score, provide evidence of the model's proficiency in

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 9
***Article Received:** 25 July 2023 **Revised:** 12 September 2023 **Accepted:** 30 September 2023*

categorizing memory dumps. These results underscore the capacity of machine learning algorithms to bolster cybersecurity defenses against fileless malware assaults.

3.4 Feature Extraction:

The process of feature extraction is essential for comprehending the attributes and signals associated with suspicious activities in memory dumps. In this phase, specialized tools, such as the Volatility tool are utilized to extract relevant features from the dumps. These features include process information, network connections, file system activity, and registry entries, enabling the model to capture distinct patterns of malware. Careful selection of the appropriate Volatility plugins ensures comprehensive feature extraction, enhancing the model's accuracy and effectiveness.

A brief explanation regarding the plugins has already been discussed in the 2 step(dataset selection) of the proposed work.

Here are a few examples:

1.'pslist' command is used to know the processes running in a memory dump

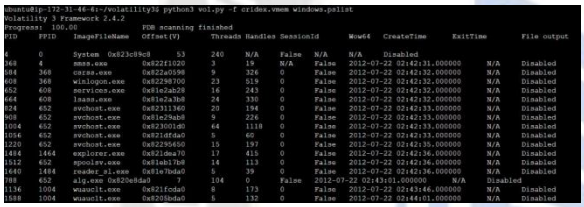

Fig2: 'pslist' command execution

2.'imageinfo' command is used to know the profile of a memory dump

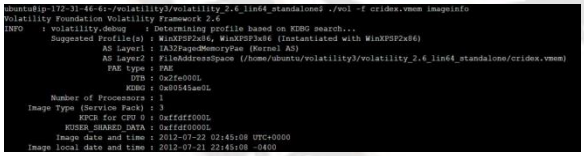

Fig 3:'imageinfo' command execution

3.'dlllist' command is used to retrieve information about loaded Dlls in a memory dump

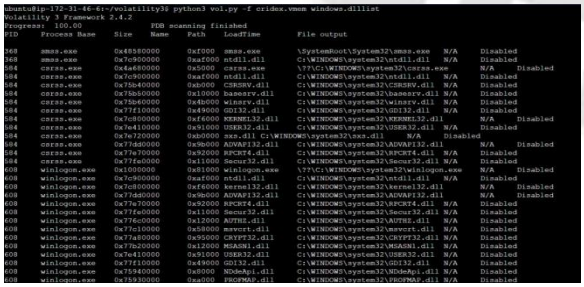

Fig 4: 'dlllist' command execution

3.5 Automation

Our project has utilized various AWS services to streamline processes and achieve efficient outcomes. The core implementation involved leveraging specific AWS services in conjunction with Bash scripting and PythonAnywhere to automate tasks. The key services utilized are as follows:

3.5.1 AWS Lambda Functions

AWS Lambda functions were employed to execute code snippets in a serverless environment. This approach eliminated the need for managing infrastructure and facilitated rapid scalability. By utilizing Lambda functions, we efficiently performed specific tasks, contributing to the automation of our project.

3.5.2 AWS S3 Buckets

To store and manage our project's data and files securely, we utilized AWS S3 buckets. These highly scalable object storage containers provided a reliable solution for our storage needs. With S3 buckets, we were able to ensure the integrity and accessibility of our project's assets.

3.5.3 EC2 Instances

To host and run our project's backend services, we leveraged EC2 instances on AWS. These virtual machines, managed by Amazon's Elastic Compute Cloud service, offered reliable performance and scalability. By utilizing EC2 instances, we met the requirements of our project while ensuring efficient resource allocation.

3.5.4 Bash Scripting

A crucial element in automating tasks and streamlining the deployment process was the implementation of a Bash script. Bash, a command-line shell and scripting language, facilitated seamless interaction between different project components. Our carefully crafted script enabled the effective integration of AWS services and enhanced the overall functionality of our proposed system

3.5.5 PythonAnywhere

For hosting our project's backend services, we utilized PythonAnywhere, a platform that offers Python-based web hosting and server solutions. PythonAnywhere provided a reliable and scalable hosting environment, ensuring the smooth operation of our backend functionality.

After validating the functionality of our project components in the VirtualBox and VMware environment, we proceeded to write the frontend and backend codes. This allowed us to develop the user interface and implement the necessary logic based on the tested and working components.

The integration of AWS services, such as Lambda functions and S3 buckets, with pythonAnywhere and EC2 instance as the hosting platform for our backend services, allowed us to build a robust and flexible infrastructure for our project. With the help of these services and our carefully crafted Bash script, we achieved efficient automation and seamless coordination between various elements of our project.

Table 3 : Comparison Table [26]

Tool Name	First Author	Year	Sandbox Analysis	Scanning Method	ML Method	Utilizes Volatility	Live Detection	Automated
	Graziano	2012	No	Yes	Yes	No	Yes	Yes
YARA		2013	Yes	No	Yes	Yes	Yes	No
	Aghaeikheirabady	2014	Yes	Yes	No	No	Yes	Yes
AMAL	Mohaisen	2015	No	No	Yes	Yes	Yes	No
	Tien	2017	No	Yes	Yes	No	No	Yes
	Cohen	2017	Yes	No	Yes	No	Yes	No
	Fowler	2017	Yes	No	Yes	Yes	Yes	No
	Nou	2017	Yes	Yes	No	Yes	No	No
MemScrimper	Brengel	2018	Yes	No	Yes	Yes	Yes	No
	Murthaja	2019	Yes	Yes	No	No	Yes	No
USIM Toolkit	Pendergrass	2019	Yes	No	Yes	Yes	Yes	No
SpeakEasy		2020	No	Yes	Yes	Yes	Yes	Yes
	Lashkari	2021	Yes	Yes	No	No	Yes	Yes
	Bozkir	2021	Yes	Yes	No	No	Yes	No
	Arfeen	2022	Yes	Yes	No	No	Yes	No
FilelessMalwareDetector		2023	No	No	Yes	Yes	No	Yes

Conclusion:

This proposed work presents a novel approach for the detection of fileless malware within a system through the analysis of RAM dumps. The methodology involves the collection of memory samples from the target system, facilitated by applications such as FTK Imager and DumpIt. The primary analytical tool employed in this process is Volatility, which enables the examination of various aspects within the collected memory samples, including processes, parent processes, browser history, and other pertinent details.

An integral component of this study involves incorporating machine learning techniques to identify both fileless and file-based malware, utilizing a meticulously curated dataset.The Random Forest algorithm, employed in this context, demonstrates a remarkable overall accuracy of 93.33%, accompanied by a True Positive Rate (TPR) of 87.5% at a zero False Positive Rate (FPR).

Furthermore, the entire workflow, encompassing RAM dump analysis and machine learning predictions, has been automated for efficiency and scalability through the utilization of AWS services, including S3 buckets, EC2 instances, and Lambda functions. Complementary tools, such as bash scripts, are also employed to streamline the process. Additionally, the Flask server, hosted on PythonAnywhere, is equipped with a user-friendly and interactive interface created using PyQt. This interface is further converted into a standalone executable file via pyInstaller, ensuring the tool's compatibility and accessibility across various systems.

References

1. Saad S, Mahmood F, Briguglio W, &. Elmiligi, H. 2019. Jsless. A tale of a fileless javascript memory-resident malware. In: Inf Sec Pract Experience. Proceedings of the 15: 15th International Conference, ISPEC 2019, Kuala Lumpur, Malaysia, Nov 26-28, 2019. Springer International Publishing. p. 113-31.
2. Saad S &. Briguglio, William &Elmiligi, Haytham. 2019. The curious case of machine learning in malware detection.
3. Menendez HD. Malware: the never-ending arm race. Open J Cybersecurity. 2021:1-25. doi: 10.46723/ojc.1.1.3.
4. Sudhakar S, Kumar S. An emerging threat Fileless malware: a survey and research challenges. Cybersecurity. 2020;3(1):1-12. doi: 10.1186/s42400-019-0043-x.
5. Anand H. File-less malware detection [doctoral dissertation]; 2022.
6. Parichha PK. Introduction to digital forensics. In: Big data analytics and computing for digital forensic investigations. CRC Press; 2020. p. 1-19.
7. Sihwail R, Omar K, Ariffin KZ. A survey on malware analysis techniques: static, dynamic, hybrid and memory analysis. Int J Adv Sci Eng Inf Technol. 2018;8(4-2):1662-71.
8. Dave R, Mistry NR, Dahiya MS. Volatile memory based forensic artifacts & analysis. Int J Res Appl Sci Eng Technol. 2014;2(1):120-4.
9. Thethi N, Keane A. Digital forensics investigations in the Cloud IEEE International Advance Computing Conference (IACC), Gurgaon, India, 2014; 2014. p. 1475-80. doi: 10.1109/IAdCC.2014.6779543.
10. Faiz MN, Prabowo WA. Comparison of acquisition software for digital forensics purposes. KINETIK. 2018;4(1):37-44. doi: 10.22219/kinetik.v4i1.687.
11. Khalid O, Ullah S, Ahmad T, Saeed S, Alabbad DA, Aslam M et al. An insight into the machine-learning-based fileless malware detection. Sensors(Basel).2023;23(2):612.doi:10.3390/s23020612, PMID 36679406.
12. Nyholm H, Monteith K, Lyles S, Gallegos M, DeSantis M, Donaldson J et al. The evolution of volatile memory forensics. J Cybersecurity Privacy. 2022;2(3), 5C56-572:556-72. doi: 10.3390/jcp2030028.
13. DeSantis M, Lyles S, Gallegos M, Nyholm H, Taylor C, Donaldson J et al. Machine learning analysis of memory images for process characterization and malware detection (No. LLNL-CONF-830179). Livermore, CA: Lawrence Livermore National Laboratory (Lawrence Livermore National Laboratory, Office of Science); 2022.
14. Dener M, Ok G, Orman A. Malware detection using memory analysis data in big data environment. Appl Sci. 2022;12(17):8604 .doi:10.3390/app12178604.

15. Carrier T, Victor P, Tekeoglu A, Lashkari AH. Detecting Obfuscated Malware using Memory Feature Engineering. InIcissp 2022 Feb 9 (pp. 177-188).
16. Craik FI. A "levels of analysis" view of memory. InCommunication and affect 1973 Jan 1 (pp. 45-65). Academic Press.
17. Louk MHL, Tama BA. Tree-based classifier ensembles for PE malware analysis: A performance revisit. Algorithms.2022; 15(9):332. doi: 10.3390/a15090332.
18. Talukder MA, Hasan KF, Islam MM, Uddin MA, Akhter A, Yousuf MA et al. A dependable hybrid machine learning model for network intrusion detection. J Inf Sec Appl. 2023;72:103405. doi: 10.1016/j.jisa.2022.103405.
19. Shafin SS, Karmakar G, Mareels I. Obfuscated memory malware detection in resource-constrained IoT devices for smart city applications. Sensors (Basel). 2023;23(11):5348. doi: 10.3390/s23115348, PMID 37300073.
20. Khalid O, Ullah S, Ahmad T, Saeed S, Alabbad DA, Aslam M et al. An insight into the machine-learning-based fileless malware detection. Sensors (Basel). 2023;23(2):612. doi: 10.3390/s23020612, PMID 36679406.
21. Saad S, Briguglio W, Elmiligi H 2019. The curious case of machine learning in malware detection. arXiv preprint arXiv:1905.07573. doi: 10.5220/0007470705280535.
22. Kamal MA, Raza HW, Alam MM, Su'ud MM. Highlight the features of AWS, GCP and Microsoft Azure that have an impact when choosing a cloud service provider. Int J Recent Technol Eng. 2020;8(5):4124-32. doi: 10.35940/ijrte.D8573.018520.
23. Marinescu DC. Cloud computing: theory and practice. Morgan Kaufmann; 2022.
24. Comparative data security measures in various cloud computing platforms. Available from: https://ijarcce.com/wpcontent/uploads/2022/10/IJARCCE.2022.11913.pdf.
25. Giménez-Alventosa V, Moltó G, Caballer M. A framework and a performance assessment for serverless MapReduce on AWS Lambda. Future Gener Comput Syst. 2019;97:25974.doi:10.1016/j.future.2019.02.057.
26. Nyholm H, Monteith K, Lyles S, Gallegos M, DeSantis M, Donaldson J et al. The evolution of volatile memory forensics. J Cybersecurity Privacy.2022;2(3):556-72.doi:10.3390 /jcp2030028.