

Detecting Obfuscated Malware using Memory Feature Engineering

Tristan Carrier¹, Princy Victor¹, Ali Tekeoglu²^a and Arash Habibi Lashkari¹^b

¹Canadian Institute for Cybersecurity (CIC), University of New Brunswick (UNB), Fredericton, NB, Canada

²Johns Hopkins University Applied Physics Laboratory, Critical Infrastructure Protection Group, Maryland, U.S.A.

Keywords: Obfuscated Malware, Memory Analysis, Ensemble Learning, Malware Detection, Stacking, Machine Learning

Abstract: Memory analysis is critical in detecting malicious processes as it can capture various characteristics and behaviors. However, while there is much research in the field, there are also some significant obstacles in malware detection, such as detection rate and advanced malware obfuscation. As advanced malware uses obfuscation and other techniques to stay hidden from the detection methods, there is a strong need for an efficient framework that focuses on detecting obfuscation and hidden malware. In this research, the advancement of the *VolMemLyzer*, as one of the most updated memory feature extractors for learning systems, has been extended to focus on hidden and obfuscated malware used with a stacked ensemble machine learning model to create a framework for efficiently detecting malware. Also, a specific malware memory dataset (MalMemAnalysis-2022) was created to test and evaluate this framework, focusing on simulating real-world obfuscated malware as close as possible. The results show that the proposed solution can detect obfuscated and hidden malware using memory feature engineering extremely fast with an Accuracy and F1-Score of 99.00% and 99.02%, respectively.


1 INTRODUCTION


Since the advent of Malware in the 1980s, it has become one of the focal points in the field of cybersecurity. Malware is any malicious software used by cybercriminals that harms the system or user by performing various criminal activities. With the fast advancement of technology and internet access, malware has also evolved regardless of the available security measures (Statista, 2021). Moreover, their ability to evade from the detection methods has made the process of malware detection complex. With diversity in malware categories and families, it is essential to cover all the bases.

There are several Malware categories such as Worms, Viruses, Bots, Botnets, Trojan Horses, Ransomware, Spyware, Rootkits, etc. As malware families has several functionalities like infiltrating the system, gaining access to information, preventing access for authorized users, or performing other cybercrimes within each category, the best solution to detect them should focus on both categories and families to prevent and stop them in the future.

As the complexity and time consumption of manual detection methods are very high, different learning systems like machine learning and deep learning are used, which can produce intelligent insights from the data automatically. The primary importance of these learning systems is figuring out which training data to feed to the system to make the quickest and most accurate assessment. Machine learning systems take in a set of features that can be looked at with a large sample size to compare and contrast differences. Furthermore, these features can be input in different formats, which is a factor in determining the machine learning system that should be used. While some machine learning algorithms are focused on speed, others are focused on accuracy and precision. Therefore, selecting the different algorithms that correspond to the objective and input type has an enormous impact on the results of the system. Ensemble machine learning is one such model in machine learning that correlates well with these goals of detection and characterization.

There exists several approaches for obfuscated malware detection based on memory analysis. However, in most of the works, the complexity and time consumption are high, which makes them not suitable

^a <https://orcid.org/0000-0001-7638-0941>

^b <https://orcid.org/0000-0002-1240-6433>

for real-world application. This is the motivation to propose a fast, efficient and easy to develop solution for obfuscated malware detection by making use of the most effective features captured through memory analysis.

Main Contributions: The main contributions of this research include:

- Proposing a malware analysis framework that uses a two-layer stacked ensemble learning model to improve the current obfuscated or hidden malware detection solution.
- Proposing 26 new memory-based features for the only available open-source memory analyzer for learning systems, VolMemLyzer, by focusing specifically on the obfuscated and hidden malware detection and implementing the new version of the open-source project, VolMemLyzer-V2.
- Generating and releasing a comprehensive dataset by executing more than 2500 malware samples on three common obfuscated and hidden categories, including Spyware, Ransomware, and Trojan Horse, to test and evaluate the proposed framework.

The structure of this paper is shown as follows. Section 2 introduces the related works on memory analysis models that used machine learning and deep learning for malware detection. Section 3 proposes an obfuscated and hidden malware detection framework that tackles the challenges identified in this study. Section 4 presents the dataset creation process and the malware types, families, and samples, while section 5 presents the experimental analysis. Finally, section 6 concludes the paper by discussing the findings and future works.

2 LITERATURE REVIEW

Since the inception of malware, it has gained enormous attention in the cybersecurity field due to its various delivery methods and categories. Although there exist several detection methods, each carries its challenges. This section highlights the related works on malware detection through memory analysis and discusses the remaining issues and challenges in this research field.

2.1 Malware Memory Analysis

Memory analysis is a method that provides a strong understanding of the activities in the system by capturing memory snapshots and extracting features from them. (Shree et al., 2021) discusses the reliability

in malware memory analysis since all critical information is stored in memory. The memory analysis that is not performed live needs snapshots, and obtaining these snapshots is essential to ensure that the memory files are not affected. Affected memory files could change the results of the memory analysis process and would remove the reliability of the analysis taking place.

(Stüttgen and Cohen, 2014) proposed a framework that shows how to capture a memory from a Linux system with minimal impact by using a relocation hooking that can copy the information safely. Furthermore, since this technique doesn't require the installation of an environment on the system, those tasks will not be in the analyzed memory. In addition to the memory snapshot capturing difficulties, automation and the complexity of the analysis process are other challenges. As a solution for this, (Socala and Cohen, 2016) explains the method of automatic profile generation for live memory analysis, which can automate the analysis process in a viable manner.

Moreover, the work by (Okolica and Peterson, 2010) discusses the importance of having a highly flexible memory analysis process that can work on different platforms and systems as this would significantly reduce the amount of time needed to match the system with the profile. Furthermore, the work also discovered debugging structures on memory analysis to allow the tools to run on more systems. In another work, (Block and Dewald, 2017) introduced a memory analysis plugin that can use to simplify the analysis process. This plugin focuses on the details of the heap objects in memory, and these heap objects can help a memory analysis professional understand what undergoes in the system memory.

When the memory has been successfully captured, the next step to consider is how to extract the data from within it. (Okolica and Peterson, 2011) explains the importance of DLLs and Windows drivers, which are difficult to extract with no entry point to gain access, especially with no export functions. To get the information from these drivers, a huge work is needed from a memory forensic professional. The authors show the method of reversing the drivers to gain quicker and more efficient access to the driver information.

A work by (Dolan-Gavitt, 2008) discusses the importance of gaining access to the full registry in memory with the use of cell indexes. Similarly, (Zhang et al., 2011) also explains the extraction of registry information from physical memory for Windows systems and the importance of understanding the file structure. In the other work by (Zhang et al., 2009), the use of the data structure, Kernel Processor Con-

trol Region, is explained for translating the difference from virtual to physical memory in the address space, thus improving the memory forensics on windows machines. (Zhang et al., 2010) also did their study on converting virtual to physical addresses by using the paging structure for 2MB pages in a Windows 7 system.

Memory analysis can be used in many different ways to find out what happened to a victim. (Thantilage and Jeyamohan, 2017) discuss the usage of volatile memory analysis to gain information on social media evidence. The developed application focuses specifically on targeting volatile memory analysis to obtain social media evidence.

Updating lots of systems in an industry can be expensive and often goes unnoticed until an attack occurs. For this, (Sharafaldin et al., 2017) proposes a new tool called BotViz that uses a hybrid approach for detecting bots in a network. In addition to that, this model uses hooks to strengthen bot detection. The work by Martin-Perez (Martin-Perez et al., 2021) presents an interesting concept of memory dump preprocessing with two different strategies that can relocate file objects to make the analysis process quicker and easier. The first strategy, called Guided De-Relocation, specifically selects a new space for the information. The second strategy is Linear Sweep De-Relocation, which sweeps through the memory to find a storage spot. Memory forensic tools have drastically changed how memory analysis is performed; however, they can still be refined and improved to be faster, more efficient, and easier to use. (Lewis et al., 2018) discusses the method by which the defects are fixed and improvements are added to professional tools such as Volatility. Improving memory forensic algorithms to adapt to current standards is also needed as they explain many novel memory analysis algorithms.

2.2 Malware Detection Using Memory Analysis

Sometimes, dynamic and static methods of malware classification can be both inaccurate and imprecise. According to (Dai et al., 2018), the idea of using an extracted malware memory dump file that is converted into grayscale image results in higher accuracy and precision than static and dynamic methods. Results show a 20 percent increase in accuracy when converting to a grayscale image before comparison with other known malware.

According to (Yucel and Koltuksuz, 2019), using a three-dimensional heat mapping system can reduce the time it takes to classify malware along with in-

creasing its accuracy. The work discussed layering the levels on top of each other, which shows the intensity of malware in each section of the malware memory dump. Moreover, this heat map can be compared to other malware systems to show a higher accuracy detection and classification rate. As Malware classification analysis can be costly in time and accuracy, (Kang et al., 2019), suggests the use of vectoring assembly source code using the Long Short-Term Memory-Based (LSTM) method for classifying malware. Using word2vec with the LSTM system, the increase in accuracy reached 0.5 percent higher than other methodologies.

Malware detection and analysis are difficult for advanced systems; however, malware detection in a cloud is even more difficult with more liabilities. It can be hard to examine if malicious acts are happening with constant live processes running, especially while taking privacy into account. Using an unbiased training set, the minHash method was able to have a nearly perfect detection rate. With increasing cloud operations, using the minHash method can increase efficiency and reliability, as shown in the numerous experiments as (Nissima et al., 2019) has shown. In this work, the results are drastically different in detection across the different classifiers, which shows the impact of the classifiers based on the different types of malware being input into the system. To reduce this variance, classifiers can work together to make up for their weaknesses.

The current direction of remote computing leads to more information stored in the cloud; as such, they have been a bigger target for malware with an increase in demand for security. With the static and dynamic approaches not being applicable for cloud computing, the need for new security methods has increased for specialized cloud computing security. (Li et al., 2019) suggests a deep learning approach that collects a memory snapshot of the system and converts it into a grayscale image. The convolutional neural network then models the system and trains deep learning to differentiate between malicious and benign memory snapshots. Results showed that this process reduces runtime for analysis as well as accurately identifies malware. After obtaining the target virtual machine introspection, it is fed to the extracting model, which converts it to the grayscale image passed from the target VM to the secure VM for analysis.

(Sai et al., 2019) developed the concept of managing memory with API call mining. This method analyzes API calls that access the system's memory and observes the transitions in the memory to watch the management and ensure that the system does not contain any malicious activity. This method can check the

allocated memory during runtime and detect roughly 95 percent of all malicious programs from the system memory behavior.

The importance of detecting new malware is extremely high to prevent new attacks from harming systems. Many techniques have high detection rates on known malware using in-depth training techniques. However, while comparing previous works, it can be identified that the works do not deal with new never seen before malware. As a solution for this, (Si-hwail et al., 2019) suggests using memory forensics to extract artifacts from memory combined with memory feature extraction. Based on past known malware and the extracted artifacts, the framework can determine what future malware will consist of. Results showed that the model has an extremely high detection rate and accuracy while still keeping a low amount of time needed to run.

As some malware like Objective-C malware, also known as userland, puts MacOS X systems at risk, (Case and Richard, 2016) proposed a plugin for the Volatility framework that focuses on automatically analyzing the artifacts of the system that have importance. This is done by monitoring the Objective-C at runtime and outputting a file that can be analyzed. Based on this file, it can be examined and determined how to deal with the current situation. This results in a fast analysis time and less work for the analysts, thus allowing more systems to be monitored in the same amount of time. As typical Malware detection and unpacking tools can be detected from the malware debuggers, malware stays dormant during scans and avoids malware detection methods.

However, according to (Kawakoya et al., 2010), while using the stealth debugger, malware is not aware when to stay dormant or when to run to avoid malware detection scans. In addition to that, the stealth debugger takes the virtual machine memory and sends it to the guest operating system. After which, it runs the analysis to identify the true origins of the code. Since most malware is advanced enough to contain obfuscation methods, this model can detect most packers at an incredibly high accuracy rate, with some packers getting a perfect detection rate. While static and dynamic approaches are a good start for detecting malware, they can often be exploited by obfuscated malware, leading to malware deactivating the detection methods. Using application-specific detection with machine learning, (Xu et al., 2017) was able to get nearly a perfect malware detection rate. This method works on the top layer and works down to the kernel level, where many corruption attacks can occur. With this approach, corruption attacks were stopped 99 percent of the time with less than a five

percent false positive rate.

To combat the malware obfuscation techniques, the detection method needs to be designed with obfuscation in mind. This can be done using a specifically designed dataset to test how well a detection system deals with obfuscated malware. (Sadek et al., 2019) challenged detection methods by using a large dataset that consists of positive and negative memory snapshots, advanced payload systems, and malware obfuscation. (Bozkir et al., 2021) have come up with a novel approach that uses an RGB image to show memory dump files in their malware detection system.

While using the manifold learning technique called UMAP, (Javaheri and Hosseninzadeh, 2017) identified the original memory dump file showing malicious or benign activity. After testing with ten malware families and benign samples, the results were roughly 96 percent accuracy at the extremely fast speed of only 3.56 seconds. Moreover, a framework was also developed to combat the obfuscation of malware. Using the detection presence time of the malware at each level of the operating system down to the kernel, they were able to dump the malware memory at the precise time and view the malware installation. The framework was focused specifically on obfuscation and packaging in mind to challenge one of the biggest problems in malware detection. After testing the framework, it obtained roughly 85 percent accuracy in detecting kernel-level malware. Though there are many different methods to detect obfuscated malware, each method has to be looked into for different situations.

Malware and botnets can be difficult to blacklist when they use obfuscation and concealment. Botnet command and control servers can also make a real-time prediction for domain names extremely challenging. (S et al., 2019), discusses the use of a framework to counter obfuscation by using the LSTM network. This framework operates for both binary and multi-class data with a high recall rate and precision, producing a good F1 score. This F1 score consists of over 80 percent for binary class data and over 60 percent for multi-class data. Moreover, this framework can be used to help identify concealed and obfuscated malware in botnet systems.

VMSHield, a proposed method by (Mishra et al., 2021), protects virtual domains in the cloud from obfuscated and stealthy malware attacks. This work used a state-of-the-art method that collects runtime behavior from the different processes and analyzes the results to make obfuscated and stealthy malware unable to sneak past detection. Passing down to the system, VMSHield is able to monitor the results of

each layer and trace all of the system calls and extract the features that are the biggest impact on the system. VMShield can detect more than 97 percent of the attacks using these introspection techniques, including hidden and obfuscated attacks. VMShield cloud protection process step by step, where it discusses the tracing of the hypervisor from the virtual machine, feature extraction, selection process, and profile generation. Finally, VMshield obtains the result of the model and delivers a status report that can be looked by the admin.

Virtual machine introspection has become a common tactic with detecting malware and other malicious sources as it can miss hidden, dead, or obfuscated malware. With the use of a virtual machine monitor, otherwise known as a hypervisor, (Kumara and Jaidhar, 2016), discusses an automated internal and external system that can detect hidden, dead, and obfuscated malware inside the virtual machine with the aid of machine learning. After testing the system with an advanced data set using cross-validation, the authors found that their system has a 99.55 percent accuracy rate while still holding the extremely low false-positive rate of 0.004 percent.

There exist works like (Sklavos, 2017) that discusses the security issues in IoT devices by studying the malware for both system hardware and software. In this work, the most widespread malware categories, such as logic bombs, rootkits, bots, etc., were discussed from a software viewpoint. In addition to that, the hardware security in IoT devices was also studied by mentioning the power monitoring attacks, timing attacks, etc. The work also presented the existing malware detection approaches and summarized expected future directions.

Overall, it can be identified that several approaches exist for obfuscated malware detection based on memory analysis. To the best of our knowledge, no literature focused on the detection in the memory through feature extraction, as the methods used are very complex and time-intensive. It is also interesting to notice that the works have focused on detecting malware found in different system layers for general and obfuscated malware cases. The VolMemLyzer was developed as the first memory-based malware analysis feature extractor for learning-based solutions, but it did not focus on obfuscated malware analysis (Lashkari et al., 2020). As a result, this work proposes an obfuscated malware memory analysis framework that focuses on a fast and low-cost solution that will be discussed in the next section.

3 PROPOSED APPROACH

In most existing works, the complexity and time consumption are high, making them unsuitable for real-world application. As a solution for this, a fast, efficient, and easy to develop solution for obfuscated malware detection is proposed in this paper by using the most effective features captured through memory analysis.

3.1 General Overview

The overview of this obfuscated malware detection framework is depicted in Figure 1. The components of the proposed framework include:

- **Memory Dump File:** Memory dumps can be obtained by using programs such as MAGNET RAM, ManTech Memory DD, Forensic Tool Kit (FTK), or virtual machine managers with the memory capturing feature. This is a snapshot showing the activity that took place in memory on the system (MAG, 2021)(Man, 2021)(For, 2021).
- **Volatility:** is a completely open collection of tools, implemented in Python under the GNU General Public License, to extract digital artifacts from volatile memory (RAM) samples. (Vol, 2016).
- **VolMemLyzer-V2:** The memory feature extractor for learning-based solutions with the 26 new features implemented as part of the proposed model to target obfuscated and hidden malware. VolMemLyzer extracts the features using volatility plugins and generates a CSV file(Lashkari et al., 2020).
- **CSV Feature File:** This is the output from the VolMemLyzer feature extractor, which contains all the features that have been extracted in a compact comma-separated values file (CSV).
- **Ensemble Learning:** A machine learning technique that focuses on combining classifiers to cover its weaknesses. As some classifiers are easily swayed by outliers or have a high bias, ensemble learning allows these weaknesses to have less impact on the overall results (Ens, 2021). The stacking ensemble technique was used for this framework which has two layers of classifiers.
- **Malicious and Benign Classification output:** The binary output for each memory dump file that shows whether there is a malicious activity or benign activity.

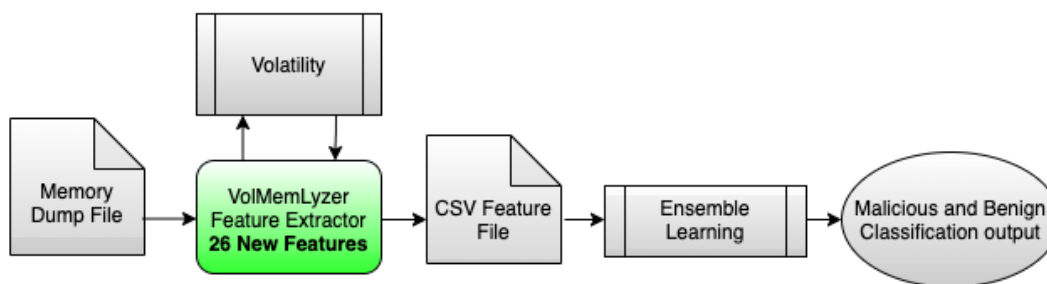


Figure 1: Malware Memory Analysis Process.

3.2 Proposed Features

The first step in this model is to obtain the memory dump, which is compatible with the current Volatility version 2.6 (Vol, 2016) used for the framework with the VolMemLyzer (Lashkari et al., 2020). The memory dumps are then passed to the VolMemLyzer feature extractor, which uses Volatility to extract 58 features. Among these 58 features, the newly added 26 features specifically focus on targeting obfuscated and hidden malware, as explained in this section.

There are five different categories that these features belong to from Volatility. The first category is called Malfind, which detects potential malicious executables that are usually DLLs associated with Trojan malware. The next category is called Ldrmodule, which gives information on potential injected code into the system, which is often the way which spyware enters the system. The Handle category is the one that looks at the type of information in memory and its classification. The Process View features show the process list with information that can be used to find malicious processes. The last feature type is the API-hook features which show the total number of API-hooks of key types. Table 1 depicts the features extracted for the memory analysis framework.

3.3 Detection Model

Once these features are extracted, they are ready to be fed into the proposed ensemble learning. The ensemble learner has two stages, being the training stage and the validation stage. In the training stage, the ensemble learner runs the base learners, and the prediction results from these classifiers are used as input for the second layer classifier. After the ensemble learner is trained, it is then validated, going through the same process as the training to validate the data set results and ensure that the training was successful.

There exist several ensemble learning techniques such as stacking, voting, boosting, bagging etc. In voting, the weights given by the user are used to combine the classifiers, whereas stacking achieves this

aggregation with a meta classifier. In this proposed work, stacking is selected as the method for ensemble learning due to the speed and variance performance. As mentioned above, the first layer runs individually, allowing them to run in tandem and reduce the time needed for classification. The second layer deals with a slight variance of inputs which allows it to be run quickly after the first layer is finished. The different classifiers can compensate for each other's weaknesses, keeping the accuracy high while classifying fast.

As there are a plethora of machine learning classifiers, it is vital to identify the suitable classifier that can be applied to the proposed model. Usually, the classifier is selected based on the type of dataset used for classification. This work's chosen classifiers are SVM, Decision Tree, Linear Perceptron, Naive Bayes, Random Forest, and KNN as base learners, experimenting with different combinations. However, only three were selected at a time, as the proposed model developed using Python is built to use three base learners and one meta learner. Similar to base learners, different meta-learners such as SVM, KNN, Naive Bayes, and Logistic Regression were also used for finding the best classifier. All these classifiers were chosen with the goal in mind for speed and accuracy. The best combination among these will be selected in the experimentation phase. Finally, the meta learner outputs the binary results showing whether the memory snapshot was malicious or benign. Figure 2 shows the different base learners and meta learners used in the proposed model.

4 CREATING A NEW DATASET

As the proposed malware detection framework focuses explicitly on targeting obfuscated malware, a dataset is developed to simulate real-world conditions close to the malware found in the real world.

Table 1: Extended Feature List.

Feature Type	Feature List	Feature Discription
Malfind	commitCharge	Total number of Commit Charges
	protection	Total number of protection
	uniqueInjections	Total number of unique injections
Ldrmodule	avgMissingFromLoad	The average amount of modules missing from the load list
	avgMissingFromInit	The average amount of modules missing from the initialization list
	avgMissingFromMem	The average amount of modules missing from memory
Handles	port	Total number of port handles
	file	Total number of file handles
	event	Total number of event handles
	desktop	Total number of desktop handles
	key	Total number of key handles
	thread	Total number of thread handles
	directory	Total number of directory handles
	semaphore	Total number of semaphore handles
	timer	Total number of timer handles
	section	Total number of section handles
	mutant	Total number of mutant handles
Process View	pslist	Average false ratio of the process list
	psscan	Average false ratio of the process scan
	thrdproc	Average false ratio of the third process
	pspcid	Average false ratio of the process id
	session	Average false ratio of the session
	deskthrd	Average false ratio of the deskthrd
Apihooks	nhooks	Total number of apihooks
	nhookInLine	Total number of in line apihooks
	nhooksInUsermode	Total number of apihooks in user mode

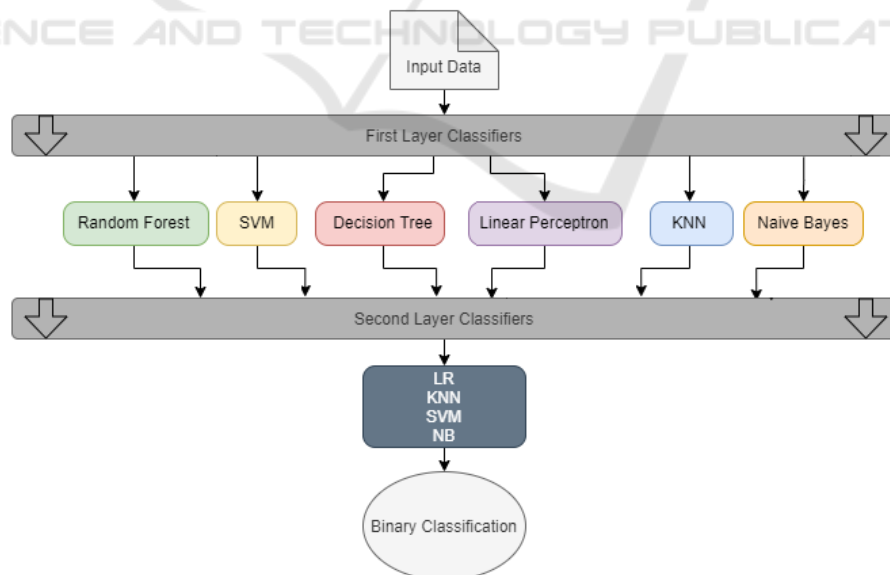


Figure 2: Stacked Ensemble Learning Classifiers.

4.1 Overview

In this research, a dataset (MalMemAnalysis-2022) of 58,596 records with 29,298 benign and 29,298

malicious records is created by capturing malicious as well as benign dumps. For capturing malicious memory dumps, 2,916 malware samples collected

from VirusTotal that have different malware categories including Ransomware, Spyware, and Trojan Horse as listed in the Table 2, are executed in a VM with 2 GigaBytes of memory. Similarly, for the creation of benign memory dumps, normal user behaviour is captured by using various applications in the machine. The detailed process will be discussed in the below section. (New dataset will be available in “<https://www.unb.ca/cic/datasets/MalMem-2022.html>”).

Table 2: Malware sample count.

Malware Category	Malware Families	Count
Trojan Horse	Zeus	195
	Emotet	196
	Refroso	200
	scar	200
	Reconyc	157
Spyware	180Solutions	200
	Coolwebsearch	200
	Gator	200
	Transponder	241
	TIBS	141
Ransomware	Conti	200
	MAZE	195
	Pysa	171
	Ako	200
	Shade	220

4.2 Creating Dataset

Four main steps were considered in this dataset creation: research, memory dump extraction, memory dump transfer, and feature extraction.

- First step is the research of the malware category, family, and sample type. It is important to have malware that simulates as close to a real-world example as possible. As such, malware designed to specifically target old systems that are no longer in use and do not work on newer systems would not accurately detect the malware of current systems. This is the reason why in-depth research was done on each family and type of malware. Based on the research, we have collected a minimum of 100 and a maximum of 200 malware samples from five different families in three malware categories: Trojan Horse, Ransomware, and Spyware.
- The second step is memory dumping. The memory dump can be activated outside the virtual machine, where the memory snapshot is captured from using the VirtualBox virtual machine man-

agement system. This ensures that the memory dump is not contaminated with a process usually not on the typical system. The memory dump is captured from a Windows10 system rather than a windows XP or older system that is not used as much. This is to ensure that the malware being tested is as close to a real-world simulation as possible. To expand the dataset, this process was automated where 2,916 malware samples from three malware categories including Trojan Horse, Ransomware, and Spyware were executed in the VM. As it is important to have some benign processes executed during the malicious memory dump creation, different applications in Windows VM were opened along with executing the malware samples. This is done to make sure that the classifier is not able to determine the difference just based on the benign processes alone. For each sample execution, 10 memory dumps, each with a 15 seconds gap, were captured to ensure no malware behaviour is left out, and extracted 29,298 malicious memory dumps. For benign dumps, normal user behaviour is captured by using different applications in the machine and performed oversampling using SMOTE algorithm to make the dataset balanced. Unlike other oversampling methods, SMOTE does not generate duplicates instead produces synthetic values that are negligibly distinct from the actual values.

- The third step consists of transferring the resulting memory dump files to a Kali Linux machine to perform the feature extraction using the VolMemLyzer with the 26 new features added to target malware obfuscation.
- The fourth main step on the initial process was the feature extraction of the memory dump files and the creation of the final combined CSV file for all tested memory dump files, which is to be used in the ensemble learning system. After the memory dumps were acquired, the VolMemLyzer feature extractor ran on all the memory dumped files in the folder and generated the resulting CSV file to be used in the ensemble learning system.

section EXPERIMENTS To finalize our proposed model, we have used the newly created dataset. The detailed experimental setup, along with the finalized model, is discussed in the below sections.

4.3 Experimental Setup

A python code and a bash script are used to execute the malware samples on a 64-bit Windows 10 isolated virtual machine inside Oracle Virtual Box and cap-

tured the local machine’s memory dumps. For the feature extraction, we created the CSV file with features from the captured memory dump using VolMemLyzer feature extractor for learning systems, publicly available on GitHub (Lashkari et al., 2020). In addition to that, for developing stacked ensemble learners, python was used with the Sklearn library and deployed in the Jupyter Notebook IDE for simplifying the development of model (skl, 2021).

4.4 Finalizing The Proposed Model

This section finds the best combination of base learners and meta learners by performing several experiments. First, each base-learner is evaluated using the created dataset, and results are analyzed using different evaluation metrics, including Accuracy, weighted average Precision, weighted average Recall, and F1-score, as shown in Table 3. From the results, it can be identified that RF, Decision Tree, and KNN exhibited better performance, whereas Linear Perceptron has the least performance.

Table 3: Individual Classifiers Result.

Classifiers	Pre.	Rec.	F1	Acc.
RF	0.98	0.97	0.97	0.97
NB	0.92	0.92	0.92	0.92
DT	0.97	0.97	0.97	0.97
KNN	0.95	0.95	0.95	0.95
SVM	0.91	0.90	0.90	0.90
LP	0.61	0.59	0.53	0.60

To select the best stacking model and finalize it, different combinations of base-learners and meta-learners were considered and the top four highest accuracy results among them are shown in Table 4. Results proved that the performance of the model is increased when ensemble methods like stacking are used. One of the main goals of the research was to focus on overall speed of which classification speed has been optimized. The classifiers that were considered had a fast classification speed while also able to work on a large variance of data for the first layer. The second layer classifiers didn’t need a large variance so strictly speed and accuracy was looked into. Stacking was chosen to satisfy the goal of a fast speed in classification since stacking allows first layer classifiers to run in parallel. At a fast speed, it is able to check the results from multiple classifiers and determine the right binary classification. Moreover, it is also identified that although some classifiers perform poorly, their performance can be enhanced when combined with some other classifiers.

Hence, based on the results, the selected classifiers

Table 4: Ensemble Model Comparison.

Base Learner	Meta Learner	Pre.	Rec.	F1	Acc.
NB, LP, DT	SVM	0.96	0.95	0.95	0.95
SVM, LP, DT	KNN	0.97	0.96	0.96	0.96
NB, LP, RF	LR	0.98	0.97	0.97	0.97
NB, RF, DT	LR	0.99	0.99	0.99	0.99

were Naive Bayes, Random Forest, and Decision Tree for the base-learners and Logistic regression as the meta-learner in the finalized model. Figure 3 shows a confusion matrix representing the true positive, false positive, false negative and true negative.

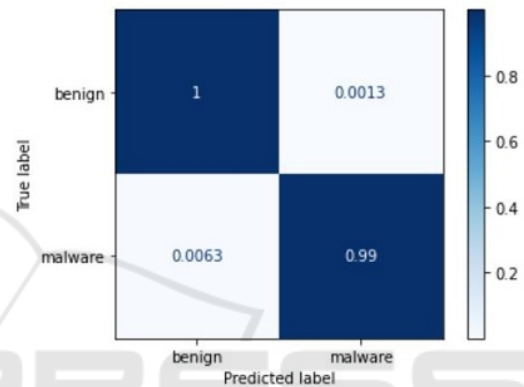


Figure 3: Confusion matrix results.

Furthermore, since the memory dump used for feature extraction is taken from a Windows10 system, the size of each memory dump is 2GB which is quite large. However, the resulting CSV files’ size is around 2KB, allowing for scalability in the proposed model. Moreover, this model could solve the complexity and time-consuming issues found in the existing works.

Table 5 shows the approximate time for samples to be classified as malicious or benign. From this, it can be identified that the classification time displayed is linear, which proves the model’s scalability. Overall, the proposed stacked ensemble model has a classification time of around 0.008 milliseconds per sample, which is significantly lower than most of the other models listed in the related works (the second column of the Table 6).

4.5 Comparing Proposed Model With Relevant Studies

The comparison mainly focused on existing works that targeted obfuscated malware or hidden malware that belonged to one of the three categories of Ransomware, Spyware, and Trojan Horse malware. Table 6 shows the results of the four various related works

Table 5: Approximate Classification Time Based On The Number Of Samples.

No. of samples	Time (Sec)
50	0.4
100	0.8
150	1.2
200	1.6
250	2
300	2.4

in the targeted area. Due to the differences in datasets and approaches, the exact results are difficult to compare and often biased towards a specific method.

The four criteria for comparison were overall accuracy of the technique, overall speed per sample, memory usage needed to classify a sample, and overall model complexity. Based on these criteria, each work was given a result on a scale from very low to very high. These four related works are then compared to the proposed model, Detecting Obfuscated Malware using Memory Feature Engineering.

The first work selected for comparison is the work by (Xu et al., 2017) which has very impressive accuracy and medium complexity. However, the memory usage is high, and the work did not mention the speed. In the second work, (Bozkir et al., 2021) detected obfuscated malware with high accuracy and a rate of 3.56 seconds to detect per sample. Although the complexity of this model is high, the primary concern is the memory usage as the system needs to store the RGB image representation for each sample and three different images for the creation of the final image. The third work is from (Javaheri and Hosseninzadeh, 2017) which has a model that requires a lower amount of memory with high speed. However, it is worth noting that this framework does have increased complexity and has less accuracy than the previous methods. The fourth work is from (Nissima et al., 2019), and the model exhibits a high accuracy with lower complexity compared to the previous two methods. However, it is slower, and the amount of memory used is not mentioned.

Comparing these related works shows the importance of a fast detection method for obfuscated or hidden malware. The high amount of obfuscated malware has made speed and scalability a requirement rather than a luxury. From the results, it can be identified that the new features and stacked machine learning model improve the overall accuracy for obfuscated and hidden malware detection.

5 CONCLUSIONS

As networking and the internet evolved, malware authors swiftly adapted their malicious code, and most of them are used to exploit vulnerabilities in Microsoft Windows. Although several techniques exist to detect obfuscated or hidden malware based on memory analysis, the time consumption and complexity of the works are high. As a solution for this problem, obfuscated malware detection model is proposed, which extracts features from memory dumps using VolMemLyzer, a feature extractor for learning systems. For this, a dataset (MalMemAnalysis-2022) was constructed by executing malware samples from three main categories, Spyware, Ransomware, and Trojan Horse malware, in an isolated virtual machine.

The model was designed to use memory feature engineering with a stacked ensemble learner to achieve the goals of this research. Different combinations of base-learners and meta-learners were used using the created dataset, and the final model was selected based on different evaluation metrics. The best results were exhibited when Naive Bayes, Decision Tree, and Random Forest were used as base learners and Logistic Regression as the meta-learner with an accuracy of 99%. Moreover, this model is compared with related works that focus on obfuscated malware detection in memory. The comparison results showed that the proposed model has less classification time and better performance.

5.1 Future Work

Detecting obfuscated malware in memory using feature engineering with the VolMemLyzer shows a quick and precise way of dealing with malware that is attempting to hide in memory. The next step of this research is to work with more advanced designer malware that is specifically designed for different systems, including Mac and Linux-based systems. Moreover, this can ensure that older systems are protected with this detection model and can be incorporated with upcoming systems. This would protect most systems from obfuscated malware attacks by focusing on automated detection. With ransomware running rampant in today's society, ensuring that this malware is detected and dealt with before it causes harm is extremely important. The speed of this model can help detect the malware before such harm is caused and thus reduce overall harm.

Table 6: Comparison Table With Other Obfuscated Malware Detection Methods.

Method	Accuracy	Speed	Memory Usage	Complexity
(Xu et al., 2017)	Very High	Not Mentioned	High	Medium
(Bozkir et al., 2021)	High	Medium	Very High	High
(Javaheri and Hosseninzadeh, 2017)	Medium	High	Low	High
(Nissima et al., 2019)	High	Medium	Not Mentioned	Medium
Proposed Model	High	Very High	Low	Medium

ACKNOWLEDGEMENTS

We thank the Mitacs Program for providing the Global Research Internship (GRI) opportunity to support this project.

REFERENCES

- (2016). Volatility framework - volatile memory extraction utility framework. <https://github.com/volatilityfoundation/volatility>. (Accessed on 08/10/2021).
- (2021). Ensemble methods in machine learning: What are they and why use them? <https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f5>. (Accessed on 08/10/2021).
- (2021). Ftk® forensic toolkit: The gold standard in digital forensics for over 15 years. <https://www.exterro.com/forensic-toolkit>. (Accessed on 08/9/2021).
- (2021). Magnet ram capture: What does it do? <https://www.magnetforensics.com/resources/magnet-ram-capture/>. (Accessed on 08/11/2021).
- (2021). Mantech memory dd version 1.3 for forensic analysis of computer memory. <https://investor.mantech.com/press-releases/press-release-details/mantech-memory-dd-version-13-forensic-analysis-computer-memory>. (Accessed on 08/12/2021).
- (2021). Understanding logistic regression in python. <https://realpython.com/logistic-regression-python/>. (Accessed on 08/10/2021).
- Block, F. and Dewald, A. (2017). Linux memory forensics: Dissecting the user space process heap. *Digital Investigation*, 22:pp. 66–75.
- Bozkir, A. S., Tahilioglu, E., Aydos, M., and Kara, I. (2021). A malware detection approach through memory forensics, manifold learning and computer vision. *Science Direct*, 103.
- Case, A. and Richard, G. G. (2016). Detecting objective-c malware through memory forensics. *Digital Investigation*, 18.
- Dai, Y., Li, H., Qian, Y., and Lu, X. (2018). A malware classification method based on memory dump grayscale image. *Digital Investigation*, 27:pp. 30–37.
- Dolan-Gavitt, B. (2008). Forensic analysis of the windows registry in memory. *Digital Investigation*, 5:pp. 26–32.
- Javaheri, D. and Hosseninzadeh, M. (2017). A framework for recognition and confronting of obfuscated malwares based on memory dumping and filter drivers.
- Kang, J., Jang, S., Li, S., Jeong, Y.-S., and Sung, Y. (2019). Long short-term memory-based malware classification method for information security. *Computers and Electrical Engineering*, 77.
- Kawakoya, Y., Iwamura, M., and Itoh, M. (2010). Memory behavior-based automatic malware unpacking in stealth debugging environment. *5th International Conference on Malicious and Unwanted Software, Nancy, Lorraine*, pages pp. 39–46.
- Kumara, A. and Jaidhar (2016). Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. *Digital Investigation*, 23:pp. 99–123.
- Lashkari, A. H., Li, B., Carrier, T. L., and Kaur, G. (2020). Volatility memory analyzer. <https://github.com/ahlashkari/VolMemLyzer>.
- Lewis, N., Case, A., Ali-Gombe, A., and III, G. G. R. (2018). Memory forensics and the windows subsystem for linux. *Digital Investigation*, 26:pp. 3–11.
- Li, H., Zhan, D., Liu, T., and Ye, L. (2019). Using deep-learning-based memory analysis for malware detection in cloud. *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW), Monterey, CA, USA*, pages pp. 1–6.
- Martin-Perez, M., Rodriguez, R. J., and Balzarotti, D. (2021). Pre-processing memory dumps to improve similarity score of windows modules. *Computer & Security*, 101.
- Mishra, P., Aggarwal, P., Vidyarthi, A., Singh, P., Khan, B., Alhelou, H. H., and Siano, P. (2021). Vmshield: Memory introspection-based malware detection to secure cloud-based services against stealthy attacks. *IEEE Transactions on Industrial Informatics*.
- Nissima, N., Lahava, O., Cohena, A., and Rokacha, Y. E. L. (2019). Volatile memory analysis using the minhash method for efficient and secured detection of malware in private cloud. *Computers & Security*, 87.
- Okolica, J. and Peterson, G. L. (2010). Windows operating systems agnostic memory analysis. *Digital Investigation*, 7:pp. 48–56.

- Okolica, J. S. and Peterson, G. L. (2011). Windows driver memory analysis: A reverse engineering methodology. *Computers & Security*, 30:pp. 770–779.
- S, A., S, S., Poornachandran, P., Krishna Menon, V., and P, S. K. (2019). Deep learning framework for domain generation algorithms prediction using long short-term memory. *ICACCS*.
- Sadek, I., Chong, P., Rehman, S. U., Elovici, Y., and Binder, A. (2019). Memory snapshot dataset of a compromised host with malware using obfuscation evasion techniques. *Data in brief*, 26.
- Sai, K. V. N., Thanudas, B., Chakraborty, A., and Manoj, B. S. (2019). A malware detection technique using memory management api call mining. *IEEE*.
- Sharafaldin, I., Gharib, A., and Lashkari, A. H. (2017). Botviz: A memory forensic-based botnet detection and visualization approach. *International Carnahan Conference on Security Technology (ICCST)*.
- Shree, R., Shukla, A. K., Pandey, R. P., Shukla, V., and Bajpai, D. (2021). Memory forensic: Acquisition and analysis mechanism for operating systems. *Materials Today: Proceedings*.
- Sihwail, R., Omar, K., Ariffin, K. A. Z., and Afghani, S. A. (2019). Malware detection approach based on artifacts in memory image and dynamic analysis. *Applied Sciences*.
- Sklavos, N. (2017). Malware in iot software and hardware. In *Workshop on Trustworthy Manufacturing and Utilization of Secure Devices (TRUDEVICE'16)*, pages 8–11.
- Socala, A. and Cohen, M. (2016). Automatic profile generation for live linux memory analysis. *Digital Investigation*, 16:pp. 11–24.
- Statista (2021). Statista: annual number of malware attacks worldwide from 2015 to 2019. <https://www.statista.com/statistics/873097/malware-attacks-per-year-worldwide/>. (Accessed on 08/10/2021).
- Stüttgen, J. and Cohen, M. (2014). Robust linux memory acquisition with minimal target impact. *Digital Investigation*, 11:pp. 112–119.
- Thantilage, R. and Jeyamohan, N. (2017). A volatile memory analysis tool for retrieval of social media evidence in windows 10 os based workstations. *National Information Technology Conference (NITC)*.
- Xu, Z., Ray, S., Subramanyan, P., and Malik, S. (2017). Malware detection using machine learning based analysis of virtual memory access patterns. *Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne*, pages pp. 169–174.
- Yucel, C. and Koltuksuz, A. (2019). Imaging and evaluating the memory access for malware. *Forensic Science International: Digital Investigation*, 32.
- Zhang, R., Wang, L., and Zhang, S. (2009). Windows memory analysis based on kpcr. *International Conference on Information Assurance and Security*.
- Zhang, S., Wang, L., and Zhang, L. (2011). Extracting windows registry information from physical memory. *3rd International Conference on Computer Research and Development*.
- Zhang, S., Wang, L., Zhang, R., and Guo, Q. (2010). Exploratory study on memory analysis of windows 7 operating system. *International Conference on Advanced Computer Theory and Engineering (ICACTE)*, 3.