RESEARCH **Open Access**

MRm-DLDet: a memory-resident malware detection framework based on memory forensics and deep neural network

Jiaxi Liu[1,2], Yun Feng[1], Xinyu Liu[1,2], Jianjun Zhao[1,2] and Qixu Liu[1,2*]

Abstract
Cyber attackers have constantly updated their attack techniques to evade antivirus software detection in recent years. One popular evasion method is to execute malicious code and perform malicious actions only in memory. Malicious programs that use this attack method are called memory-resident malware, with excellent evasion capability, and have posed huge threats to cyber security. Traditional static and dynamic methods are not effective in detecting memory-resident malware. In addition, existing memory forensics detection solutions perform unsatisfactorily in detection rate and depend on massive expert knowledge in memory analysis. This paper proposes MRm-DLDet, a state-of-the-art memory-resident malware detection framework, to overcome these drawbacks. MRm-DLDet first builds a virtual machine environment and captures memory dumps, then creatively processes the memory dumps into RGB images using a pre-processing technique that combines deduplication and ultra-high resolution image cropping, followed by our neural network MRmNet in MRm-DLDet to fully extract high-dimensional features from memory dump files and detect them. MRmNet receives the labeled sub-images of the cropped high-resolution RGB images as input of ResNet-18, which extracts the features of the sub-images. Then trains a network of gated recurrent units with an attention mechanism. Finally, it determines whether a program is memory-resident malware based on the detection results of each sub-image through a specially designed voting layer. We created a high-quality dataset consisting of 2,060 benign and memory-resident programs. In other words, the dataset contains 1,287,500 labeled sub-images cut from the MRm-DLDet transformed ultra-high resolution RGB images. We implement MRm-DLDet for Windows 10, and it performs better than the latest methods, with a detection accuracy of up to 98.34%. Moreover, we measured the effects of mimicry and adversarial attacks on MRm-DLDet, and the experimental results demonstrated the robustness of MRm-DLDet.

Keywords Memory-resident malware, Memory forensics, Malware detection, Deep learning, Ultra-high resolution image

*Correspondence:
Qixu Liu
liuqixu@iie.ac.cn
[1] Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100085, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100049, China

Introduction

Over the years, artificial intelligence (AI) techniques have significantly promoted the efficiency and ability of file-based malware detection engines. Yet, at the same time, cyber-attackers also keep exploring advanced methods to evade or compromise antivirus software. One such method is **I**n-memory **C**ode **E**xecution (ICE) (Fewer 2008; Team 2021; Paschen 2020; Malik 2019; odzhan 2019; Microsoft 2018; Kumar et al. 2020). ICE attacks only execute malicious operations in memory and leave almost no evidence on the disk, making it challenging for traditional static and dynamic analysis methods to detect (Arefi et al. 2018; Alrawi et al. 2021). For instance, in cyberattacks against the National Bank of Malawi (CERT 2018), attackers rewrote and recompiled the multiple open-source codes embedding the encrypted DarkComet (Lesueur 2020) remote access trojan (RAT) into relevant codes. In the actual operational process, the encrypted data will be loaded, decrypted, and expanded into a complete DarkComet RAT portable executable (PE) file in memory. In this way, the payload implements an ICE attack to avoid killing and bypassing anti-virus solutions. Advanced Persistent Threats (APT) groups and malware families such as Lazarus Group (MITRE 2021), Poweliks (O'Murchu and Gutierrez 2015), Zeus (Binsalleeh et al. 2010) all employed ICE attacks.

Kumar et al. (2020) briefly presents memory-resident malware. In this work, we define memory-resident malware as malware that executes shellcodes and PE files in memory using ICE technology.

With the spread of memory-resident malware, memory forensics has become more critical. Memory forensics (MF), is a technique that captures volatile memory data from computers' memory dumps and analyzes them. Memory dumps contain processes, network connections, open files, and registry modifications created during the malware's runtime, significant traces for identifying memory-resident malware. There have been many memory forensics studies incorporating machine learning (Barabosch et al. 2017; Bozkir et al. 2021; Wang et al. 2020; Sihwail et al. 2021). These works have significantly improved the memory-resident malware detection accuracy and efficiency. Unfortunately, variants of ICE attacks may inject different processes and use different methods to load shellcodes or PE files. Moreover, attackers always look for never-detected vulnerable processes or methods to construct advanced attacks. Thus, manual feature engineering requires analysts to be familiar with and possess extensive domain knowledge to distinguish high and low differentiation features.

Challenges

Based on the above discussions, existing memory-resident malware detection methods face two challenges: (1) The accuracy of detection frameworks relies on various hand-crafted features of memory-resident malware, which requires massive expert knowledge in the field of memory analysis and it is somewhat subjective and not generalizable. (2) Existing detection tools do not take full advantage of memory data information.

In the malware detection field, many studies have used computer vision to convert malware into images, then classify malware programs by specific image features (Nataraj et al. 2011; Ni et al. 2018; Bozkir et al. 2021; O'Shaughnessy and Sheridan 2022). These studies obtained great detection results. However, existing vision-based malware detection techniques usually analyze PE files directly. They still face the drawbacks of existing static and dynamic malware detection methods, i.e., they cannot effectively detect malware only running in memory. Moreover, a memory dump can be reshaped into an RGB image, but the size of a memory dump file is the same as the virtual machine's memory, which is 2GB or above. Therefore, the generated memory dump images are of ultra-high resolution, and their minimum size is about 6000×6000 pixels after our processing method (more details can be found in the "MRm-DLDet" section). However, existing vision-based malware detection methods only allow the input of regular-size pictures. For example, Ni et al. (2018) used images of max 32×32 pixels for their model; the detection model of Bozkir et al. (2021) extracts features from images converted from malware with 256×256 pixels; O'Shaughnessy and Sheridan (2022) proposed a hybrid malware classification model that extracts visual features from the images with a max size of 512×512 pixels, none of which can handle our ultra-high resolution images. Thus, it can be inferred that existing vision-based detection methods can not efficiently handle ultra-high resolution images.

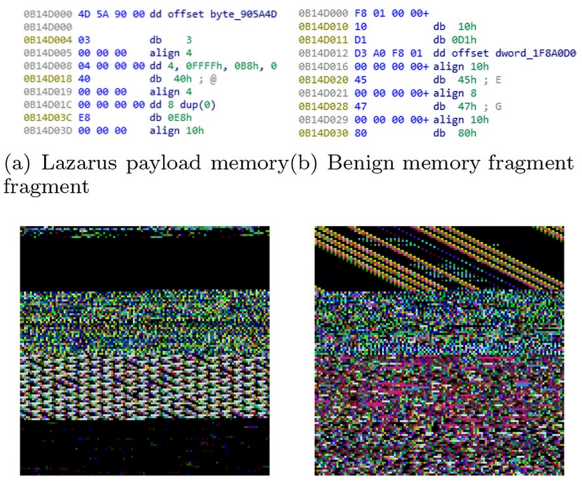
Motivation

Therefore, we proposed a novel approach by combining the information of the malware's whole memory dumps, such as memory pages, processes, and other related data with deep neural network for detection to solve the difficulties that traditional static and dynamic analysis methods to detect memory-resident malware. And solve the two challenges in memory-resident malware detection. Our work can better use the malware-specific execution data to detect memory-resident malware by converting memory dumps to pictures without extensive and complex expert knowledge. A memory dump file can be converted into an RGB image, every pixel of a memory dump image is associated with memory data, and the difference between images can help separate benign from malicious memory dumps. Moreover, this paper designs a memory dump file preprocessing method to relieve the storage space pressure caused by the size of memory dump files and solve the problem that existing vision-based malware detection methods cannot handle ultra-high resolution images.

In order to further discuss whether visualization can help detect memory-resident malware, we analyzed the memory dump of a Lazarus Group's sample. An MS-DOS header is found at address 0xB14D000 of this memory dump, which is the start of a PE file. Lazarus implements the ICE attack by decrypting the payload and loading it into its own memory space, so the PE file found at 0xB14D000 is the payload of the malware from Lazarus Group. We analyzed a benign sample for comparison, Fig. 1 shows a motivating example. The two images on top are data from the same location in the two memory dump files. We converted the two memory dumps to RGB images by our framework. The bottom of Fig. 1 shows a part of the two RGB images that correspond to the code fragments of two dumps at addresses from 0xB14D000 to 0xB1589DF. These images have significant differences between color, texture, and structure. That leads us to a driving thesis of our work: **The semantic and structural differences between malicious memory-resident actives and benign memory dumps can be effectively identified by visually comparing memory dump RGB images.** Detailed description of the memory dump files and its visualization can be found in the "MRm-DLDet" section.

Our work

We propose a state-of-the-art memory forensic framework based on deep learning called MRm-DLDet (Memory-Resident malware Deep Learning Detector). MRm-DLDet first captures memory dumps, and their size will be reduced by memory duplicate page deletion. Then MRm-DLDet converts memory dumps into ultra-high resolution RGB images and uses a non-overlapping sliding window to crop the images into sub-images served as inputs to MRm-DLDet's MRmNet neural network. The MRmNet combined with ResNet-18 (He et al. 2016), gated recurrent units (Cho et al. 2014), and attention mechanism (Zhou et al. 2016). Our framework



```
0B14D000 4D 5A 90 00 dd offset byte_905A4D
0B14D000
0B14D004 03          db    3
0B14D005 00 00 00    align 4
0B14D008 04 00 00 00 dd 4, 0FFFFh, 0B8h, 0
0B14D018 40          db  40h ; @
0B14D019 00 00 00    align 4
0B14D01C 00 00 00 00 dd 8 dup(0)
0B14D03C E8          db 0E8h
0B14D03D 00 00 00    align 10h
```

(a) Lazarus payload memory fragment

```
0B14D000 F8 01 00 00+
0B14D010 10           db  10h
0B14D011 D1           db 0D1h
0B14D012 D3 A0 F8 01  dd offset dword_1F8A0D0
0B14D016 00 00 00 00+ align 10h
0B14D020 45           db  45h ; E
0B14D021 00 00 00 00+ align 8
0B14D028 47           db  47h ; G
0B14D029 00 00 00 00+ align 10h
0B14D030 80           db  80h
```

(b) Benign memory fragment

(c) Lazarus image fragment (d) Benign image fragment

Fig. 1 A motivating example of memory dump visualization

overcomes the two challenges faced by existing memory-resident malware detection methods and solves the problem that current image detection methods cannot handle ultra-high resolution images. Experiments show that MRm-DLDet has a high detection accuracy (98.34%).

To summarize, in this paper, we make the following contributions:

- We comb through the latest ICE methods from malware families and APT groups and first define malware that uses ICE methods to execute shellcode or malicious PE files in memory as memory-resident malware.
- We propose the first memory-resident malware detection framework that combines memory forensics and deep learning named MRm-DLDet, which focuses on capturing and analyzing memory dumps. MRm-DLDet has a virtual machine environment for capturing memory dumps, a novel memory dump preprocessing method combining data deduplication and ultra-high resolution image cropping, and a neural network named MRmNet.
- Because of the lack of publicly available open source datasets for in-memory-resident malware detection, we collected a dataset with 2,060 benign and malicious programs. The memory dumps of the programs that are converted into ultra-high resolution images will be cropped into 1,287,500 sub-images. Now our dataset is ready to be used publicly for non-commercial reasons.
- We studied the influence of different image sizes on the performance of MRm-DLDet and compared MRm-DLDet with the most advanced methods. Compared to the latest methods, our framework is better in all experimental evaluation metrics. Specifically, MRm-DLDet has a detection rate of up to 98.34%.

Related work

Memory forensics-based memory-resident malware detection

Since Malik (2019) first performed how to load and run portable executables entirely from memory manually, in-memory malware execution has gradually become a prevalent attack method for cyber attackers. In this paper, we divide MF-based malicious code detection methods into two types based on different technical bases: method based on the characteristics of operating system and memory pages (*OS Characteristics Based*), and detection method combining artificial intelligence

(*AI Based*). Figure 2 succinctly shows the latest MF-based malware detection techniques.

OS characteristics-based

Volatility (Foundation 2020) is the most widely used and authoritative open-source MF-based framework with a plugin called malfind. It determines whether the process is suspicious by checking the memory pages' virtual address descriptor (VAD), a process' VAD tree that describes the layout of memory segments. In the VAD tree, there is also some information about the type and level of protection (read, write, execute) of the memory page (Ligh et al. 2014), in addition to information related to the mapped object and several other flags. For example, if the protection field of a memory page is set to "PAGE_EXECUTE_READWRITE", then malfind will initially determine that it is a malicious process. However, malfind can be easily bypassed by malware developers. For instance, the "Bypass Malfind" method proposed by Block and Dewald (2019) will assign a memory of one memory-resident malware with READONLY protection and then change the protection state of all contained pages to EXECUTE_READWRITE by VirtualProtectEx. What's more, malfind needs an expert in the memory forensics field, as malfind does not provide a post-processing algorithm to distinguish benign software from malicious ones, which means it requires extensive expert knowledge of memory forensics to analyze Volatility's output and determine if a program is a malicious one.

Arefi et al. (2018) have reported a reverse engineering tool named FAROS to detect in-memory-only malware injection attacks. FAROS only focused on three in-memory code injection attack techniques. And only implemented on Windows 7 VM, without considering new attack methods on Windows 10 systems, which are now more widely used.

A very recent effort by Alrawi et al. (2021) presented a post-detection technique named FORECAST to predict capabilities that malware has staged for execution automatically. FORECAST guides a symbolic analysis of the malware's code by leveraging the execution context of the ongoing attack from the malware's memory image.

AI-based

Wang et al. (2020) proposed PROVDETECTOR that detects malware with steganography, which is a provenance-based approach. PROVDETECTOR first uses a novel selection algorithm to identify potentially malicious parts of the process' OS-level provenance data. Then it applies neural embedding and machine learning. In another study, Quincy (Barabosch et al. 2017) extracted 38 features from the volatile memory and used Random Forests and Extremely Randomized Trees to classify the memory storage area, which achieved an AUC score of 93.8% on Windows XP, but only 84.4% on Windows 10.

Bozkir et al. (2021) represented the suspicious processes' memory dumps into RGB images and reported 96.39% prediction accuracy by combining the RBF kernel-based SMO algorithm with GIST+HOG for feature vectors. Still, This study only investigated malware processes and did not consider malware that hides in benign processes to execute, such as UUID Shellcode (Team 2021) and Earlybird (spotheplanet 2020). We assume that this limits the accuracy of the study by Bozkir et al. (2021) to some extent.

Sihwail et al. (2021) applied memory forensics to extract memory-based features from malware memory images to expose the actual behavior of malware. They used feature engineering and the SVM algorithm, converted the features into binary vectors, and obtained a classification accuracy of 98.5% in Windows 7 OS.

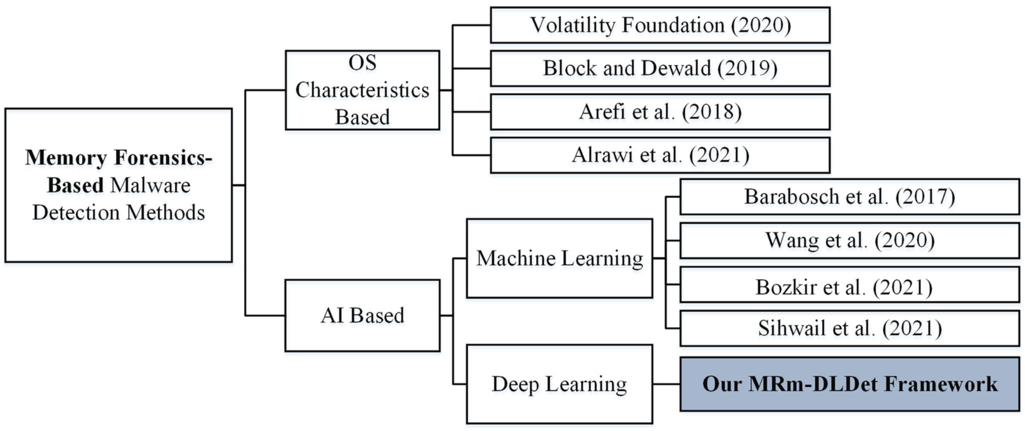


Fig. 2 Classification of related work on memory forensics-based malware detection

However, this task is time-consuming for manual feature extraction and performs poorly on Windows 10.

Deep learning-based malware detection vision methods

In recent years, computer vision has been applied to malware detection with good results. The idea of converting files into images before detection inspired our research.

Nataraj et al. (2011) first visualized Malware binaries as grayscale images based on the observation that the images belonging to the same malware family appear very similar in layout and texture. In their solution, Bozkir et al. (2019) employed several various convolutional neural networks to classify persistent malware files. They converted PE files' binary bytes into images, and they reported 97.48% detection accuracy in experiments.

Pinhero et al. (2021) used three malware visualization methods: grayscale maps, RGB maps, and Markov images, and then extracted features of the three types of images using Gabor filters. Twelve different neural networks were trained and the F-measure up to 99.97%.

Tekerek and Yapici (2022) proposed a new method based on CNN by converting byte files to gray and RGB image formats respectively for malicious code classification. O'Shaughnessy and Sheridan (2022) proposed a hybrid framework for malware classification by setting an entropy threshold to quickly determine whether a sample is packed or not, then analyzing the samples using static and dynamic methods respectively. Static PE files or memory dump files of processes are mapped into images by space-filling curves, then the model extracts visual features from the images, reporting an accuracy of 97.6%.

However, most of the existing vision-based malware detection methods directly analyze the binary files. O'Shaughnessy and Sheridan (2022) converted memory dumps to images when the malicious program is running, but they do not analyze the complete memory data. All existing vision-based malware detection efforts cannot deal with ultra-high resolution images. Our work solves this problem by deduplicating complete memory dumps and using non-overlapping sliding windows to cut images into multiple sub-images.

Framework overview

We first discuss the threat model, then introduce the overall framework of MRm-DLDet. Finally, we describe the background of ultra-high resolution image classification (one of the essential techniques used in this study) in detail.

Threat model

MRm-DLDet is a framework for memory-resident malware detection in Windows 10. It takes PE files as input, converts the memory dumps of PE programs runtime to RGB images, and then uses deep neural network to detect memory-resident malware. In this paper, memory-resident malware is primarily used as attack payloads to launch attacks on the target system to directly execute the malware in the victim computer's memory, instead of writing malware to the hard drive to evade the progressively increasing malware detection process and remain invisible in the target device.

We assume that the memory-resident malware executes the attack when we capture memory dumps. Recent research (Wang et al. 2020) can help alleviate this assumption. Moreover, it is assumed that all ICE attacks leave traces of in-memory data.

Our framework

MRm-DLDet is a memory-resident malware detection tool that integrates computer vision and deep learning techniques with memory forensics to model ICE attacks. Figure 3 gives an overview of the MRm-DLDet architecture, which consists of three main parts. We outline the approach here, and the following two sections provide complete information.

First, MRm-DLDet captures memory dump files and removes duplicate memory pages (**A** in Fig. 3). Next, we convert deduplicated memory dumps to RGB images. The RGB images are ultra-high resolution since the memory dumps still contain too much data after removing duplicates. Inspired by ultra-high resolution image processing methods in remote sensing image recognition, we propose a vision-based enormous image processing solution. To avoid the important information loss caused by traditional multiple downsampling layers image scaling methods, we cropped the enormous ones into sub-images (**B** in Fig. 3). After that, sub-images are fed into the MRmNet. MRmNet extracts the feature vector of each sub-image by a pre-trained ResNet-18 network. The feature vectors of sub-images formed the feature of the whole memory dump file, which are fed into the gated recurrent units (GRU) model later. Then, we add an attention layer to retain important details and prevent information loss. Finally, we design a voting layer to output the memory-resident malware detection results (**C** in Fig. 3).

Background on ultra-high resolution image classification

Our MRm-DLDet framework first visualizes binary files into RGB images. Then we want to get the features of these images (i.e., features of the memory dump files) by the ResNet-18 network. However, the dumps converted after deduplication still have a minimum size of 6000×6000. Limited by the storage of GPUs in general devices at this stage, it is not possible to handle the computation of ultra-high resolution images, and

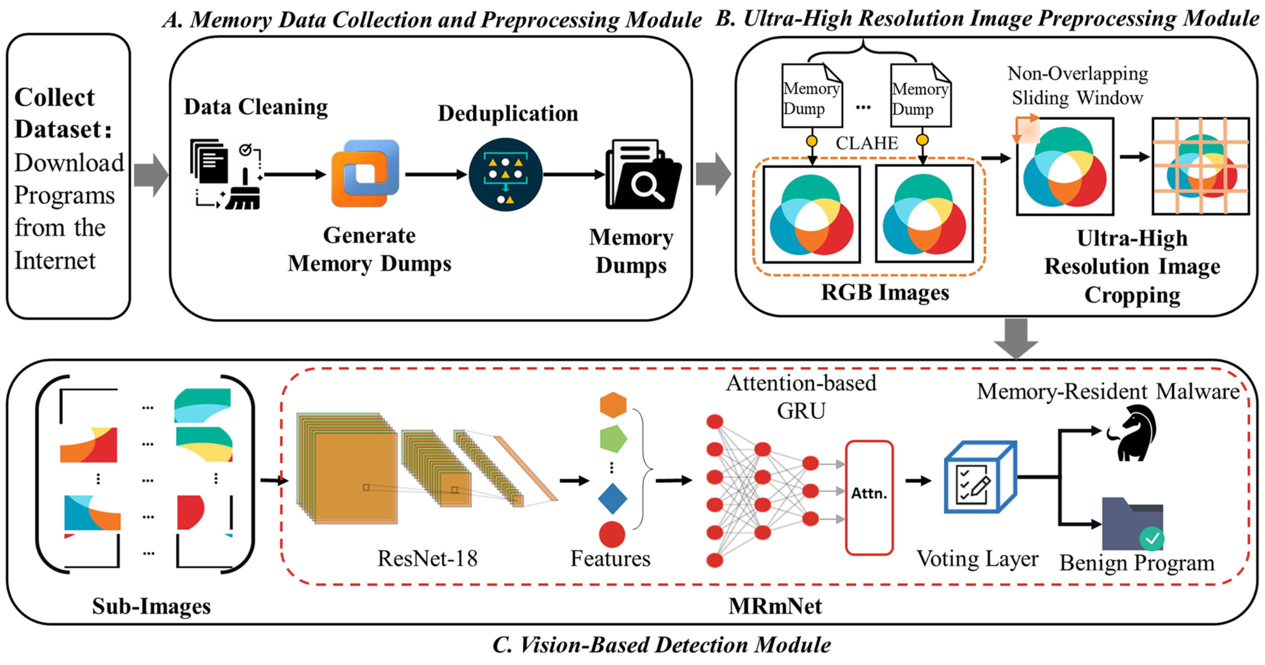

A. Memory Data Collection and Preprocessing Module
B. Ultra-High Resolution Image Preprocessing Module
Collect Dataset: Download Programs from the Internet
Data Cleaning
Deduplication
Generate Memory Dumps
Memory Dumps
Memory Dump
Memory Dump
CLAHE
Non-Overlapping Sliding Window
RGB Images
Ultra-High Resolution Image Cropping
Sub-Images
Attention-based GRU
Memory-Resident Malware
ResNet-18
Features
Attn.
Voting Layer
Benign Program
MRmNet
C. Vision-Based Detection Module

Fig. 3 The overview of MRm-DLDet framework

processing such large images directly by CNN networks will lead to memory overflow. Thus, it is considered necessary to preprocess ultra-high resolution images to reduce the image size and retain complete memory information contained in the image. The direct resize method is the easiest and fastest, but this will cause a substantial loss of features, resulting in poor detection accuracy.

In this work, we find inspiration from some methods in remote sensing images field. Van Etten (2018) proposed YOLT to detect small objects from large swaths of imagery. YOLT first uses a sliding window in which sizes and overlaps (15% by default) are defined by users to partition ultra-high resolution images into cutouts, then puts them into a network architecture to train and test the model. The F1-score of YOLT is higher than 0.8. In the study proposed by Wang et al. (2019), they cropped remote sensing images into several small sub-images by large-scale cropping, utilizing the non-overlapping sliding window method, to ensure that the big pictures are not scaled during training and testing.

In the MRm-DLDet framework, we use a non-overlapping sliding window technique to cut memory dump images, which can preserve as much information as possible in the memory dumps compared to resizing images directly to the target size.

MRm-DLDet

To solve the challenges presented in the "Introduction" section, we created a memory-resident malware detection framework. This section shows the MRm-DLDet framework and its three component modules in detail.

Memory data collection and preprocessing module

This module executes malicious samples and benign programs in the processed virtual machines and generates a memory snapshot file for each program. This module also does memory dump deduplication as the first memory dump preprocessing step. Specifically, the following three modules are included.

Modify virtual machine

We use virtualization-based software VMware Workstation (VMware 2022), to generate memory dumps for memory-resident malware and benign samples. Before getting the memory dump file, we first made some changes to the virtual machine settings since malware is likely to be sensitive to its operating environment. For example, according to cybersecurity experts from G DATA (Ebach 2017), malware of Zeus family verifies if it is being launched on a VMware system by checking whether `\\.\HGFS` file, `\\.\vmci` file, or registry key: `HKLM \SOFTWARE\VMware Inc. \VMware Tools` exists. If any of them is present, Zeus aborts execution and removes itself. Screen resolution is a commonly used anti-virtual machine detection indicator by malware as well. For example, the banking Trojan TrickBot (Abrams 2020) checks the target device's screen resolution to detect virtual machines. If the screen resolution is 800 × 600 or 1024 × 768, the machine will be considered a virtual machine. In addition, malware will search for user activity on the device by whether the mouse is moved or clicked, or the keyboard is typed, etc., to determine if it is being analyzed within a virtual machine (Miramirkhani et al. 2017; Yokoyama et al. 2016; Bulazel and Yener 2017). According to Malware Behavior Catalog (2022), samples from the DarkComet family will check if the mouse is moving. The Darkhotel and Ursnif malware (MITRE 2021; Ionut Arghire 2017), check whether the mouse cursor position has changed to determine whether it is running on a real device. Therefore, we mitigate these anti-VM detections by performing actual user actions while the malicious sample is running, including moving and clicking the mouse and typing characters on the keyboard.

Therefore, we modified the configuration of our Windows 10 virtual machine by the following steps to prevent it from being checked by memory-resident malware.

- Uninstall VMware Tools.
- Do some modifications to the.vmx file, such as making the virtual machine use the same BIOS serial number as the physical machine, etc.
- Modify the MAC address to a random one except default VMware MAC address (e.g., 00:0c:29, 00:50:56, 00:05:69).
- Modify screen resolution to any value except 800 × 600 and 1024 × 768. In our virtual machine, we set it to 1152 × 864.
- Mimic normal user behavior by clicking or moving the mouse and tapping random characters on the keyboard when running samples in the virtual machine.

Generating memory dumps

To generate memory dumps, firstly, we do data cleaning to remove samples that use outdated in-memory code execution methods and those that don't run in our Windows 10 VM. Then we automate the memory dump generation process by controlling the vmrun utility through python scripts to improve generation efficiency. Vmrun utility is a command-line utility that controls virtual machines to perform various tasks, such as power on/off, and create a new snapshot. Finally, we use vmrun's 'createSnap' operation to capture a snapshot that dumps the VM's memory state to a file. We created a snapshot of the Windows 10 VM before any operations and named it 'Initial State'. Every time before running one sample, the VM machine will roll back to the 'Initial State'. Küchler et al. (2021) suggests that most malicious behavior can be observed within the first two minutes that it is executed. Each malicious sample was given two minutes to initialize and execute.

To further analyze the memory dump file, we present the structure of the captured memory dump files. Windows memory management can be summarized into three mechanisms: (1) virtual address space management, (2) physical page management, and (3) address translation and page swapping (Yu et al. 2015). MRmDLDet analyzes the entire data of the memory dump, including the data of the physical and virtual memory space, where each process runs in its own virtual address space. In Fig. 4, we briefly show the layout of a typical process (Yosifovich et al. 2017), with each part of it described as follows.

- **Kernel address space:** Users do not have access to this part of the memory, which is managed by the operating system and used for paging pools, system cache, device drivers, etc.
- **User address space:** Programs running in user mode have no access to the kernel address space but are

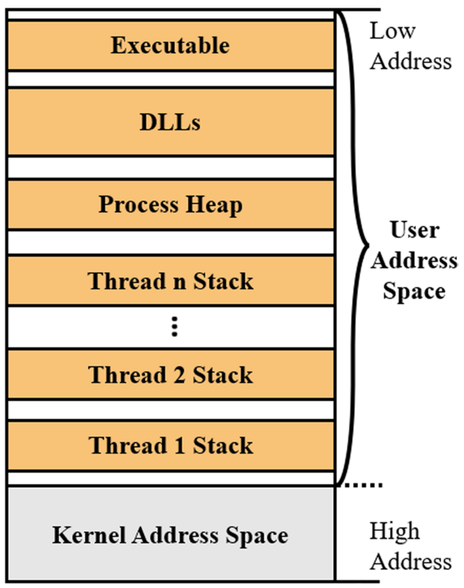


Fig. 4 Windows Memory Layout overview of one typical process

allowed to enter the user address space to which they are assigned.

- **Thread stack:** It is used to display the memory allocation for the stack used by each thread in this process and orderly allocate short-term storage for local variables.
- **Process heap:** It is the dynamically allocated memory portion, shows the memory allocation for this process heap, and is used by programs to store global variables.
- **DLLs:** Contains the DLL files that the sampling process needs to call.
- **Program image:** Placement of the executable files.

Figure 4 illustrates a high-level layout since current Windows operating systems use techniques such as address space layout randomization (ASLR) to defend against attacks such as buffer overflow, which means that several parts of Fig. 4 may not be contiguous in a complete memory dump file. Further exploration of these techniques is out of scope of this paper. Furthermore, the “Experiments” section shows that MRm-DLDet obtains excellent detection results without turning off these techniques.

Memory dump deduplication

The generated memory dumps are saved as snapshot files. As the fact that the memory size of the virtual machine and one memory dump is the same, which is above 2GB. To reduce the consumption of storage space and improve MRm-DLDet’s efficiency when analyzing memory dumps, we are inspired by Brengel and Rossow (2018) to design and implement a memory duplicate page delete process, which is the first step of memory dump preprocessing. We base on two observations: (1) The memory-resident malware only injects payload in small areas of memory compared to the whole memory. (2) Rolling back to the ’Initial State’ to start running guarantees that the memory is always the same each time we run a sample. Therefore, we only retained the memory data related to ICE attacks to improve the specificity of memory dumps. We define memory dump deduplication as: Analyzes the target memory dump and deletes its pages that are the same as the ’Initial State’ memory dump.

After memory dump deduplication, the memory dump files from the previous module only retain the memory data about changes after running samples, such as new processes and threads, added registry configurations, injected shellcodes, etc. Algorithm 1 shows the deduplication process briefly.

Algorithm 1 MEMORY_DUMP_DEDUPLICATION

```
Input: The path of intital, target and deduplication memory dumps
    are: $Init\_dump\_path$, $Tgt\_dump\_path$, $Dedup\_dump\_path$;
    $pagesize$ is the size of one page, $page(i = 1, 2, 3 \ldots n)$ is the
    data of one page, $path$ is the path of a memory dump, $f$ is a
    file to be read or write.
Output: $Dedup\_dump$ is the deduplication memory dump file.
 1: function GET_PAGES($path$, $pagesize \leftarrow 4096$)
 2:     $f \leftarrow open(path$, read)
 3:     while $true$ do
 4:         $page_i \leftarrow f.read(pagesize)$
 5:     end while
 6:     YIELD ($page_i$)
 7: end function

 8: Initialize a list $Init\_pages \leftarrow []$
 9: Initialize a list $Dedup\_pages \leftarrow []$
10: for $i$, $page_i$ in $enumerate$(GET_PAGES($Init\_dump\_path$)) do
11:     $Init\_pages.append(page_i)$
12: end for
13: for $i$, $page_i$ in $enumerate$(GET_PAGES($Tgt\_dump\_path$)) do
14:     if $hash(page_i) = hash(Init\_pages[i])$ then
15:         $Dedup\_pages.append(page_i)$
16:     end if
17:     $Dedup\_dump \leftarrow open(Dedup\_dump\_path$, write)
18:     $Dedup\_dump.write(Dedup\_pages)$
19: end for
20: $Dedup\_dump.close()$
```

Ultra-high resolution image preprocessing module

In this module, deduplicated memory dump files are converted into RGB images. We cut one ultra-high resolution RGB image into sub-images and labeled each sub-image. It is the second memory dump preprocessing step.

Visualization memory dumps

In order to overcome the challenges faced by current memory-resident malware detection method that relies on massive expert knowledge and does not fully exploit memory information. We are inspired by the motivating example in Fig. 1. Moreover, from “Generating Memory Dumps” section, we find that the data between different parts of a process memory also correspond to different uses and structures. We infer that the injected memory regions will be distinguished from the benign memory dump structure. Therefore, we represented the memory dump files as RGB images. The difference between the different data contents after visualization is used to distinguish benign programs from memory-resident ones.

The deduplicated memory dump files from the previous module can essentially be represented as binary strings consisting of zeros and ones. We convert every 8 bits (1 byte) of a memory dump file to a pixel value ($0x00 \rightarrow 0, 0xFF \rightarrow 255$). First, read three-pixel values from one memory dump file at a time, and fill them into

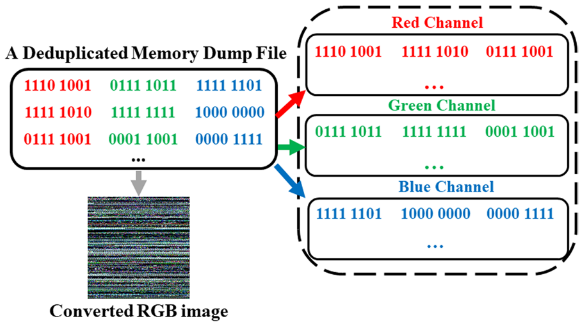


Fig. 5 Convert a memory dump file to an RGB image

a 3D array. Hence, the three dimensions are respectively loaded into the R, G, B channels to generate an RGB image. What's more, to increase the variability in visual features between various memory dump images and to improve detection accuracy, we use the CLAHE (Reza 2004) technology on all memory dump images. Figure 5 depicts a process of developing a memory dump image. The images we generate are all square, and the final size of the image is determined by the number of pixels in the image, which is square root of the number that represents (*dumpfilesize*)/3. At the same time, we choose a lossless PNG format for the images to minimize the loss of features. The resolution of the images ranged from 6000 × 6000 pixels to 10000 × 10000 pixels.

Figure 6 shows six memory dump images of benign and memory-resident samples, with three images of benign memory dumps on the left and three images of memory-resident samples on the right. An empirical observation that can be made is that the benign and malicious memory dump images are visually distinct. For color, the benign image is lighter, while the memory-resident image is darker. For texture, benign and malicious images have different textures. This observation is also consistent with our inference above. Therefore, it can be found that the RGB images generated by memory dumps of memory-resident malware are significantly different from those generated by benign programs, i.e., the method of using deep neural networks to classify images is efficacious for memory-resident malware detection.

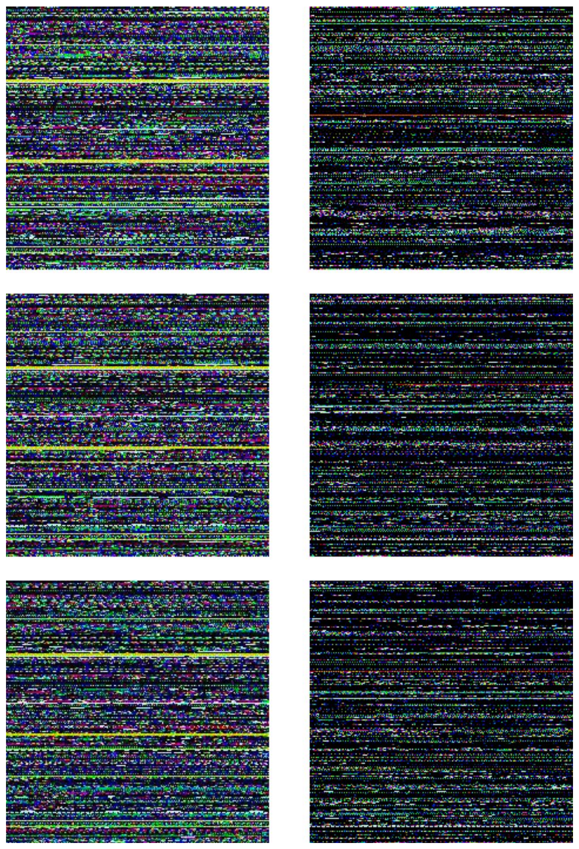


Fig. 6 Several RGB memory dump images belonging to benign samples and memory-resident malware

Ultra-high resolution image processing

The RGB images converted from memory dump files are ultra-high resolution images, and CNN networks commonly used for image classification usually do not support such large-scale inputs. As mentioned in "Background on Ultra-High Resolution Image Classification" subsection, in previous studies (Wang et al. 2019; Van Etten 2018), researchers have proposed sliding window methods for ultra-high resolution images, which are mainly used for target detection tasks of remote sensing images and do not aim at image classification tasks.

Therefore, we first use the non-overlapping sliding window method in vision-based malware detection. Our approach overcomes the drawback of traditional image scaling methods, which use multiple downsampling layers and lead to information loss.

We set the size of each sub-image to 224 × 224. However, the size of memory dumps varies, and so does the number of sub-images. To solve this problem, MRm-DLDet resizes all RGB images to the same size using bicubic interpolation. In this resizing method, the textural features are still visible (Vasan et al. 2020a). In addition, to choose an appropriate size, we investigate the influence of three image sizes on model detection results and choose 5600 × 5600 as the adjusted size of RGB images. The memory dump images are then cut into multiple sub-images, and each memory dump produces 625 sub-images after cropping. "Experiments" section describes more details on the influences of image size on model detection accuracy.

Vision-based features extraction and detection module

This module mainly includes the MRmNet, the neural network in MRm-DLDet. "Structure of MRmNet" section shows more details of this module.

Images features extraction

We use a pre-trained ResNet-18 network to extract a vector with a length of 512 for each sub-image that was cut from one memory dump file. The features of each memory dump file are represented as a matrix of [625, 512].

Memory-resident malware detection

We train the attention-based gated recurrent units (GRU) network with the features of the sub-images and divide the training set, verification set, and test set. Consider performance evaluation metrics to adjust hyperparameters, including accuracy, precision, recall and F1-score. In the end, the model adds a voting layer, which we first generated. After the neural network outputs the prediction results of each sub-picture, the voting layer first calculates the sum of each 625 outputs, i.e., each sub-picture votes for the final classification of memory dump, with a value of 1 or 0. The voting layer then calculates the average value of sub-images voting. If the average value is above the threshold, the sample that the memory dump represents is detected as a memory-resident malware.

Structure of MRmNet

We designed and implemented MRmNet consisting of ResNet-18, an attention-based GRU, and a self-made voting layer in the Vision-Based Detection Module of the MRm-DLDet framework. It extracts high-dimensional features from memory dump images to improve the accuracy of our detection framework.

More recently, methods combining convolutional neural networks and recurrent neural networks have been widely used, the CNN models extract spatial features in depth and retain valid information, followed by training and prediction with RNN, which allows the model to better express temporal and spatial features. This paper applies this approach to the malware detection problem, using more advanced ResNet-18 and GRU networks to improve the CNN-RNN combination. The MRmNet structure in MRm-DLDet framework is shown in Fig. 7.

Input layer

In MRm-DLDet Framework, malicious and benign programs to be detected are first extracted from the runtime memory dumps. Then the memory dumps are converted into RGB images and cropped into multiple sub-images. The sub-images are then imported into the ResNet-18 layer.

In this study, D_i $(i = 1, 2, 3 \ldots n)$ denotes the memory dump files generated by memory-resident samples and benign samples, i denotes that the sample is the i-th of our dataset. While $d_{i,j}$ $(i = 1, 2, \ldots, n, j = 1, 2, \ldots, 625)$

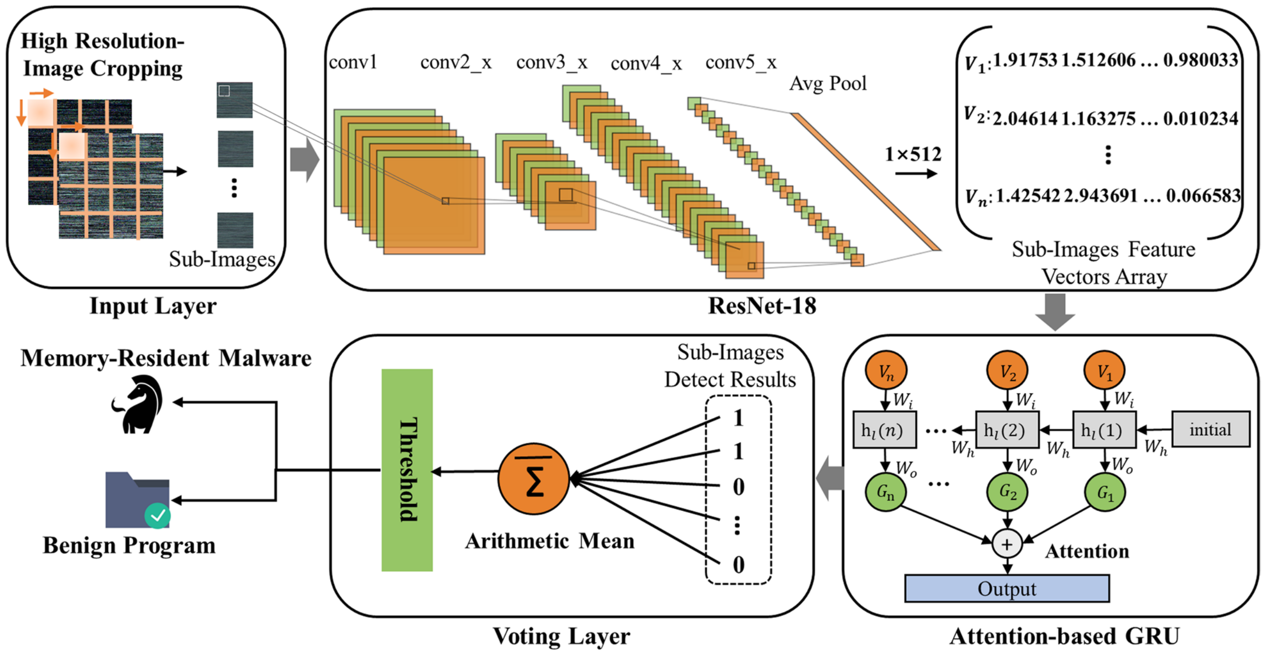


Fig. 7 The MRmNet structure in MRm-DLDet framework

denotes each sub-image generated after sliding window partitioning a D_i image, j denotes the serial number of the sub-picture, 625 in total.

ResNet-18 layer

The second layer is the network to extract the $d_{i,j}$ features. In this study, we use the ResNet-18 network (He et al. 2016). To get over the difficulty that deep networks are not easily optimized, the ResNet-18 network uses a residual structure. Each residual block is a multilayer neural network consisting of a convolutional layer, a batch normalization layer, and an activation layer. The new technique introduced by the ResNet model provides shortcut connections between non-contiguous convolutional layers. This technique allows the model to skip layers to process vanishing gradients to achieve lower losses and better results.

To obtain the image features, we extract the output of the ResNet-18 model's avgpool layer as the result of one sub-image feature extraction, which is a vector with a length of 512. In detail, the cropped sub-images, represented as d$_{i,j}$ that obtained from the input layer, enter the ResNet-18 layer as the input. After going through the pre-trained ResNet network, it extracts a [1, 512] vector $V_{i,j}$ $(i = 1, 2, ..., n, j = 1, 2, ...625)$, which is generated by avgpool layer. i denotes that the sample is the i-th program of our dataset, and j denotes the serial number of the sub-image. The output of the ResNet-18 layer is formalized as:

$$V_{i,j} = ResNet - 18(d_{i,j}) \tag{1}$$

GRU layer

The third layer of MRmNet is the GRU layer. It is well known that long short-term memory (LSTM) (Hochreiter and Schmidhuber 1997) solves RNN's problem of lacking long-term dependence on learning by adding a gated mechanism and memory cell. However, the LSTM network has many parameters and converges slowly. GRU (Cho et al. 2014) is an improved version of standard LSTM. GRU makes simplifications and improvements on LSTM networks. GRU only has update gate and reset gate, while LSTM has three gates (forget gate, input gate, and output gate). GRU has fewer training parameters, so it saves much time when the training data is enormous.

This layer divides the feature-extracted sub-image vectors from the previous layer into the training set, validation set, and test set. Then, for example, the memory-resident malware in the training set can be obtained as the vector sequence shown in Eq.(2). $Train_c(c = 0, 1)$ represents the class of samples in the training set, i.e., 0,1.

$$Train_c = \left\{V_{1,1}, V_{1,2}, ...V_{2,1}, V_{2,2}, ..., V_{n,625}\right\} \tag{2}$$

To further describe the GRU principle, set V_t to represent the input at the current moment, z_t as the update gate, r_t as the reset gate, h_t as the hidden state that passes to the next moment, while h_{t-1} is the old state, $\tilde{h}_t$ is the candidate hidden state. The specific implementation of a single gated recurrent unit is as follows:

$$\begin{aligned} z_t &= \sigma(W_z \cdot [h_{t-1}, V_t]) \\ r_t &= \sigma(W_r \cdot [h_{t-1}, V_t]) \\ h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t \\ \tilde{h}_t &= tanh(W \cdot [r_t * h_{t-1}, V_t]_j) \end{aligned} \tag{3}$$

In addition, we applied dropout technique on our GRU layer to reduce the risk of overfitting. Overall, the output by GRU layer at moment t is represented as:

$$G_t = GRU\left(V_{i,j}\right) \tag{4}$$

Attention layer

The attention mechanism was proposed by Zhou et al. (2016), which could assign weights to data and weight summation, and is highly interpretable. The attention mechanism effectively retains important details and prevents critical information from being lost. For that reason, we added an attention layer after the GRU layer and let it assign weights to the output vectors so that it could further improve the detection accuracy of the MRm-DLDet framework. In our neural network, the attention mechanism estimates the association level between features. G_t is the vector output by the GRU layer at moment t. W is set as the result of the GRU layer weighted summation of the output vectors. Let a_t represent the weight of the hidden layer of the attention module, b represent the bias.

$$W = a_t G_t + b \tag{5}$$

Set $L_{i,j}$ as the final output label.

$$L_{i,j} = softmax(a_t W) \tag{6}$$

In the end, the output $L_{i,j}$ is the classification result of sub-images. The output of MRmNet has two cases: on the one hand, the source binary of the sub-image is a memory-resident malware, i.e., $L_{i,j}$ is set to 1. On the other hand, the program which generated the sub-image is a benign file, i.e., $L_{i,j}$ is set to 0.

Voting layer

The final layer of MRmNet is voting layer, a layer that we designed. In previous layers, ResNet-18 will go through

GRU, and then the attention layer will assign weights and give detection results.

The detection result $L_{i,j}$ from the attention layer is only the classification result of one sub-image. In MRm-DLDet, a memory dump image will be cropped into 625 sub-images. Therefore, a combined result of the 625 sub-images will report the whole memory dump image's classification result, which is the tested sample's detection result.

In order to effectively detect memory-resident malware, it becomes a challenge as how to most effectively organize the classification results of the 625 sub-images from each memory dump image. We designed a voting layer, the attention layer's output $L_{i,j}$ as input. Every 625 sub-images represent a memory dump file, that can be considered as a group, for example, when $i = 1$, $Group_1 = \left\{L_{(1,1)}, L_{(1,2)}, ...L_{(1,625)}\right\}$. Calculate the arithmetic mean of each sub-images group, the result is shown as M_i.

$$M_i = \overline{\sum_{i=0}^{n}\sum_{j=0}^{625} L_{i,j}} \tag{7}$$

That is, each sub-picture is considered to vote for the classification of the final executable, with a value of 1 as memory-resident malware, or 0 as benign. The arithmetic mean is averaged over those 625 sub-pictures, with a threshold of 0.6 for classification, "Experiments" section describes more details on the selection basis of the threshold values.

- $M_i < 0.6$, detecting as benign program.
- $M_i \geq 0.6$, detecting as memory-resident malware.

Experiments

This section introduces our basic experimental setup, discusses existing malware datasets, and presents our dataset. After that, we described each of our experiments in detail and showed the experiments' results.

Dataset

Many research institutions and companies have provided malware datasets that can be used for artificial intelligence and big data analysis, such as the 2015 Microsoft Malware Classification Challenge dataset (BIG2015 dataset) (Ronen et al. 2018), EMBER dataset (Anderson and Roth 2018), and SOREL-20M dataset (Harang and Rudd 2020).

However, these datasets have some drawbacks. On the one head, in order to prevent the spread of malware, they all provide processed malicious sample data. For example, the BIG2015 dataset only provides processed byte files and asm files. EMBER only includes features extracted after parsing the PE file, and SOREL-20M offers malicious samples with the PE header set to 0. Since memory-based detection methods require memory data when a program runs, these samples are not available for memory forensics. On the other hand, these datasets provide few benign executables. Therefore, the existing datasets do not apply to our study. To construct a suitable dataset, we constructed a dataset that meets the following requirements:

- Malicious samples from memory-resident malware family.
- Both malicious samples and benign programs are complete PE files and can be run on Windows 10.
- All the samples were built recently.

Malware samples

We collected memory-resident malware that use up-to-date evasion methods from VirusShare, which is a malware sample repository that provides security researchers and forensic analysts access to real-time malware samples. To cover as many existing ICE attack techniques as possible, we selected more than 80 malware families using ICE attack techniques such as process injection based on security companies' publicly available technical analysis and the analysis in the globally-accessible knowledge base of adversary tactics and techniques ATT &CK matrix (MITRE).

Benign samples

The benign binaries consist of system files and some freeware with a large number of users. Some benign data is extracted from the "System32" directory of the Windws 10 system. In addition, we collected some free popular programs from CNET Download (Ventures 2022). To ensure that the benign programs are not bundled with malicious or adware, we verified the binary labels by uploading benign samples to VirusTotal. VirusTotal aggregates a large number of antivirus products and online scanning engines to detect malware. We removed the samples obtained from CNET Download that VirusTotal detection ratio >3%.

We collected 1120 benign and 1648 malicious samples. After data cleaning, we eventually obtained 1010 benign and 1050 malicious binaries. Since every memory dump image is a set of 625 sub-images, there are 631,250 benign sub-images and 656,250 malicious sub-images as the input for MRmNet. Once all the data are processed, we divide the train, validation, and test set according to the ratio of $6:2:2$. Table 1 describes the detailed division

Table 1 Data distribution of benign and malicious samples

Class	Train set	Val set	Test set	Total
Memory-resident	630	210	210	1,050
Benign	606	202	202	1,010
Total	1,236	412	412	2,060

Table 2 Data distribution of sub-images

Class	Train set	Val set	Test set	Total
Memory-resident	393,750	131,250	131,250	656,250
Benign	378,750	126,250	126,250	631,250
Total	772,500	257,500	257,500	1,287,500

Table 3 The hyperparameters during training

Configuration	Value
Epoch	60
Batch Size	2048
Learning Rate	0.001
ModelCheckpoint	monitor='val_acc', mode='max'

of benign and malicious data, and the division of processed sub-images for MRmNet's model training can be found in Table 2. Moreover, our datasets are now ready to be used for non-profitable purposes (C1air3 2023).

Experiment settings

We implemented our execution environment on one ThinkPad T480 physical machine with Intel Core i5-8250U, 1.80 GHz processor, and 24 GB of RAM. We generated memory dump files in VMware Workstation (VMware 2022), version 16.0, which installed a Windows 10 OS. The main programming language environment is Python 3.7.

After several evaluations and adjustments, the hyperparameter configuration of the attention based GRU model in the experiment shows in Table 3. In the experiments of this study, we chose Accuracy, Precision, Recall, and F1-score to evaluate the effectiveness of our MRm-DLDet framework and the neural network models used for comparison. These four evaluation metrics have been widely used in previous studies, and they are important basis for model performance evaluation.

Threshold for detection

In MRmNet, the final voting layer needs to choose a threshold to classify the voting results into memory-resident malware and benign. Selecting a reasonable threshold would help our model achieve the best detection results. We used the median value 0.5 as guideline with a step size of 0.03 and tested the four evaluation metrics of the model when the threshold values were respectively chosen from 0.45 to 0.69, which is presented in Fig. 8.

According to Fig. 8, when the threshold value is selected as 0.6, we got the best accuracy, recall, and F1-score scores. Thereby, it can be seen that 0.6 is an appropriate threshold for our detection framework.

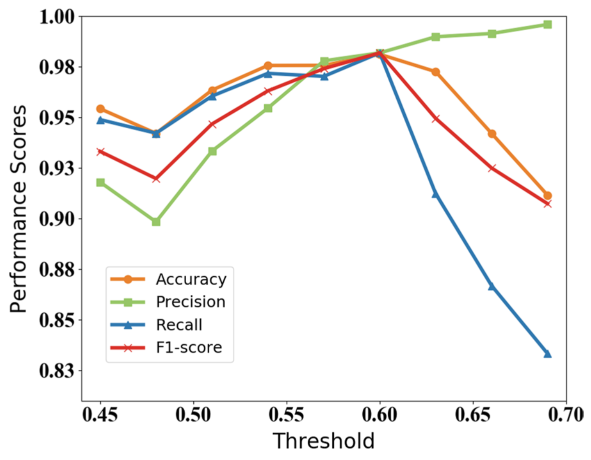


Fig. 8 Detection results of different thresholds

Table 4 Comparison of different image processing methods

Method	Accuracy (%)	Precision	Recall	F1-score
Directly resize	90.55	0.8824	0.9002	0.8912
Non-overlapping sliding window	**98.34**	**0.9896**	**0.9777**	**0.9836**

Bold values indicate the best detection result

Ultra-high resolution image preprocessing method evaluation

The first experiment explored the effect of different ultra-high resolution image processing methods on the MRm-DLDet framework's detection performance. On the one hand, the ultra-high resolution images were directly reduced to 224×224 by the bicubic interpolation method. On the other hand, a non-overlapping sliding window cuts every ultra-high resolution image into 625 sub-images. Then separately fed, the memory dump images using these two methods into the MRmNet (ResNet-18+GRU+Attention) network for training and testing. The evaluation metrics' results of the two experiments are in Table 4. Compared with the direct scaling

Table 5 Description of three CNN models

Model	Author	Description
VGG16	Simonyan and Zisserman (2014)	The VGG16 network has a strong fitting ability. It is often used as a benchmark for malware identification, and its core design idea is to use smaller convolutional kernels and build deeper network layers.
Inception V3	Vasan et al. (2020b)	Proposed by Google, the highlight is the addition of decomposition techniques to decompose the convolutional kernel.
ResNet-18	He et al. (2016)	ResNet-18 is one of the ResNet network family, its network structure balances training efficiency and accuracy well, and it has achieved excellent results in visual malware classification.

Table 6 Comparison of the MRm-DLDet framework evaluation using different neural networks

Method	Model	Accuracy (%)	Precision	Recall	F1-score
Directly resize	VGG16	87.76	0.8842	0.8733	0.8787
	ResNet-18	91.59	0.9124	0.9062	0.9093
	Inception V3	85.79	0.8649	0.8813	0.8730
	MRm-DLDet	90.55	0.8824	0.9002	0.8912
Non-overlapping sliding window	VGG16	92.36	0.9192	0.9210	0.9201
	ResNet-18	94.20	0.9276	0.9524	0.9398
	Inception V3	91.77	0.9193	0.9128	0.9160
	MRm-DLDet	**98.34**	**0.9896**	**0.9777**	**0.9836**

Bold values indicate the best detection result

method, using a non-overlapping sliding window could significantly improve the memory-resident malware detection performance of MRm-DLDet.

Furthermore, to compare the effect of different neural networks with different image processing methods on the performance of visual detection of memory-resident malware, we selected three neural network models that are widely used in malware visualization detection methods as comparison baselines. Then trained and tested the models using the processed memory dump data. The models are briefly described in Table 5.

Table 6 shows the results of training and testing the four neural network models separately with images processed by two different dimensionality reduction methods. It can be found that MRm-DLDet using sub-images cropped by non-overlapping sliding windows as input has the best detection accuracy of 98.34% and F1 score > 0.98, and the detection results of the models using images processed with the non-overlapping sliding window method as input are better than those of the models trained with direct resized images. This also proves that it is reasonable to apply non-sliding windows with ultra-high resolution memory images, which can preserve the features of memory dumps better than directly resized images to get higher detection accuracy.

We analyzed the misreported programs. A malicious sample of the DarkComet family was misreported as benign, and we found that this was because when the memory dump of the sample was obtained, the runtime sample did not successfully connect to C&C and therefore did not perform the following attack behavior, resulting in similar characteristics to the benign program. Another observation is that memory-resident malware samples are falsely reported at a higher rate, probably because hackers are constantly improving ICE attacks to make their actions increasingly slight and more similar to the APIs used by benign programs, for example, to obtain a higher evade capability.

Different memory dump image sizes and neural networks evaluation

In MRm-DLDet's Ultra-High Resolution Image Preprocessing Module, when using the non-overlapping sliding window to crop memory dump RGB images, the number of sub-images varies depending on the resolution of the memory dumps. MRm-DLDet solves this problem by resizing all RGB images to the same size by bicubic interpolation. We evaluate the effect of three different image sizes: 3360×3360, 4480×4480, and 5600×5600 on the detection performance of the model. Additionally, to choose the best CNN-RNN combination for MRm-Net, we selected three CNN models (VGG16, Inception V3, ResNet-18), three RNN models (RNN, LSTM, GRU) and respectively cross-combined them. Each CNN-RNN combination uses three different sizes of memory dump images as input, the CNN models extract image features, and the features are then transferred to the RNN models that are combined with the attention mechanism.

Totally 27 detection models were generated, 9 of each memory dump image size. Each sub-image is 224×224, through the non-overlapping sliding window, one RGB image of three sizes produces 625, 400 as well as 225 sub-images respectively. Table 7 shows the training and detection results of each model. Besides the four

Table 7 Comparison of different memory dump image sizes and different neural networks

Memory dump image size	MRmNet's deep learning model	Accuracy	Precision	Recall	F1-score	Feature extraction time (minutes)
5600 × 5600(625 Sub-Images)	VGG16 + RNN + Attention	96.49%	0.9474	0.9671	0.9572	0.5970
	VGG16 + LSTM + Attention	97.51%	0.9554	0.9676	0.9614	0.5970
	VGG16+GRU+Attention	97.62%	0.9546	0.9680	0.9613	0.5970
	Inception V3 + RNN + Attention	94.27%	0.9328	0.9351	0.9340	3.9532
	Inception V3 + LSTM + Attention	94.90%	0.9450	0.9526	0.9488	3.9532
	Inception V3 + GRU + Attention	95.22%	0.9309	0.9674	0.9488	3.9532
	ResNet-18 + RNN + Attention	97.66%	0.9817	0.9642	0.9729	0.4779
	ResNet-18 + LSTM + Attention	98.11%	0.9828	0.9810	0.9819	0.4779
	ResNet-18 + GRU + Attention	**98.34**%	**0.9896**	**0.9777**	**0.9836**	0.4779
4480 × 4480(400 Sub-Images)	VGG16 + RNN + Attention	95.67%	0.9529	0.9524	0.9527	0.5837
	VGG16 + LSTM + Attention	95.97%	0.9500	0.9575	0.9537	0.5837
	VGG16 + GRU + Attention	96.05%	0.9581	0.9534	0.9557	0.5837
	Inception V3 + RNN + Attention	93.28%	0.9251	0.9388	0.9319	1.1931
	Inception V3 + LSTM + Attention	94.01%	0.9468	0.9418	0.9442	1.1931
	Inception V3 + GRU + Attention	93.78%	0.9321	0.9375	0.9348	1.1931
	ResNet-18 + RNN + Attention	97.81%	0.9808	0.9787	0.9797	0.4598
	ResNet-18 + LSTM + Attention	97.66%	0.9821	0.9733	0.9777	0.4598
	ResNet-18 + GRU + Attention	97.41%	0.9808	0.9714	0.9761	0.4598
3360 × 3360(225 Sub-Images)	VGG16 + RNN + Attention	95.28%	0.9427	0.9531	0.9479	**0.2674**
	VGG16 + LSTM + Attention	94.81%	0.9345	0.9463	0.9403	**0.2674**
	VGG16 + GRU + Attention	95.59%	0.9464	0.9644	0.9553	**0.2674**
	Inception V3 + RNN + Attention	94.98%	0.9316	0.9536	0.9425	1.4769
	Inception V3 + LSTM + Attention	94.61%	0.9439	0.9404	0.9422	1.4769
	Inception V3 + GRU + Attention	95.02%	0.9405	0.9562	0.9483	1.4769
	ResNet-18 + RNN + Attention	95.58%	0.9807	0.9432	0.9616	0.3964
	ResNet-18 + LSTM + Attention	96.00%	0.9794	0.9528	0.9659	0.3964
	ResNet-18 + GRU + Attention	96.03%	0.9784	0.9643	0.9713	0.3964

Bold values indicate the best experimental results

evaluation metrics, we also consider the feature extraction time, which shows the time taken by different CNN models to extract features from a memory dump image of different sizes.

Table 7 shows that in terms of image size, the accuracy of the neural networks decreases with the increased compression of the images. The evaluation metrics of the model gradually decrease from 5600 × 5600 to 3360 × 3360. For example, the ResNet-18+GRU+Attention model has 98.34% accuracy, 0.9896 precision, 0.9777 recall, and F1-score is 0.9836 when the image size is 5600 × 5600, while the accuracy drops to 97.41% and 0.9808 when the image size is 4480 × 4480. When the size is reduced to 3360 × 3360, the detection accuracy is only 96.03%, the precision is only 0.9784. The same finding was found in the other 9 combinations of neural network models. This may be because using a 5600 × 5600 image, more sub-images can be generated by non-overlapping sliding windows and the model is relatively getting better detection results. Therefore, we choose 5600 × 5600 for MRm-DLDet as the size of the memory dump image after bicubic interpolation processing in order to get the best detection results.

For the CNN models, five evaluation metrics are considered: accuracy, precision, recall, F1-score, and feature extraction time. In MRm-DLDet, the pre-trained CNN model extracts features for each memory dump file's sub-images, which are then transferred to the RNN model to train and test. The feature extraction time is affected by two factors:

- The number of sub-images of each memory dump file is different due to the different sizes of the images.
- The difference between the structure of the neural network results in different feature vector lengths extracted by each model.

In this experiment, the image feature vector extracted from the ResNet-18 model's avgpool layer is 512 in length, the VGG16 model extracts features of 4096 in length for each sub-image, and the pre-trained Inception V3 model outputs sub-image features with 1 dimension and 2048 in length. Therefore, for the 27 detection models, the combination of different memory dump image sizes and pre-trained CNN models resulted in 9 different feature matrices, and the corresponding feature extraction times are shown in Table 7 as well. Since the feature extraction time only depends on the CNN model, every three detection models using the same CNN share the same feature extraction time.

Table 7 shows that among the nine models using the same memory dump image size, ResNet-18 with different RNN models combination obtained better detection results than the other CNN models. Taking a 5600 × 5600 size image as an example, the highest accuracy of the three models using the pre-trained ResNet-18 extracted features was 98.34%, and the results of precision, recall, and F1-score were also the best. The best model that combines the VGG16 model with each of the three RNN models is VGG16+GRU+Attention, with a detection accuracy of 97.62%. The detection accuracy of the combination of Inception V3 and three RNNs is lower than that of the model using the other two methods to extract features, with the highest detection accuracy of only 95.22%. The same conclusion can be found in other memory dump image sizes.

Comparing the feature extraction time consumption of the three CNN models in Table 7, Inception V3 takes much more time than the other two, which could be more efficient and suitable for further deployment. The difference between VGG16 and ResNet-18 in feature extraction efficiency is less than 0.1 min, which is not significantly different. The shortest feature extraction time is found when the image size is 3360 × 3360, and the pre-training model is VGG16, which takes only 0.2674 min to extract the features of one sample. However, ResNet-18 is still selected as the CNN model for the vision-based detection module because the detection accuracy of ResNet-18 models is much better than that of the VGG16 models, and the required feature extraction time is 0.4779 min. There is no significant difference in detection efficiency compared to VGG16.

Once the memory dump image size and CNN model are determined, it can be found that the best performing RNN model in the MRmNet is GRU by analyzing the detection results of the 27 deep learning models in Table 7. Comparing the results with different combinations of the three RNN models using the same image size and CNN models, the combination of CNN with GRU

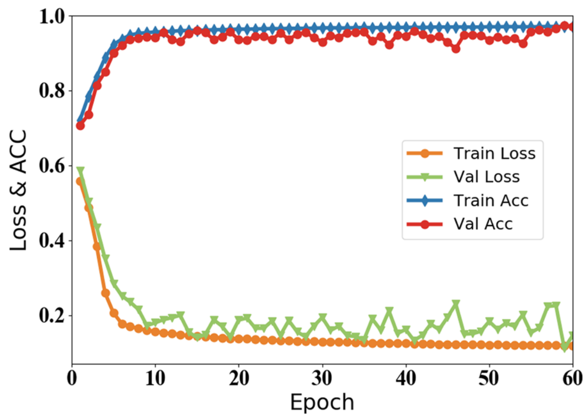


Fig. 9 Accuray and loss of each epoch

and attention achieved the best detection results out of 25 models.

In conclusion, the ResNet-18+GRU+Attention model with memory dump images at the size of 5600 × 5600 achieved the best detection result, with an accuracy of 98.34%. Figure 9 shows the variation of both accuracy as well as loss of the model on the train and validation sets. We noticed that this model attains high accuracy with both train and validation sets, and the training loss and validation loss are very close to each other, which means our model works well in memory-resident malicious code detection experiments.

Ablation study

In the previous subsection, we evaluated the effect of using different CNN-RNN combinations and different image sizes on the final detection results of MRm-DLDet. We selected the image size as 5600 × 5600 and the model combination ResNet-18+GRU+Attention for MRmNet that resulted in the best detection results. To further explore the impact of different components in the MRmNet of MRm-DLDet on the final detection performance, we performed an ablation study. Three MRmNet combinations: only ResNet-18, ResNet-18+GRU, and ResNet-18+GRU+Attention, were considered separately to evaluate the effectiveness of each component. The results can be found in Table 8.

According to Table 8, when using only the ResNet-18 model, MRmNet has the same results as those in Table 6. The lowest detection accuracy is achieved by converting the size of ultra-high resolution images to 224 × 224 and inputting it directly into the ResNet-18 model for detection only in this model. The detection accuracy of MRmNet was significantly improved to 95.43% when the ultra-high resolution images were cropped into sub-images and then using ResNet-18 to extract sub-image

Table 8 Ablation Study for MRm-Net

Model used by MRmNet	Accuracy	Precision	Recall	F1-score
ResNet-18	91.59%	0.9124	0.9062	0.9093
ResNet-18 + GRU	95.43%	0.9633	0.9504	0.9568
ResNet-18 + GRU + Attention	**98.34%**	**0.9896**	**0.9777**	**0.9836**

Entries in bold font indicate the best results

features before calling the GRU network for detection. Finally, the best detection results were obtained after adding the attention mechanism to the GRU network. Thus, MRmNet using ResNet-18+GRU+Attention as in MRm-DLDet gets the best results.

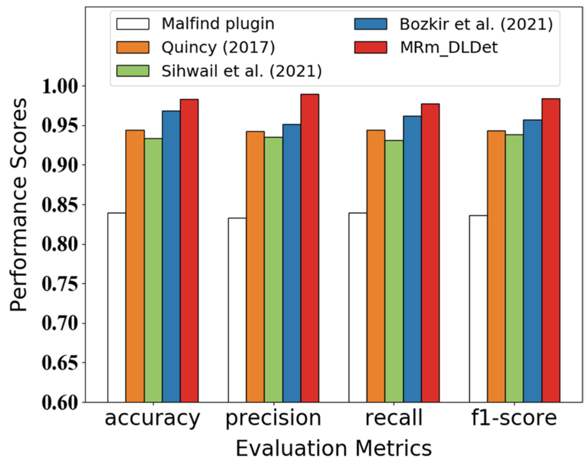


Fig. 10 Comparing MRm-DLDet with Memory Forensics-based detection methods

Comparing existing techniques

To comprehensively evaluate the effectiveness of our MRm-DLDet framework, as MRm-DLDet focuses on capturing and analyzing memory dumps, we compared the MRm-DLDet with several up-to-date memory forensics-based memory-resident malware detection works. In addition, since our work detects memory-resident malware code using a vision-based approach, we also compare it with several latest existing vision-based malware detection methods. The methods used for both comparisons have been described in "Related work" and are only briefly described in this subsection.

Comparing memory forensics-based detection methods

We compared MRm-DLDet with four memory forensics-based malware detection works. Malfind is a plugin of Volatility, which detects malware by checking if the VAD protection is set to PAGE_EXECUTE_READWRITE. Quincy (Barabosch et al. 2017) extracted 38 features from a memory dump and used Random Forests and Extremely Randomized Trees to classify the memory storage area. Sihwail et al. (2021) converted the features extracted from malware memory into binary vectors, further using the SVM algorithm to detect malware. Bozkir et al. (2021) is the most advanced study for detecting memory-resident malware using memory forensics. It represents memory dumps as RGB images and uses an SMO algorithm with radial basis kernels combined with feature vectors extracted by GIST+HOG.

We reproduced the detection methods according to the principles described in the article and trained and tested them by our dataset. The experimental results can be found in Fig. 10.

The MRm-DLDet framework gets better results than other methods in accuracy, precision, recall, and F1-score according to Fig. 10. The dataset used in this experiment was our memory-resident malware dataset contains state-of-the-art evade methods. Malfind treats every non-empty memory area with RWX permissions as malicious, and once those permissions are well-tuned (i.e., only RX permissions are set), malfind performs poorly. The other methods were trained and tested mainly on Windows 7, and the features they selected may not take full advantage of memory data information, resulting in poor performance in our dataset.

Comparing deep learning and vision-based malware detection methods

To evaluate MRm-DLDet as comprehensively as possible, we compared it with four recent malware detection methods combining deep learning and vision-based methods. Bozkir et al. (2019) converted raw binary bytes of PE files into RGB images and used convolutional neural networks to classify persistent malware files. Pinhero et al. (2021) visualized PE files into grayscale, RGB, and Markov images and extracted features of the three types of images by Gabor filters. VGG3 got the best classification result in their work. Tekerek and Yapici (2022) converting PE files to gray and RGB images for malware classification. O'Shaughnessy and Sheridan (2022) used a hybrid framework for malware classification, first setting an entropy threshold to determine whether a sample is packed or not, then analyzing the samples using static and dynamic methods respectively. Their dynamic methods converted malicious process memory dumps into space-filling curve images.

Again, we reproduced the methods according to the principles in the article, then trained and tested them by our dataset. The results are shown in Fig. 11. Due to image layout limitations, Tekerek and Yapici (2022) is represented as "Tekerek (2022)", and O'Shaughnessy

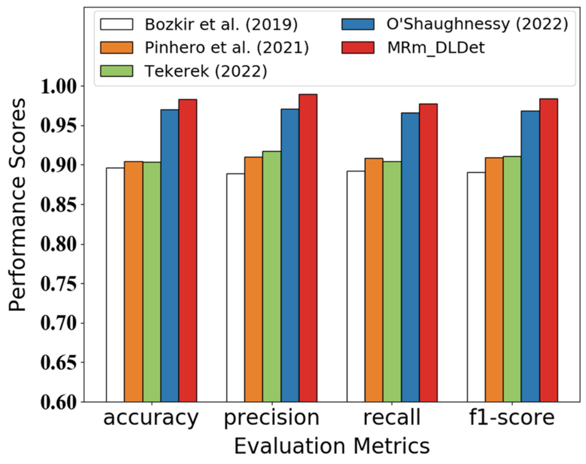


Fig. 11 Comparing MRm-DLDet with Vision-Based detection methods

and Sheridan (2022) is represented as "O'Shaughnessy (2022)".

Analyzing the results in Fig. 11, the methods of Bozkir et al. (2019), Pinhero et al. (2021) and Tekerek and Yapici (2022) perform similarly, but all have significant differences with the detection results of the other two methods. This may be because the above three methods visualize PE files directly, and the packed samples affect the accuracy. O'Shaughnessy and Sheridan (2022)'s method uses static and dynamic analysis methods by distinguishing between packed and unpacked samples. Its dynamic approach can capture malware activities in the memory, thus showing high accuracy in our dataset. However, its dynamic method only extracts the mini-processes containing process threads and handles information of malware, which means they do not analyze the complete memory data, resulting in lower detection results than our MRm-DLDet.

The two comparing experiment results prove that our framework that converts memory-resident malware's memory dumps to RGB images and uses deep learning models to detect them is an effective ICE attacks detector with high accuracy and recall.

Time consumption evaluation in realistic environment

We investigated the performance of MRm-DLDet working in a realistic environment and evaluated the time consumption of deploying it in a real environment. We deployed our framework on a Windows 10 computer and kept providing suspicious PE files into MRm-DLDet for analysis to detect if they were memory-resident malicious programs. Statistically, the average time taken by MRm-DLDet from receiving a PE file to the end of detection is 2.39 min. This is expected as a PE file needs to be put into the machine and run for about two minutes to initialize and execute. It looks like MRm-DLDet is somewhat time-consuming. However, from previous experimental results, we found that the MRm-DLDet framework is far ahead of the existing memory-resident malware detection methods regarding accuracy. Therefore, we consider the time consumption of the memory dumps preprocessing stage to be acceptable.

Robustness of MRm-DLDet framework

To measure the robustness of MRm-DLDet, we evaluate the impact of evasion attacks on our model. Neural network-based malware detection methods are easily attacked by mimicry attacks (Wagner and Soto 2002) and adversarial attacks. In this section, we analyze both types of attacks and evaluate the performance of our MRm-DLDet framework against these evasion attacks.

Mimicry attacks to MRm-DLDet framework

To evaluate MRm-DLDet's robustness to mimicry attack, we described four kinds of up-to-date memory-resident evasion methods in Table 9, they all have good evasion capability.

We compared our MRm-DLDet detection framework with the seven most popular AV engines (PCmag 2022). We generate 30 mimicry attack samples for each state-of-the-art memory-resident evasion method in Table 9 and detect them separately with the eight AV engines. All mimicry attack samples do not appear in the training dataset of MRm-DLDet. The detection results can be found in Table 10.

Table 10 shows that MRm-DLDet gets better detection results than popular AV engines, and the only ICE attack which is not detected is module stomping. We define the criterion of robustness as the ability to detect three out of four state-of-the-art attacks proving that the detection engines are robust. It can be found that only MRm-DLDet and Microsoft's Windows Defender are robust against the latest mimicry attacks. Overall, this periment confirmed that our MRm-DLDet is robust to the latest mimicry attacks.

Adversarial attacks to MRm-DLDet framework

Over the past few years, many studies of adversarial attacks against deep learning-based malware detection tools have proved effective (Suciu et al. 2019; Hu and Tan 2017; Anderson et al. 2018; Grosse et al. 2017). In theory, MRm-DLDet is a deep learning-based detection framework that can also be attacked by an adversarial example carefully constructed by the attackers. We assume that the attackers will not intercept our model, so the attackers can only modify the attack sample based on the model binary decision, i.e., black box attack.

Table 9 The latest evasion methods used by memory-resident malware

Evasion Method	Description
UUID Shellcode (Team 2021)	UUidFromStrinA API takes a string based UUID and converts it to its binary representation.
	Providing UUidFromStrinA a pointer to a heap address, it decodes data and writes them to memory without using common functions such as memcpy or WriteProcessMemory.
	The EnumSystemLocales function executs shellcode.
Earlybird (spotheplanet 2020)	Earlybird method creates a new legitimate process in a suspended state and allocates memory for shellcode in the new process's memory space.
	Declare APC routine pointing to the shellcode, then shellcode is written to the previously allocated memory.
	Queuing APC to the main thread, resuming the thread and executing the shellcode.
Phantom DLL Hollowing (orr 2021)	Phantom DLL Hollowing first open a TxF handle to a Microsoft signed DLL file on disk, and infect its .text section with shellcode.
	Generate a phantom section from this malware-implanted image and map a view of it to the address space of a process of his choice.
	The shellcode is hidden and then executed in the .text section with +RX permissions.
Module Stomping (ired.team 2020)	Module Stomping first injects some benign Windows DLL into a remote (target) process.
	Overwrites DLL's that loaded in step one, AddressOfEntryPoint point with shellcode.
	Starts a new thread in the target process at the benign DLL's entry point, where the shellcode has been written to.

Table 10 Comparison of the detection results for the latest ICE attacks by different antivirus solutions

AV engines	Attacks			
	UUID Shellcode	Earlybird	PhantomDLL hollowing	Module stomping
McAfee	U [*]	U	U	M
Bitdefender	M [**]	U	U	M
Webroot	U	U	U	U
Malwarebytes	U	U	U	U
ESET-NOD32	U	U	U	M
Sophos	U	U	U	U
Microsoft	M	M	U	M
MRm-DLDet	M	M	M	U

[*] U means Undetcted

[**] M means Memory-Resident Malware

We believe that attacks against deep neural networks are not feasible in our environment. On the one hand, our method takes PE files as input. It is deployed on the client system, so the attackers cannot theoretically obtain the feature data during detection for data poisoning attacks, etc. On the other hand, existing black-box attack methods (Hu and Tan 2017; Anderson et al. 2018) are aimed at static PE anti-malware, such as modifying malware by adding irrelevant junk characters to bypass detection. Since most of these methods are intended to alter malware without affecting the program's regular function, malware's operations in memory will not change. So the existing adversarial attacks that modify the PE files have minimal impact on our detection framework.

Discussion

Our experimental results indicate that MRm-DLDet framework is superior to the state-of-the-art memory-resident malware detection methods. Measurements of MRm-DLDet's runtime have shown that it could also be used in real-world malware detection. We detect ICE attacks with memory forensics, which takes the memory dumps of program runtime as information sources. MRm-DLDet takes full advantage of the feature that memory forensics can directly analyze RAM data and detect malware operations in memory for capturing memory-resident malware. Some challenges are still faced in memory-resident malware detection.

The risk of overfitting

Our dataset consists of 1010 benign and 1050 memory-resident samples, which seems insufficient for a deep learning model, and the model may be at risk of overfitting. In this paper, the risk of overfitting is concentrated in the neural network MRmNet of MRm-DLDet, where ResNet-18 is primarily used for extracting image features. The attention-based GRU network is used to complete the classification task. The high-resolution images input into the GRU network is split into multiple sub-images after the ultra-high resolution image preprocessing module, samples for training and testing GRU network are 631,250 benign sub-images and 656,250 malicious sub-images, so the sample size is enough to avoid overfitting. We have also added dropout and early stopping techniques that effectively generalize the network and reduce the overfitting of trained data to the GRU network.

Runtime time overhead

Getting a memory dump when performing malicious operations usually takes more than two minutes for execution time. It will reduce the efficiency of the detection system. This situation challenges memory forensics and deep learning-based malware detection techniques, which we identify as future work to be addressed.

Apply the MRm-DLDet framework in a production environment

To run MRm-DLDet in the production environment, it will be deployed to the client device and detect suspicious PE files according to the user's requirements, get the memory dumps and detect them, and return the detection results to the client in time so that the malicious files can be handled in time to ensure the safety of the client device. On the one hand, future research will need to monitor malicious sample analysis reports published by major security companies and forums to obtain updated samples. On the other hand, there is also a demand to investigate a way to reduce the training and testing time for periodic model updates in the future, with an aim to find a solution that doesn't lead to a lag in model detection but also achieves high efficiency.

Conclusion

This study has applied deep neural networks to detect memory-resident malware in memory forensics. We proposed and evaluated a detection framework for memory-resident malware named MRm-DLDet. It combines the information of the malware's whole memory dumps with neural networks to overcome the bottleneck faced by existing detection methods that rely on massive expert knowledge and do not fully exploit memory information. We Deduplicate the large memory dump file first. Then in the process of visualizing memory dumps, we converted memory dumps to RGB images. We used a non-overlapping sliding window to cut the generated high-resolution images to maximize the preservation of memory dump features. This aims to improve deep neural networks' accuracy and determine whether there are suspicious processes in the computer's memory. To support these findings, we collected a publicly available dataset consisting of state-of-the-art memory-resident malware and benign samples. We have designed and implemented MRmNet for detection in MRm-DLDet, which is composed of ResNet-18, GRU, and attention, and it was trained by our dataset.

We explored the performance of different detection methods. In our experiments, MRm-DLDet achieved an impressive accuracy rate and detected 98.34% of memory-resident malware. This method offers a reasonable time to detect memory-resident malware. Moreover, MRm-DLDet exhibits well robustness in detecting the latest memory-resident malware, demonstrating that it is potentially valuable in practical usage. As a result, MRm-DLDet is a powerful detection scheme for memory-resident malware.

Acknowledgements
The authors of the paper sincerely appreciate anonymous reviewers who reviewed this manuscript and provided constructive comments.

Author Contributions
JL participated in all the work, proposed the framework with careful experiments, and wrote the manuscript. YF and XL provided suggestions on the detection model and joined the discussion of this work. JZ did some work on data collection. QL reviewed the manuscript and gave suggestions on the revision of the details of the article. All authors read and approved the final manuscript.

Funding
This work is supported by the Youth Innovation Promotion Association CAS (No.2019163), the Strategic Priority Research Program of Chinese Academy of Sciences (No. XDC02040100), the Key Laboratory of Network Assessment Technology at Chinese Academy of Sciences and Beijing Key Laboratory of Network security and Protection Technology.

Availability of data and materials
The full dataset used in this paper and the demo code for the detection framework are publicly available and could be accessed at:https://github.com/C1air3/MRm-DLDet.

Declarations

Competing interests
The authors declare that they have no competing interests.

Received: 16 January 2023 Accepted: 6 April 2023
Published online: 03 August 2023

References

Abrams L (2020) TrickBot malware now checks screen resolution to evade analysis. https://www.bleepingcomputer.com/news/security/trickbot-malware-now-checks-screen-resolution-to-evade-analysis/

Alrawi O, Ike M, Pruett M, Kasturi RP, Barua S, Hirani T, Hill B, Saltaformaggio B (2021) Forecasting malware capabilities from cyber attack memory images. In: 30th USENIX security symposium (USENIX security 21), pp 3523–3540

Anderson HS, Roth P (2018) Ember: an open dataset for training static pe malware machine learning models. arXiv preprint arXiv:1804.04637

Anderson HS, Kharkar A, Filar B, Evans D, Roth P (2018) Learning to evade static pe machine learning malware models via reinforcement learning. arXiv preprint arXiv:1801.08917

Arefi MN, Alexander G, Rokham H, Chen A, Faloutsos M, Wei X, Oliveira DS, Crandall JR (2018) Faros: illuminating in-memory injection attacks via provenance-based whole-system dynamic information flow tracking. In: 2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp 231–242. IEEE

Barabosch T, Bergmann N, Dombeck A, Padilla E (2017) Quincy: Detecting host-based code injection attacks in memory dumps. In: international conference on detection of intrusions and malware, and vulnerability assessment, pp 209–229. Springer

Binsalleeh H, Ormerod T, Boukhtouta A, Sinha P, Youssef A, Debbabi M, Wang L (2010) On the analysis of the zeus botnet crimeware toolkit. In: 2010 eighth international conference on privacy, security and trust, pp 31–38. IEEE

Block F, Dewald A (2019) Windows memory forensics: detecting (un) intentionally hidden injected code by examining page table entries. Digit Investig 29:3–12
Bozkir AS, Tahillioglu E, Aydos M, Kara I (2021) Catch them alive: a malware detection approach through memory forensics, manifold learning and computer vision. Comput Sec 103:102166
Bozkir AS, Cankaya AO, Aydos M (2019) Utilization and comparision of convolutional neural networks in malware recognition. In: 2019 27th signal processing and communications applications conference (SIU), pp 1–4. IEEE
Brengel M, Rossow C (2018) Memscrimper: Time-and space-efficient storage of malware sandbox memory dumps. In: international conference on detection of intrusions and malware, and vulnerability assessment, pp 24–45. Springer
Bulazel A, Yener B (2017) a survey on automated dynamic malware analysis evasion and counter-evasion: Pc, mobile, and web. In: proceedings of the 1st reversing and offensive-oriented trends symposium, pp. 1–21
C1air3: MRm-DLDet (2023). https://github.com/C1air3/MRm-DLDet
CERT A (2018) Analysis of cyberattacks against the national bank of Malawi. https://www.antiy.com/response/20181127.html
Cho K, Van&nbsp;Merriënboer B, Bahdanau D, Bengio Y (2014) On the properties of neural machine translation: encoder-decoder approaches. arXiv preprint arXiv:1409.1259
Ebach L (2017) Analysis Results of Zeus. Variant Panda G DATA, G DATA
Fewer S (2008) Reflective DLL injection
Foundation V (2020) The volatility framework. http://www.volatilityfoundation.org
Grosse K, Papernot N, Manoharan P, Backes M, McDaniel P (2017) Adversarial examples for malware detection. In: European symposium on research in computer security, pp 62–79. Springer
Harang R, Rudd EM (2020) Sorel-20m: A large scale benchmark dataset for malicious pe detection. arXiv preprint arXiv:2012.07634
He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778
Hu W, Tan Y (2017) Generating adversarial malware examples for black-box attacks based on gan. arXiv preprint arXiv:1702.05983
Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780
Ionut Arghire: Ursnif banking Trojan gets mouse-based anti-sandboxing (2017). https://www.securityweek.com/ursnif-banking-trojan-gets-mouse-based-anti-sandboxing/
ired.team: Module stomping for shellcode injection (2020). https://www.ired.team/offensive-security/code-injection-process-injection/modulestomping-dll-hollowing-shellcode-injection
Küchler A, Mantovani A, Han Y, Bilge L, Balzarotti D (2021) Does every second count? time-based evolution of malware behavior in sandboxes. In: proceedings of the network and distributed system security symposium, NDSS. The Internet Society
Kumar S et al (2020) An emerging threat fileless malware: a survey and research challenges. Cybersecurity 3(1):1–12
Lesueur J-P (2020) Darkcomet: remote administration tool. https://www.darkcomet-rat.com/
Ligh MH, Case A, Levy J, Walters A (2014) The art of memory forensics: detecting malware and threats in windows, Linux, and Mac memory. John Wiley, USA
Malik A (2019) In-memory execution of an executable. https://securityxploded.com/memory-execution-of-executable.php
Malware Behavior&nbsp;Catalog: Dark Comet (2022). https://github.com/MBCProject/mbc-markdown/blob/master/xample-malware/dark-comet.md#4
Microsoft: Out of sight but not invisible: Defeating fileless malware with behavior monitoring, AMSI, and next-gen AV - microsoft security (2018). https://www.microsoft.com/security/blog/2018/09/27/out-of-sight-but-not-invisibledefeating-fileless-malware-with-behavior-monitoring-amsi-and-next-gen-av
Miramirkhani N, Appini MP, Nikiforakis N, Polychronakis M (2017) spotless sandboxes: evading malware analysis systems using wear-and-tear artifacts. In: 2017 IEEE symposium on security and privacy (SP), pp 1009–1024. IEEE
MITRE: virtualization/sandbox evasion: user activity based checks (2021). https://attack.mitre.org/techniques/T1497/002/
Mitre: mitre attck. https://attack.mitre.org/
Mitre: lazarus group (2021). https://attack.mitre.org/groups/G0032/
Nataraj L, Karthikeyan S, Jacob G, Manjunath BS (2011) Malware images: visualization and automatic classification. In: proceedings of the 8th international symposium on visualization for cyber security, pp 1–7
Ni S, Qian Q, Zhang R (2018) Malware identification using visualization images and deep learning. Comput Sec 77:871–885
odzhan: Shellcode: in-memory execution of DLL (2019). https://modexp.wordpress.com/2019/06/24/inmem-exec-dll/
orr F (2021) Phantom DLL hollowing. https://github.com/forrest-orr/phantom-dll-hollower-poc
O'Murchu L, Gutierrez FP (2015) The evolution of the fileless click-fraud malware poweliks. Symantec Corp
O'Shaughnessy S, Sheridan S (2022) Image-based malware classification hybrid framework based on space-filling curves. Comput Sec 116:102660
Paschen C (2020) Avoiding get-injectedthread for internal thread creatioN. https://www.trustedsec.com/blog/avoiding-get-injectedthread-for-internal-thread-creation/
PCmag: the best antivirus protection for 2022 (2022). https://www.pcmag.com/picks/the-best-antivirus-protection
Pinhero A, Anupama M, Vinod P, Visaggio CA, Aneesh N, Abhijith S, AnanthaKrishnan S (2021) Malware detection employed by visualization and deep neural network. Comput Sec 105:102247
Reza AM (2004) Realization of the contrast limited adaptive histogram equalization (clahe) for real-time image enhancement. J VLSI Signal Proc Syst Signal, Image Video Technol 38(1):35–44
Ronen R, Radu M, Feuerstein C, Yom-Tov E, Ahmadi M (2018) Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135
Sihwail R, Omar K, Ariffin KAZ (2021) An effective memory analysis for malware detection and classification. CMC-Comput Mater Continua 67(2):2301–2320
Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
spotheplanet: Early Bird APC Queue Code Injection (2020). https://www.ired.team/offensive-security/code-injection-process-injection/early-bird-apc-queue-code-injection
Suciu O, Coull SE, Johns J (2019) Exploring adversarial examples in malware detection. In: 2019 IEEE security and privacy workshops (SPW), pp 8–14. IEEE
Team R (2021) RIFT: analysing a lazarus shellcode execution method. https://research.nccgroup.com/2021/01/23/rift-analysing-a-lazarus-shellcode-execution-method/ Accessed Accessed 23 January 2021
Tekerek A, Yapici MM (2022) A novel malware classification and augmentation model based on convolutional neural network. Comput Sec 112:102515
Van&nbsp;Etten A (2018) You only look twice: Rapid multi-scale object detection in satellite imagery. arXiv preprint arXiv:1805.09512
Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, Zheng Q (2020) Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. Comput Netw 171:107138
Vasan D, Alazab M, Wassan S, Naeem H, Safaei B, Zheng Q (2020) Imcfn: Image-based malware classification using fine-tuned convolutional neural network architecture. Comput Netw 171:107138
Ventures R (2022) Download.com. https://download.cnet.com/
VirusShare: VirusShare. https://virusshare.com/
VirusTotal: virustotal. https://www.virustotal.com/gui/home/upload
VMware I (2022) VMware. https://www.vmware.com/
Wagner D, Soto P (2002) Mimicry attacks on host-based intrusion detection systems. In: proceedings of the 9th ACM conference on computer and communications security, pp 255–264
Wang Q, Hassan WU, Li D, Jee K, Yu X, Zou K, Rhee J, Chen Z, Cheng W, Gunter CA et al (2020) You are what you do: hunting stealthy malware via data provenance analysis. In: NDSS
Wang L, Tao D, Wang R, Wang R, Li H (2019) Big map r-cnn for object detection in large-scale remote sensing images. Mathemat Foundations Comput 2(4):299
Yokoyama A, Ishii K, Tanabe R, Papa Y, Yoshioka K, Matsumoto T, Kasama T, Inoue D, Brengel M, Backes M et al (2016) sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In: research in attacks, intrusions, and defenses: 19th international symposium, RAID 2016, Paris, France, September 19-21, 2016, Proceedings 19, pp 165–187. Springer
Yosifovich P, Solomon DA, Ionescu A (2017) Windows internals, part 1: system architecture, processes, threads, memory management, and more. Microsoft Press, USA, pp 113–202
Yu Z, Qing-Zhong L, Tao L, Li-Hua W, Chun S (2015) Research and development of memory forensics. J Software 26(5):1151–1172

Zhou P, Shi W, Tian J, Qi Z, Li B, Hao H, Xu B (2016) Attention-based bidirectional long short-term memory networks for relation classification. In: proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short Papers), pp 207–212

Publisher's Note