Journal of Cybersecurity and Privacy

MDPI

Review

The Evolution of Volatile Memory Forensics

Hannah Nyholm * , Kristine Monteith, Seth Lyles, Micaela Gallegos, Mark DeSantis, John Donaldson and Claire Taylor

Lawrence Livermore National Laboratory, Livermore, CA 94550, USA
* Correspondence: nyholm7@llnl.gov

Abstract: The collection and analysis of volatile memory is a vibrant area of research in the cybersecurity community. The ever-evolving and growing threat landscape is trending towards fileless malware, which avoids traditional detection but can be found by examining a system's random access memory (RAM). Additionally, volatile memory analysis offers great insight into other malicious vectors. It contains fragments of encrypted files' contents, as well as lists of running processes, imported modules, and network connections, all of which are difficult or impossible to extract from the file system. For these compelling reasons, recent research efforts have focused on the collection of memory snapshots and methods to analyze them for the presence of malware. However, to the best of our knowledge, no current reviews or surveys exist that systematize the research on both memory acquisition and analysis. We fill that gap with this novel survey by exploring the state-of-the-art tools and techniques for volatile memory acquisition and analysis for malware identification. For memory acquisition methods, we explore the trade-offs many techniques make between snapshot quality, performance overhead, and security. For memory analysis, we examined the traditional forensic methods used, including signature-based methods, dynamic methods performed in a sandbox environment, as well as machine learning-based approaches. We summarize the currently available tools, and suggest areas for more research.

Keywords: memory dump; memory acquisition; memory forensics; volatile memory; cyber forensics; malware identification; survey; machine learning

Citation: Nyholm, H.; Monteith, K.; Lyles, S.; Gallegos, M.; DeSantis, M.; Donaldson, J.; Taylor, C. The Evolution of Volatile Memory Forensics. *J. Cybersecur. Priv.* **2022**, *2*, 556–572. https://doi.org/10.3390/jcp2030028

Academic Editors: Mário Antunes, Carlos Rabadão and Danda B. Rawat

Received: 5 April 2022
Accepted: 13 July 2022
Published: 20 July 2022

1. Introduction

The acquisition and analysis of volatile memory to identify cyber threats is currently an active area of research in cybersecurity. The importance that computer systems play in modern life continues to grow, and with it, the creativity and capabilities of those who wish to gain access to them unlawfully. Businesses are currently experiencing 50% more cyberattacks per week compared to 2020 [1]. In particular, fileless malware, which only utilizes legitimate programs to infect a computer and leaves no footprint in the file system, continues to increase in prevalence, and is often capable of evading antivirus software products. In fact, fileless malware is believed to be 10 times more successful than other malware types at evading detection [2]. Because of this, we remain in the dark about the full extent of the damage fileless malware causes. However, one security company saw fileless malware attacks increase by 900% during 2020 [3]. As the use of fileless malware continues to skyrocket, memory forensics will remain a central pillar of forensic methods moving forwards. Even in the absence of fileless malware, the data stored in a system's random access memory (RAM) is of high forensic value. Volatile memory contains fragments of encrypted files' contents, lists of running processes, and lists of network connections. Due to the rising use of full disk encryption and other protective measures, it is much more difficult, and often impossible, to extract such information from the file system. When passing through RAM, however, this obscured information can be extracted in an unencrypted, readily available format. Because of this usefulness, a lot of research has

been conducted on the best ways to extract volatile memory, including ways to address the challenges of page-smearing, slow performance, storage costs, and subversion from malware itself.

1.1. Contributions

In this work, we summarize and compare the approaches used to both collect volatile memory and to analyze it for the purpose of malware identification. Previous surveys have been conducted on acquisition [4], but to the best of our knowledge, this is the first survey to cover both acquisition and analysis. In Section 2, we reference existing survey literature on the topics of memory acquisition and volatile memory forensics. In Section 3, we discuss the different techniques used to dump memory images, as well as issues of access level hierarchy, the memory snapshot quality, tool deployment timing, and the effects of the tools on the system's state. In Section 4, we discuss tools (both open source and commercial) used for parsing memory dumps, traditional forensic approaches such as signature scanning and dynamic analysis in a sandbox environment (Section 4.2), and novel forensic methods including machine learning (Section 4.3).

1.2. Threat to Validity

In order to ensure this review considered all relevant works for inclusion, we utilized the following search methods and inclusion/exclusion criteria.

For our survey of volatile memory acquisition (Section 3), we queried the Google Scholar, Scopus, and Elsevier databases with the following key words: Memory Dump, Memory Acquisition, Memory Forensics, Volatile Memory. The results of these queries were added to our list of potential sources. Additionally, we included each document's references as well as subsequent works that cited the documents to our body of literature. To this body of literature we employed the following inclusion/exclusion criteria. To avoid redundancy, works were not considered here if they were referenced in a previous survey that we in turn referenced. We also excluded literature regarding the acquisition of Android and Internet-of-Things (IoT) device memory, which we leave to future work.

For our survey of volatile memory analysis (Section 4), we again queried the Google Scholar, Scopus, and Elsevier databases with these key words: Volatile Memory Analysis, Memory Forensics, Memory Analysis, RAM Analysis, Memory Dump Forensics, Memory Dump, and Malware Identification. The relevant documents produced by these queries were considered potential sources. Again, we also reviewed each of these documents' references and works that cited each of these documents to find additional relevant work. Then, we employed the following inclusion/exclusion criteria. Works were included if their methodology was intended for the identification of malware and included some element of volatile memory in that methodology. Works were not considered here if they did not utilize artifacts from the system's volatile memory. Again, we excluded works related to the analysis of volatile memory from Android or IoT devices; there is a broad range of literature on that subject, which we leave to future work. Additionally, in order to keep this survey relevant for modern operating systems, we excluded methods that were published more than a decade ago.

1.3. Limitations

The main limitation of this review is the potential that we unintentionally excluded relevant references. To the best of our knowledge, we included all relevant sources published prior to this review.

2. Literature Review

Here, we review other surveys and systematizations of knowledge on the topics of volatile memory acquisition and analysis. This work serves as an addition to several excellent existing reviews on the topic of memory acquisition. The area of malware identification using artifacts from volatile memory has only been indirectly or briefly addressed in previous surveys.

2.1. Memory Acquisition Literature

VöMel and Freiling [5] lay the groundwork in this topic area with a survey of acquisition and analysis methods for the Windows operating system. More recently, Or-Meir et al. [6] devoted a section of their work for acquisition through both software and hardware means, explaining the mechanisms and trade-offs of each. Sudhakar [7] thoroughly covered the means by which fileless malware operates as well as the challenges this poses for timely detection. Taylor et al. [8] experimented with different acquisition tools to determine the correctness, impact, and efficacy of memory acquisition tools, particularly by examining the tools' ability to correctly capture read-only memory in order to detect firmware rootkits. Latzo et al. [4] also provided an extensive review of acquisition methods, including specific tools. They defined a taxonomy that still adequately characterizes state-of-the-art memory acquisition tools. We used this taxonomy later to define and compare features of acquisition tools. To this superb review, we simply add subsequent developed methods.

2.2. Volatile Memory Analysis Literature

To the best of our knowledge, there exists no comprehensive, detailed, and current review on the topic of volatile memory analysis methods, and we especially seek to address that need with this review. Memory analysis methods have been widely researched and published, and we attempt to systematize that knowledge for the research community. Some authors have touched on the topic in a broad manner, and we summarize the relevant points below.

Sanjay et al. [9] provided a useful, generalized discussion of the types of fileless malware and the problems they pose. They outlined four categories that characterized memory forensic methods: sandboxing, execution emulation, heuristics, and signature-based methods. We add to this work by specifying memory analysis tools, and by discussing more recent machine learning approaches, which warrant another category. Case and Richard [10] provided a list of shortcomings that current memory forensic techniques experience, including the lack of methodologies aimed at userland malware, application-specific analysis tools, and changes made in Windows 10 that affect memory forensics. They also pointed out that while most cyberattacks continue to happen on PCs, the number of attacks on other devices (iOS, Chromebooks, IoT devices) is bound to rise, and that there are few if any tools to collect or analyze memory from these devices. We found this work to be useful in delineating the future research needs in the area of volatile memory forensics.

3. Memory Acquisition

The quality of any volatile memory analysis is highly dependent on the quality of the memory dump taken from that system, and obtaining a quality memory dump is not a trivial task. VöMel and Freiling [11] describe the qualities that make for a good forensic memory image: correctness, atomicity, and integrity. A memory image is correct if the snapshot contains only values that were present in the memory when the snapshot was taken. Atomicity implies that the memory snapshot was taken within an uninterrupted atomic action, or that the snapshot is free of the signs of concurrent system activity—usually those produced by the memory acquisition tool. Lastly, a snapshot is considered to have integrity if the memory region's values have not changed since the specific point of time chosen by the investigator. These qualities can be measured and compared to evaluate the quality of different memory acquisition techniques. Page smearing, which occurs when the acquired page tables reference physical pages whose contents have changed during acquisition, along with other memory inconsistencies, are prevalent issues that memory acquisition tools face. They often cause memory snapshots to fall short in these qualities [12]. Latzo et al. [4] provided a useful overview of memory acquisition techniques and a taxonomy (Figure 1) that describes the techniques. We use their taxonomy and add new tools and techniques to their overview.

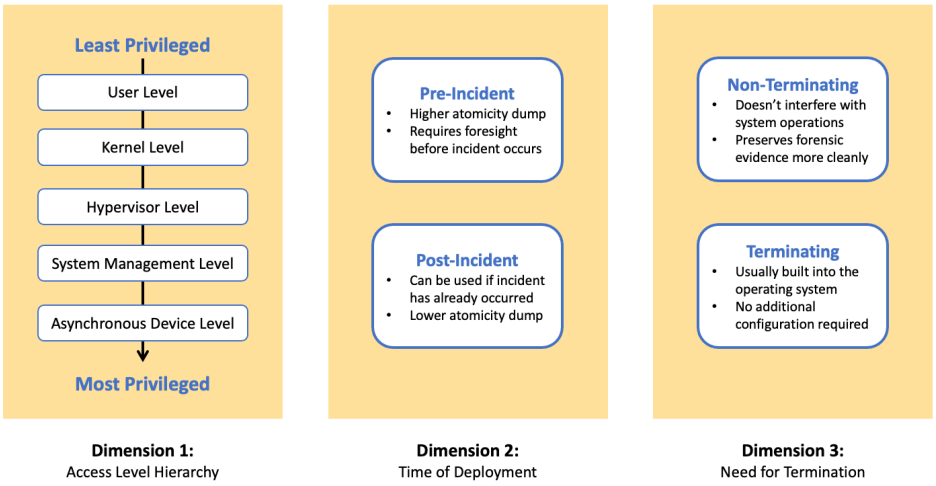

Figure 1. Taxonomy for memory acquisition methods as defined by Latzo et al. [4].

3.1. Taxonomy

One dimension of Latzo's taxonomy [4] is the access hierarchy level in which the memory acquisition tool exists. Acquisition tools that run in a higher access level are less likely to be subverted by malicious content. The hierarchy levels are: user level, kernel level, hypervisor level, synchronous management level (SML), and asynchronous device level (ADL).

The second dimension of the taxonomy involves whether an acquisition tool must be installed pre-incident or post-incident. Some tools require pre-incident deployment, which is not always possible if the incident has already occurred. Aljaedi et al. [13] demonstrated how post-incident tools usually lead to a decrease in the integrity of the snapshot, as some of the memory is overwritten by the tool itself. The last dimension of the taxonomy differentiates terminating and non-terminating acquisition tools. A terminating tool requires programs to abort; a non-terminating tool does not. Non-terminating tools are preferred over terminating tools because they preserve forensic evidence more cleanly and do not interfere with system operations.

3.2. Acquisition Techniques

Here, we review memory acquisition techniques, along with their classifications in Latzo et al.'s taxonomy [4]. We present these acquisition tools subdivided by the access hierarchy level they reside in (Table 1).

3.2.1. User Level

At the user access level, the acquisition of memory could theoretically be accomplished via a software emulator, although these are rarely used in practice for a couple of reasons. Emulating software incurs a high performance overhead and requires low access level, making it highly vulnerable to attack. Because the emulator itself executes the program while dumping the data, it must be deployed pre-incident. Emulators are also non-terminating [4]. Kernel debuggers and virtual machines (in Sections 3.2.2 and 3.2.3, respectively) have largely supplanted user level tools because they have higher performance and increased isolation from the host system.

3.2.2. Kernel Level

At the kernel level, there are many techniques for acquisition, implementing tools as kernel drivers, generating crash dumps or hibernation files, and debuggers running on top of executables. Tools implemented as kernel drivers can typically be deployed post-incident, at the expense of some atomicity and integrity. *WinPmem* (an open source, Windows-based memory dump tool included in the *Rekall* suite) [14], *LiME* (an open source Linux kernel module) [15], and *ProcDump* and *WinKD* (Windows-based tools from Sysinternals which

contain several functionalities, including memory dump capabilities) [16,17] are specific examples of tools implemented as kernel drivers. These tools can create memory dumps from the command line while a target program is running or be configured to watch for certain events, such as a process crash or window hang. Another technique, the utilization of crash dumps, relies on the operating systems' integrated memory acquisition abilities. When the system encounters a critical state, an image of the virtual memory is taken by the operating system for investigative reasons. Because crash dumps are built into the operating system (OS), they should be considered pre-incident, but the programs and the OS need to be properly configured in order to produce dumps. However, crash dumps are, by nature, terminating. Similarly, file hibernation is a built-in capability (and thus a pre-incident tool) that can be used for memory acquisition. When a computer hibernates, most of its physical memory is written to the disk in a hibernation file for storage while the power is not available. This file contains the volatile memory, and can thus be used for analysis as long as there is not full disk encryption on the system and the hibernation mode has been enabled. In Windows, for example, the hibernating file is located at *C:\\hiberfil.sys*, needs no setup to produce, and can be read by *Volatility* [18].

Lastly, software debuggers can be employed for memory acquisition because the debugger has full access to the debuggee's memory. Most modern operating systems come with debugging system calls that can accomplish this. A debugger can be deployed pre-incident by launching the debuggee from the debugger, or post-incident by attaching the debugger to a running process. The *GNU Project Debugger (GDB)* [19] is commonly used for UNIX systems and *WinDbg* [20] or *Visual Studio* [21] for Windows systems. *GDB* can attach to a running process via the command line, or WinDbg can attach from its graphical user interface (GUI) menu.

3.2.3. Hypervisor Level

Because rootkits and other attacks are capable of elevating their privileges to the kernel level, memory acquisition tools running at the kernel level are still not immune to subversion. In some instances, it is desirable to acquire memory from a higher hierarchy access level. Many virtualization tools, such as *VMware* [22] and *LibVMI* [23], have integrated functionality capable of acquiring the guest's memory from the hypervisor level; these tools can be accessed via GUI, command line, and libraries with accessible APIs. These tools must typically be deployed pre-incident, but there are some exceptions, including *HyperSleuth* [24], *Vis* [25], and a tool by Cheng [26]. These tools are unavailable for further analysis but offer architectures that, for instance, rely on thin virtualization layers located outside of even the host operating system. Virtual machines can be paused, and the memory collected via tools provided by the service vendor. *VMware* provides *vmss2core.exe*, and similarly *LibVMI* is packaged with *dump-memory*.

3.2.4. System Management Level

The system management level is an operating mode present on a systems' architecture that is independent of all normal system operations. The system management level only handles low-level operations such as Basic Input/Output System and Unified Extensible Firmware Interface (BIOS/UEFI), not the operating system or user applications, and it has higher privileges than a hypervisor. Because of the independence from the operating system and virtual machines, memory acquisition at this level is an attractive option. However, there are few implementations of acquisition tools at this level. A potential explanation for the dearth of BIOS-level memory capture tools is that BIOS tooling development typically has a slower iteration speed and is less portable than the development for higher-level tools. *SmmBackdoor* [27] is a rare example of such a tool, but its installation is complex and quite specific to computer model. To use SmmBackdoor, a user provisions the software directly to the UEFI system in order to "infect" the System Management Module and capture memory. The author of this tool intended it as an example of an exploit rather than

a typical, blue-team forensic aid. Additionally, it appears that SmmBackdoor is limited in memory scope to the System Management Module's System Management RAM.

3.2.5. Asynchronous Device Level

Hardware-assisted memory acquisition, or acquisition at the ADL level, makes use of external hardware to capture memory. *PCILeech* [28] and *Inception* [29] are direct memory access (DMA)-based frameworks that can be deployed post-incident and are non-terminating. They use external hardware to access memory over system buses, such as peripheral component interconnect express (PCIe); these tools require provisioning drivers for the capture hardware and the tool software itself to interact with that hardware. Once provisioned, *PCILeech* can be invoked via a command line utility, for example:

```
pcileech.exe dump -force -device usb3380://usb=2
```

Because the system continues to run uninterrupted, obtaining atomic memory snapshots is impossible. Another tool, *Snipsnap* [30], utilizes a hardware thread control block (TCB) and an untrusted kernel driver that captures memory in the target OS. This method requires a modest modification of the on-chip memory controller and CPU register file—making it a non-atomic method. However, it offers performance isolation for the applications executing on the target system. Another hardware-based approach, the cold boot technique, takes advantage of the fact the DRAM is not immediately erased after a reset or power cut. To dump the frozen memory during a cold boot, it is possible to use dumping software using a preboot execution environment (PXE), to boot from a USB drive, or to include a dumping routine in the BIOS/UEFI.

Table 1. A summary of memory acquisition tools according to the taxonomy [4]. An 'x' in a column indicates that a tool's category resides in that classification. An 'x' is listed in both the Pre-Incident and Post-Incident column if a category of tools contains examples of both classifications.

Access Level	Type	Tool Name	Pre-Incident	Post-Incident	Terminating	Non-Terminating
Kernel Level	Kernel Drivers	Pmem [14] LiME [15] ProcDump [16]		x		x
	Crash Dump Files	Built in	x		x	
	Hibernation Files	Built in	x		x	
	Debuggers	GNU Project Debugger [19] WinDbg [20] Visual Studio [21]	x	x		x
Hypervisor Level	Hypervisor	VMWare [22] LibVMI [23]	x			x
	Hypervisor	Hypersleuth [24] Vis [25] Cheng et al. [26]		x		x
System Management Level	BIOS-level	SmmBackdoor [27]	x			x
Asynchronous Device Level	Direct Memory Access	PCILeech [28] Inception [29]		x		x
	Hardware Thread Control Block	Snipsnap [30]	x			x
	Cold Boot	Built in	x		x	

3.3. Discussion

Of the developed memory acquisition tools, none provide a perfect memory image that is also practical to deploy and maintains complete security. The available tools capable of delivering correct, atomic snapshots with integrity tend to be impractical for performance reasons or because they require the termination of the system. Acquisition tools that have faster, non-terminating performance typically do so at the expense of the atomicity or integrity of the snapshot. However, if the overwritten or evolving memory sections are small enough, it has been argued that the memory snapshots are almost atomic, and

therefore, good enough for most purposes [25]. Pagani et al. [12] disagreed, arguing that changes in memory across time and space must be accounted for in order to reasonably use non-atomic snapshots. Another issue that many of the acquisition tools suffer from is vulnerability to anti-forensic methods. Memory acquisition tools executed from the hypervisor level and above can be detected and subverted by malware that can identify the presence of the hypervisor or through other sophisticated techniques [31]. Even straight hardware-based approaches can be compromised by time-based detection from the malware [6]. Acquisition tools that are more resistant to anti-forensic methods tend to have higher levels of page-smearing and thus lower atomicity, because the tool does not interrupt the host. Thus far, tools implemented at the system management level have great promise to meet these competing demands of quality, performance, and security, but more research and development is needed in this area.

4. Memory Analysis

Once a memory dump has been obtained, there are numerous methods available to analyze that memory dump for the presence of malware. Generically, the analysis process (Figure 2) includes parsing the memory dump to extract useful information, and then using that information in a specific analysis approach. Several tools, including *Volatility* and *Rekall*, have been created to parse the memory dumps. Some older, more established methods for memory analysis include the signature scanning or heuristic scanning of the memory dump. Newer dynamic methods involve executing malware in a sandbox or another instrumented environment and characterizing it with various features. Lastly, new machine learning techniques are being explored that take the features from dynamic analysis methods and use them to train machine learning classifier algorithms.

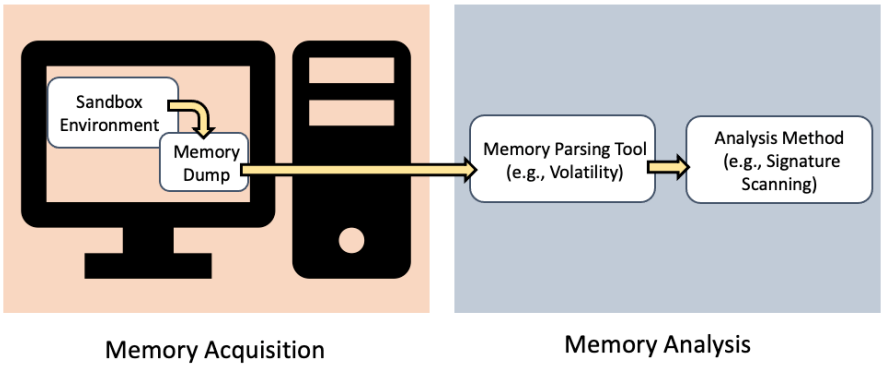

Figure 2. The typical process of acquiring a memory dump and analyzing it for the purpose of malware identification. Malware is executed (usually in a sandbox environment) and a memory dump of the system is taken. Then, the memory dump is parsed using *Volatility* and additional analysis is performed upon the output from *Volatility*.

4.1. Tooling

Several memory analysis tools have been created that allow a user to parse memory dumps for useful artifacts. The open source examples of such tools include *Volatility* [18] and *Rekall* [32], and commercial tools include *Cellebrite Inspector* [33], *FireEye Redline* [34], *Magnet AXIOM* [35], and *WindowsSCOPE* [36]. Almost all research methods make use of the *Volatility* software, and most commercial solutions expose or leverage *Volatility* within their product. Here, we describe some of the features of these tools. Commercial tools typically add features not natively found in *Volatility*, such as enterprise-level remote endpoint management with additional analysis. These additional features, while quite valuable for managing enterprise resources and boosting the efficacy of current methods, do not introduce drastically new practices to the topic here—namely, analyzing memory on individual devices.

4.1.1. Volatility

In discussing volatile memory forensic tools, one cannot neglect to mention *Volatility*—an open source, Python-based framework for analyzing memory dumps [37]. *Volatility* is capable of analyzing memory dumps from Windows, Linux, or Macintosh machines, supports many different types of file dump formats, and has an extensible application program interface (API). It also has excellent feature-creating functionality and is reasonably efficient in its implementation. *Volatility* has emerged as the largest and best-supported framework because of its large base of contributors and independently authored plugins, many of which are geared toward platform-specific forensics [38]. The framework and many of the plugins, especially the plugins for analyzing Windows systems, are mature and stable enough to pass fuzz testing attempts [39]. Fuzz testing is a software testing method that injects invalid or unexpected inputs into a program to reveal defects and vulnerabilities. Most of the novel memory forensic tools discussed in later sections of this paper are implemented as a *Volatility* plugin or utilize the *Volatility* code base.

A main drawback of *Volatility* is that it is typically run from the command line interface, which makes it inaccessible to some potential users. In order to flatten the learning curve associated with using *Volatility*, a graphical user interface (GUI) has also been developed by Meyers et al. [40] that allows inexperienced memory forensic investigators to use basic *Volatility* commands via a plugin in the software *Autopsy*.

While *Volatility* contains many different modes and commands, a typical invocation takes the following form:

```
python3 vol.py -f <dumpfile> windows.pslist
```

In this example, *Volatility* extracts the Windows process list at the time of the memory image dump.

Performance

Volatility developers claim it to be more efficient than other memory forensic software, including *Rekall* [37,41], although an independent assessment of the software's performance was not found. *Volatility* developers assert that *Volatility2* was capable of "list[ing] kernel modules from an 80 GB system in just a few seconds" [37]. Furthermore, during the public release of the *Volatility3* beta, developers claimed major performance improvements over *Volatility2* and *Rekall*, in many instances [41].

Basic Capabilities

Here, we describe some of the basic capabilities of *Volatility* that are widely used for memory forensics. Ligh et al. [42] provided a more thorough discussion and tutorial with an example code for all major architectures.

Using *Volatility*, one can extract the current and previous process handles running on a system, a process's loaded dynamic-link libraries (DLLs), any commands an attacker entered through a console shell, memory resident pages, and process executables. It can also extract information about the process's virtual address descriptor (VAD) nodes, including the addresses of the VAD structure in kernel memory, the VAD tag, the VAD flags, control flags, and the name of the memory mapped file if one exists. It can obtain data about the kernel drivers loaded on the system, including those hidden in physical memory or in files on a disk. Likewise, it finds process thread objects in physical memory with pool tag scanning. Lastly, *Volatility* is capable of obtaining information about the connections the system has made. Specifically, it extracts TCP connections that were active at the time of memory acquisition, listening sockets for any protocol, residual data and artifacts from previous sockets, and network artifacts, including TCP endpoints, TCP listeners, UDP endpoints, and UDP listeners.

4.1.2. Rekall

Rekall originated as a fork of *Volatility* in 2011, but grew to be an advanced forensic and incident response framework. Like *Volatility*, *Rekall* can parse dumps from Windows, Linux, and Mac OS, providing much of the same information regarding a system's processes, memory structures, and network connections. When provisioned to a system, *Rekall* also has a memory dump functionality and enables live memory analysis, similar to that provided in forensic mode by *Volatility*. *Rekall* contains other features such as a useful GUI and methods that better support different operating system versions [32,43]. *Rekall* is no longer maintained by its developers; however, thus limiting its usefulness to future memory forensic researchers.

4.1.3. Discussion

Both of these tools supply a reverse engineer with many tools to manually examine the contents of a memory dump file. They can provide information about the computer system, process memory, the kernel memory and objects, and network connections. These tools are not, on their own, however, an end-to-end platform that can automatically detect the presence of a malicious code in a computer system. They require an experienced operator behind the wheel in order to extract and interpret the contents of the volatile memory. While manual reverse engineering was an important first step in the development of forensic methods, it is time-intensive and requires significant subject matter expertise. Because the rate of malware creation far outpaces experts' ability to reverse engineer them, these tools are most powerful when paired with other software (such as those examined subsequently) that provide some automation in the process of malware detection.

4.2. Traditional Memory Forensic Approaches

Many traditional memory forensic methods for malware identification can be divided into the following categories: a scanning method or a dynamic analyses performed within a sandbox. In this section, we discuss established tools and new extensions of these approaches.

4.2.1. Scanning Methods

A scanning approach in cyber forensics involves searching the forensic data of an infected system—files, memory dumps, process lists, network connections—for evidence of the infection. In this article, we only focus on scanning techniques applied to memory dump files. Scanning the physical memory provides an inherently incomplete picture of a malware's behavior, and can be bypassed by malware authors using obfuscation techniques. However, scanning techniques are fast, making them a good first-response technique. Furthermore, because cyberattacks are often made with recycled malware, scanning techniques can provide a quick and effective forensic for most infections. Some scanning techniques are signature based, and look for specific strings or byte sequences. Others are heuristic based, and instead look for certain commands, logic, or instructions.

Signature Scanning

Signature scanning looks to match the signatures of known malware with the contents of memory dump files from infected systems. A signature is a footprint or pattern, typically including byte patterns and strings, that are unique to the malware type. YARA signatures have emerged as the industry de facto method for scanning, and are directly supported in *Volatility* [44]. The YARA matching engine compares a given sample with a large database of signature rule formats that each represent a type of malware or malware family. The following authors have used signature scanning in their memory forensic approaches.

When evaluating memory with scanning techniques, one can either scan the virtual address space or the memory image directly. Cohen [45] proposed a "context aware" scanning method that uses the Windows Page Frame Number (PFN) database to rapidly identify the owner of each physical page, and where it is mapped in its virtual address

space. This method improves the matching speed and accuracy since the virtual address space need not be reconstructed for every process.

A large obstacle memory forensics techniques face in practice is the amount of storage required to keep a large number of memory dump files. Efforts to create a public repository of memory dump files for the purposes of memory forensics are attempting to address this [46], but we can assume that the storage size will continue to be an issue if malware prevalence continues to increase exponentially. To this end, the use of compression utilities on memory dump files [47], and the creation of signatures based on the compressed memory files, has been considered. However, because malware typically modifies only a small portion of a machine's memory, the majority of data in memory dumps is redundantly stored. Brengel and Rossow [48] address this with their *MemScrimper* methodology. This method involves taking a snapshot of the memory of the clean system, taking a memory snapshot after exploding a single malware sample, finding the difference between those snapshots, saving and compressing the difference, reverting to the clean snapshot, and then repeating for each malware sample to be analyzed. Because the large benign portions of the memory dumps are not compressed and saved, this method is even more space efficient than simply using compression utilities.

Heuristic Scanning

Heuristic scanning is a method that detects threats using rules and algorithms to look for commands or instructions that may indicate malicious intent. The heuristic rules are more generalizable than signatures, allowing heuristics to identify previously unseen malware that shares characteristics with previously identified malware. In practice, heuristic scanning and signature scanning are often used in conjunction with each other. Scanning techniques are often significantly faster than sandboxing techniques. A recent heuristic scanning method for memory forensics was made by Pendergrass et al. [49]. Their contribution, the *USIM* toolkit, is a set of integrity measurement collection tools that look at the abstractions of the operating system and search for violations of invariants that indicate deviation from the expected run-time behavior. Primarily, they examine the abstractions of the namespaces, filesystems, networking and communication channels, environment variables, and runtime linkers/loaders. They also look at abstractions of the virtual memory management. These abstractions are then gathered into a single, graph-based structure, which is appraised by a set of rules defined by the administrator. This toolkit, while a potentially valuable resource for reverse engineers, was not built with the intent to automatically detect the presence of malware, but to allow analysts to better dive deeper into the structure of both memory and non-memory forensic data using rule-based heuristics.

4.2.2. Dynamic Analysis within a Sandbox

Sandboxing can provide a dynamic approach to cyber forensics. Malware is allowed to execute in a controlled environment, called a sandbox, and its behavior and characteristics, including information about its volatile memory, can be recorded [50]. Analyses of the information collected in the sandbox environments can be used to help identify future threats. Sandboxes are a necessary aspect of dynamic forensic techniques, as they protect the analysts' systems from unwanted infection. Bare-metal runs, malware runs outside a sandbox, are not practically scalable because they require a full system reinstall after each malware run to restore the analyst system to a clean slate.

There are two main approaches used to set up a sandbox, virtualization and emulation, and they differ in how the controlled environment is created.

Virtualized Environments

Virtualized environments, called virtual machines, are controlled by a hypervisor. The hypervisor software controls the access of different programs to the underlying hardware, so the virtual machines can be isolated from other virtual machines or programs on the

same hardware. However, the hypervisor and the virtual machine cannot be run at the same time, making it difficult to gather detailed data on the program's execution. The presence of the hypervisor is also difficult to hide from the malware, and malware authors are known to use evasive coding techniques that identify the presence of the hypervisor and subsequently modify the program's behavior [50].

Software Emulators

An emulator is a software program that simulates the functionality of a program or of a piece of hardware. Emulators can simulate the operating system, but due to the complexity of most modern versions, it is often easier to emulate the underlying hardware instead. Since emulators implement their functionality through software, they are wonderfully flexible. An emulator can be built to run guest programs on completely different hardware CPU architectures than that for which they were designed. Furthermore, while the guest program is running, the analyst can obtain an instruction-by-instruction view of what the malware is doing. One typical drawback of execution emulators, however, is a significant performance penalty that is incurred due to the addition of a software level. Furthermore, similarly to hypervisors, emulators can be detected and bypassed by malware through evasive techniques [50].

Sandbox Tools

There are many commercial sandbox systems based on virtualization or emulation, including *AnyRun* [51], *Crowdstrike Falcon Sandbox* [52], *FireEye* [53], *JoeSecurity* [54], *Palo Alto WildFire* [55], and *VirusTotal* [56]. Many such systems would well suit the needs of a corporate computer security team, but are not entirely free or open source, making them less suitable for the research teams focused on memory forensics. Some free, open source sandbox tools include *Cuckoo* [57], *DRAKVUF* [58], *Sandboxie* [59], and *SpeakEasy* [60]. Virtually all behavior-based malware detection research efforts incorporate some sort of sandbox system [61–63]. Users interface with these systems—providing programs and configurations to test and retrieve data to analyze—in a variety of ways, from command line utilities to web requests (via web GUI or HTTP APIs) and local GUIs. For instance, a typical task-creation request for a *Cuckoo* instance running at "<server>" takes the following form:

```
curl -H "Task Name" -F file=@/program http://<server>/tasks/create/file
```

Despite their ubiquity, sandbox systems are not without limitations. Sandbox systems can be difficult to configure correctly so that they effectively counter anti-analysis techniques. There is a plethora of research regarding the evasive techniques employed by malware authors [64,65] and countermeasures to combat sandbox evasion [31,66]. Research has also been conducted to improve the performance and compatibility of sandbox systems. Tien et al. [63] proposed a novel sandbox system which leverages VM introspection and memory forensic techniques. Because most sandbox systems must capture all system call behaviors, they must modify the event monitoring routines of VMExit in the hypervisor kernel, which is highly dependent on the virtualization hypervisor. To decrease this hypervisor dependency and increase the compatibility and scalability, Tien et al. proposed a novel system. The proposed system works by setting up a Xen hypervisor on a system, accessing the virtual machine's memory with *LibVMI*, and then using the *Volatility* framework to analyze system behavior. A few details were provided about the implementation of the system. The authors claim that this approach could be used to analyze live memory data, but appear to only apply it to memory dumps. The accuracy rates of the system at detecting malicious behavior individually in the process, file, and registry systems were somewhat low, but when combined, yielded accuracies as high as 90%.

4.3. Machine Learning Approaches

The use of machine learning (ML)-based approaches in malware detection has become widespread in recent years [67–70]. This is unsurprising because of the great success such

algorithms have achieved at classification problems in a wide variety of domains. In this section, we explore attempts at using ML algorithms in the space of malware detection, specifically with volatile memory forensics. These ML approaches could be classified into two groups, a feature engineering approach and a computer vision-based approach.

4.3.1. Feature Engineering Approaches

Many of the attempts at a machine learning volatile forensic method involved executing malware in a sandbox system, acquiring a memory dump, extracting features from the dump using *Volatility* (or another tool, such as *Rekall*). The creation of these features is often termed feature engineering in machine learning literature. The features can then be used in a classification algorithm. Such was the case with [61,71–73].

Aghaeikheirabady et al. [71] compared sequential minimal optimization, random forest, decision tree, naïve Bayes, and instance-based classifiers at the task of malware detection on volatile memory. The data used included 350 instances of malware, and 200 instances of benign executables. The classifiers were built using 130 features extracted from the VAD tree, the file mapped in memory, registry keys from the process handle table, and registry changes. Specific features were not provided by the authors. As the success of the above-listed classifiers is highly dependent upon the features used to build them, we believe a more thorough discussion of the selected features and the feature importance is needed to replicate these results.

Similarly, Murthaja et al. [61] built machine learning classifiers on extracted features regarding the API calls, DLL injections, registry, and network connections. These features were used to fit several classification algorithms, including naïve Bayes, support vector clustering, K-nearest neighbors, logistic regression, a decision tree, random forest, and linear discriminant analysis, to create a final feature set. A recurrent neural network was then fit with the finalized feature set, and it led to >93% accuracies at detecting malicious processes.

Arfeen et al. [72] extracted 15 features (9 of which were ultimately used after feature selection) from 900 memory dumps. These features were used in an XGBoost classification algorithm to identify the presence of ransomware on the system. Only five families of ransomware were used in the dataset, and they were compared against only three benign processes. The algorithm was able to identify the ransomware with 89% accuracy.

Lashkari et al. [73] extracted 36 features (provided as a list by the authors) from memory dumps, that described the system's processes, DLLs, handles, loaded modules, code injections, connections, sockets, services and drives, and callbacks. These features were fed into randomized decision trees which achieved a 93% true positive rate and a 6.6% false positive rate. However, their 1900 samples were derived from only 7 malware families and a few benign samples.

The promising results in these papers indicate that there is potential for malware detection based off feature engineering from volatile memory snapshots.

4.3.2. Computer Vision Approaches

The following authors adapted the methods used in computer vision to the problem of volatile memory forensics. They re-framed the data to an image (or image-like representation) and then applied computer vision techniques.

Xu et al. [74] divided program execution into epochs and summarized the data access patterns with four histograms of the random memory access. Each histogram considers a different type of memory access: (1) far calls, with an absolute address; (2) near calls, with a relative address; (3) branch instructions; and (4) load/store instructions. Using these histograms as the feature set, logistic regression, a support vector machine (SVM), and random forest classifiers were built for kernel rootkit executions and for user-level memory corruption malware. Results for the kernel rootkit classifiers achieved almost 100% accuracy at detecting rootkits, with a false positive rate of <1%, albeit on a small dataset. For the userland malware, the rate of true positive predictions was high compared to malware-aware processes detection (88%), even when the allowable false positive rate

was small (1%). The unique contribution of this work is the ability to turn it into a live detection system, as well as a unique representation of the data.

Furthermore, taking a cue from computer vision research, Bozkir et al. [75] rendered the memory dumps of processes as images, with a variety of rendering schemes. Then, visual feature descriptors (GIST and HOG) were applied to the images to compute visual features. The visual features were then used in a random forest, XGBoost, linear SVM, sequential minimal optimization (SMO), and J48 classifiers. When treated as a multi-class classification problem, the best model was able to predict the malware family of a new sample with accuracies as high as 96%. While the results of these experiments were promising, the data only contained 10 different malware families.

4.4. Discussion

Each of the many varied tools used to analyze volatile memory have strengths and weaknesses, and some offer fundamentally different approaches to the problem. Table 2 summarizes the features of these tools and shows how they have evolved over the last decade. Although dynamic analysis and machine learning are arguably better approaches to malware detection, signature scanning remains a mainstay in how industry antivirus companies provide protection to their customers. Signature scanning is established, fast, and it reliably identifies the majority of security threats that their customers face. However, it is incapable of identifying new threats. Machine learning-based approaches, although widely talked about, have yet to be widely used in industry. This is at least partially due to the quality of the datasets on which these methods have been tested and the lack of replication of results. In the machine learning approaches reviewed in this survey, all of the datasets used contained a limited number of samples from a few benign and malicious families. While the development for a large publicly available datasets of memory dump images is already underway [46] and will surely aid with this issue, in general, machine learning-based methods still need to be verified on larger, more diverse, and more realistic datasets. However, once larger datasets are used, researchers are likely to run into time complexity issues that often accompany large machine learning models. Secondly, some of the research regarding ML methods reviewed here did not provide enough implementation details to replicate similar results. Providing enough details for replication, or even better, making code bases publicly available, will aid in establishing machine learning as a reliable method for malware detection in volatile memory forensics. Eventually, the publication of trained machine learning models, or transfer learning, as is done in the natural language processing and image recognition spaces, could result in more participation and innovation from those researchers who lack the necessary resources to train large ML models and store the data.

Table 2. A visual summary of the non-commercial volatile memory analysis tools. Ordered by the year the tool was released, we see the increasing popularity of ML methods. We also notice that the use of *Volatility* and sandboxing methods is well established. Interestingly, very few tools are capable of live malware detection.

Tool Name	First Author	Year	Sandbox Analysis	Scanning Method	ML Method	Utilizes Volatility	Live Detection	Automated
	Graziano	2012	x			x		
YARA		2013		x				x
	Aghaeikheirabady	2014			x	x		
AMAL	Mohaisen	2015	x	x				x
	Tien	2017	x			x	x	
	Cohen	2017		x		x		x
	Fowler	2017		x				x
	Xu	2017			x		x	x

Table 2. *Cont.*

Tool Name	First Author	Year	Sandbox Analysis	Scanning Method	ML Method	Utilizes Volatility	Live Detection	Automated
MemScrimper	Brengel	2018		x				x
	Murthaja	2019			x	x		x
USIM Toolkit	Pendergrass	2019		x				x
SpeakEasy		2020	x					
	Lashkari	2021			x	x		
	Bozkir	2021			x	x		x
	Arfeen	2022			x	x		x

4.5. Future Work

There is a continuing need for research in the space of volatile memory acquisition and analysis. The further development of memory acquisition tools, particularly tools that reside in the system management level, is needed to develop atomic, efficient, and secure tools. The further research and validation of memory forensic methods utilizing machine learning algorithms is needed to confirm the effectiveness of those methods. Lastly, the topic of volatile memory forensics on Android and IoT devices was not considered here, but the survey of those methods is also warranted.

5. Conclusions

The landscape of volatile memory acquisition and analysis research is evolving quickly as it has proven to be a valuable forensic method. A variety of memory acquisition tools have been created for the major operating systems, but these tools vary in their accuracy, speed, and practicality. Continuing research and development is needed to create an acquisition tool capable of meeting these competing demands.

Volatility has established itself as the leading memory extraction tool and is utilized in conjunction with most memory forensic methods by researchers. Memory forensic methods can be classified as dynamic analysis from within a sandbox, a scanning method, or a machine learning approach. Methods employing a sandbox characterize malware behavior better than scanning methods, but can be ineffective against malware containing the sandbox evasion code. Scanning methods are fast to implement and already widely used in commercial software, but they fail to identify previously unseen malware and provide only a limited view of the behavior of the malware. The use of machine learning classification algorithms for volatile memory forensics shows promise, but these results need verification on larger datasets in most cases.

Author Contributions: Conceptualization, K.M. and H.N.; investigation, H.N.; resources, H.N. and K.M.; writing—original draft preparation, H.N.; writing—review and editing, C.T., J.D., M.G., M.D., K.M. and S.L.; supervision, K.M.; funding acquisition, K.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Cyberattacks 2021: Statistics from the Last Year. 2022. Available online: https://spanning.com/blog/cyberattacks-2021-phishing-ransomware-data-breach-statistics/ (accessed on 12 July 2022).
2. What Is Fileless Malware? Available online: https://www.trellix.com/en-us/security-awareness/ransomware/what-is-fileless-malware.html (accessed on 12 July 2022).

3. WatchGuard Technologies, I. New Research: Fileless Malware Attacks Surge by 900% and Cryptominers Make a Comeback, While Ransomware Attacks Decline. 2021. Available online: https://www.globenewswire.com/news-release/2021/03/30/2201173/0/en/New-Research-Fileless-Malware-Attacks-Surge-by-900-and-Cryptominers-Make-a-Comeback-While-Ransomware-Attacks-Decline.html#:~:text=Among%20its%20most%20notable%20findings,in%202020%20compared%20to%202019 (accessed on 12 July 2022).
4. Latzo, T.; Palutke, R.; Freiling, F. A universal taxonomy and survey of forensic memory acquisition techniques. *Digit. Investig.* **2019**, *28*, 56–69. [CrossRef]
5. VöMel, S.; Freiling, F.C. A Survey of Main Memory Acquisition and Analysis Techniques for the Windows Operating System. *Digit. Investig.* **2011**, *8*, 3–22. [CrossRef]
6. Or-Meir, O.; Nissim, N.; Elovici, Y.; Rokach, L. Dynamic Malware Analysis in the Modern Era—A State of the Art Survey. *ACM Comput. Surv.* **2019**, *52*, 88 . [CrossRef]
7. Sudhakar, Kumar, S. An emerging threat Fileless Malware: A survey and research challenges. *Cybersecurity* **2020**, *3*, 1. [CrossRef]
8. Taylor, J.; Turnbull, B.; Creech, G. Volatile Memory Forensics Acquisition Efficacy: A Comparative Study towards Analysing Firmware-Based Rootkits. In Proceedings of the 13th International Conference on Availability, Reliability and Security—ARES 2018, Hamburg, Germany, 27–30 August 2018; Association for Computing Machinery: New York, NY, USA, 2018. [CrossRef]
9. Sanjay, B.; Rakshith, D.; Akash, R.; Hegde, V.V. An approach to detect fileless malware and defend its evasive mechanisms. In Proceedings of the 2018 3rd International Conference on Computational Systems and Information Technology for Sustainable Solutions (CSITSS), Bengaluru, India, 20–22 December 2018; pp. 234–239. [CrossRef]
10. Case, A.; Richard, G.G., III. Memory forensics: The path forward. *Digit. Investig.* **2016**, *20*, 23–33. [CrossRef]
11. Vömel, S.; Freiling, F.C. Correctness, atomicity, and integrity: Defining criteria for forensically-sound memory acquisition. *Digit. Investig.* **2012**, *9*, 125–137. [CrossRef]
12. Pagani, F.; Fedorov, O.; Balzarotti, D. Introducing the Temporal Dimension to Memory Forensics. *ACM Trans. Priv. Secur.* **2019**, *22*, 8. [CrossRef]
13. Aljaedi, A.; Lindskog, D.; Zavarsky, P.; Ruhl, R.; Almari, F. Comparative Analysis of Volatile Memory Forensics: Live Response vs. Memory Imaging. In Proceedings of the 2011 IEEE Third International Conference on Privacy, Security, Risk and Trust and 2011 IEEE Third International Conference on Social Computing, Boston, MA, USA, 9–11 October 2011; pp. 1253–1258. [CrossRef]
14. Stüttgen, J.; Cohen, M. Anti-forensic resilient memory acquisition. *Digit. Investig.* **2013**, *10*, S105–S115. [CrossRef]
15. Sylve, J. Lime-linux memory extractor. In Proceedings of the 7th ShmooCon Conference, Washington, D.C., USA, 2012.
16. Russinovich, M.; Richards, A. ProcDump v10.11. 2022. Available online: https://docs.microsoft.com/en-us/sysinternals/downloads/procdump (accessed on 12 July 2022).
17. Safitri, C. A Study: Volatility Forensic on Hidden Files. *Int. J. Sci. Res.* **2013**, *2*, 71–75.
18. Volatility. Available online: https://github.com/volatilityfoundation/volatility (accessed on 12 July 2022).
19. GDB. Available online: https://www.sourceware.org/gdb/ (accessed on 12 July 2022).
20. WinDbg. Available online: https://docs.microsoft.com/en-us/windows-hardware/drivers/debugger/ (accessed on 12 July 2022).
21. Visual Studio. Available online: https://docs.microsoft.com/en-us/visualstudio/debugger/using-dump-files?view=vs-2022 (accessed on 12 July 2022).
22. VMWare. Available online: https://www.vmware.com/ (accessed on 12 July 2022).
23. LibVMI. Available online: https://github.com/libvmi/libvmi (accessed on 12 July 2022).
24. Martignoni, L.; Fattori, A.; Paleari, R.; Cavallaro, L. Live and Trustworthy Forensic Analysis of Commodity Production Systems. In *Recent Advances in Intrusion Detection*; Jha, S.; Sommer, R.; Kreibich, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2010; pp. 297–316.
25. Yu, M.; Qi, Z.; Lin, Q.; Zhong, X.; Li, B.; Guan, H. Vis: Virtualization enhanced live forensics acquisition for native system. *Digit. Investig.* **2012**, *9*, 22–33. [CrossRef]
26. Cheng, Y.; Fu, X.; Du, X.; Luo, B.; Guizani, M. A lightweight live memory forensic approach based on hardware virtualization. *Inf. Sci.* **2017**, *379*, 23–41. [CrossRef]
27. Oleksiuk, D. Building Reliable SMM Backdoor for UEFI Based Platforms. 2015. Available online: http://blog.cr4.sh/2015/07/building-reliable-smm-backdoor-for-uefi.html (accessed on 12 July 2022).
28. PCILeech. Available online: https://github.com/ufrisk/pcileech (accessed on 12 July 2022).
29. Inception. Available online: https://github.com/carmaa/inception (accessed on 12 July 2022).
30. Cox, G.; Yan, Z.; Bhattacharjee, A.; Ganapathy, V. Secure, Consistent, and High-Performance Memory Snapshotting. In Proceedings of the CODASPY'18: Proceedings of the Eighth ACM Conference on Data and Application Security and Privacy, Tempe, AZ, USA, 19–21 March 2018; Association for Computing Machinery: New York, NY, USA, 2018; pp. 236–247. [CrossRef]
31. Besler, F.; Willems, C.; Hund, R. Countering innovative sandbox evasion techniques used by malware. In Proceedings of the 29th Annual FIRST Conference, San Juan, Puerto Rico, 11–16 June 2017.
32. Rekall. Available online: https://github.com/google/rekall (accessed on 12 July 2022).
33. Cellebrite Inspector. Available online: https://cellebrite.com/en/inspector/ (accessed on 12 July 2022).
34. FireEye Redline. https://www.fireeye.com/services/freeware/redline.html (accessed on 12 July 2022).
35. Magnet Axiom. Available online: https://www.magnetforensics.com/products/magnet-axiom/ (accessed on 12 July 2022).

36. WindowsSCOPE. http://www.windowsscope.com/windowsscope-cyber-forensics/.
37. Volatility Foundation. Available online: https://www.volatilityfoundation.org/ (accessed on 12 July 2022).
38. Volatility Community Plugins. Available online: https://github.com/volatilityfoundation/community (accessed on 12 July 2022).
39. Case, A.; Das, A.K.; Park, S.J.; Ramanujam, J.R.; Richard, G.G. Gaslight: A comprehensive fuzzing architecture for memory forensics frameworks. *Digit. Investig.* **2017**, *22*, S86–S93. [CrossRef]
40. Meyers, C.; Ikuesan, A.R.; Venter, H.S. Automated RAM analysis mechanism for windows operating system for digital investigation. In Proceedings of the 2017 IEEE Conference on Application, Information and Network Security (AINS), Miri, Sarawak, Malaysia, 13–14 November 2017; pp. 85–90. [CrossRef]
41. Auty, M.; Case, A. Volatility 3 Public Beta: Insider's Preview. In Proceedings of the OSDFCon 2019, Open Source Digital Forensics Conference, Herndon, VA, USA, 15–17 October 2019.
42. Ligh, M.H.; Case, A.; Levy, J.; Walters, A. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*; John Wiley & Sons: Hoboken, NJ, USA, 2014.
43. Cohen, M. Forensic analysis of windows user space applications through heap allocations. In Proceedings of the 2015 IEEE Symposium on Computers and Communication (ISCC), Larnaca, Cyprus, 6–9 July 2015; pp. 237–244. [CrossRef]
44. Available online: http://virustotal.github.io/yara/ (accessed on 12 July 2022).
45. Cohen, M. Scanning memory with Yara. *Digit. Investig.* **2017**, *20*, 34–43. [CrossRef]
46. Orgah, A.; Richard, G., III; Case, A. MemForC: Memory Forensics Corpus Creation for Malware Analysis. In Proceedings of the International Conference on Cyber Warfare and Security, 25–26 February 2021; pp. 249–256.
47. Fowler, J.E. Compression of Virtual–Machine Memory in Dynamic Malware Analysis. *J. Digit. Forensics Secur. Law* **2017**, *12*, 9. [CrossRef]
48. Brengel, M.; Rossow, C. MemScrimper: Time-and Space-Efficient Storage of Malware Sandbox Memory Dumps. In Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, Saclay, France, 28–29 June 2018; Springer: Berlin/Heidelberg, Germany, 2018; pp. 24–45.
49. Pendergrass, J.A.; Hull, N.; Clemens, J.; Helble, S.; Thober, M.; McGill, K.; Gregory, M.; Loscocco, P. Technical report: A toolkit for runtime detection of userspace implants. *arXiv* **2019**. arXiv:1904.12896.
50. Kruegel, C. Full system emulation: Achieving successful automated dynamic analysis of evasive malware. In Proceedings of the BlackHat USA Security Conference, 2–7 Aug 2014, Las Vegas, NV, USA, 2014; pp. 1–7.
51. AnyRun. Available online: https://any.run/ (accessed on 12 July 2022).
52. CrowdStrike Falcon. Available online: https://www.crowdstrike.com/products/threat-intelligence/falcon-sandbox-malware-analysis/ (accessed on 12 July 2022).
53. FireEye. Available online: https://www.fireeye.com/ (accessed on 12 July 2022).
54. Joe Security. Available online: https://www.joesecurity.org/ (accessed on 12 July 2022).
55. Palo Alto Wildfire. Available online: https://www.paloaltonetworks.com/products/secure-the-network/wildfire/ (accessed on 12 July 2022).
56. VirusTotal. Available online: https://www.virustotal.com/gui/ (accessed on 12 July 2022).
57. Cuckoo Sandbox. Available online: https://cuckoosandbox.org/ (accessed on 12 July 2022).
58. Drakvuf. Available online: https://drakvuf-sandbox.readthedocs.io/en/latest/ (accessed on 12 July 2022).
59. Sandboxie. Available online: https://github.com/sandboxie (accessed on 12 July 2022).
60. FireEye SpeakEasy. Available online: https://github.com/fireeye/speakeasy (accessed on 12 July 2022).
61. Murthaja, M.; Sahayanathan, B.; Munasinghe, A.; Uthayakumar, D.; Rupasinghe, L.; Senarathne, A. An Automated Tool for Memory Forensics. In Proceedings of the 2019 International Conference on Advancements in Computing (ICAC), Malabe, Sri Lanka, 5–6 December 2019; pp. 1–6. [CrossRef]
62. Mohaisen, A.; Alrawi, O.; Mohaisen, M. AMAL: High-fidelity, behavior-based automated malware analysis and classification. *Comput. Secur.* **2015**, *52*, 251–266. [CrossRef]
63. Tien, C.W.; Liao, J.W.; Chang, S.C.; Kuo, S.Y. Memory forensics using virtual machine introspection for Malware analysis. In Proceedings of the 2017 IEEE Conference on Dependable and Secure Computing, Taipei, Taiwan, 7–10 August 2017; pp. 518–519. [CrossRef]
64. Afianian, A.; Niksefat, S.; Sadeghiyan, B.; Baptiste, D. Malware dynamic analysis evasion techniques: A survey. *ACM Comput. Surv. (CSUR)* **2019**, *52*, 126. [CrossRef]
65. Yokoyama, A.; Ishii, K.; Tanabe, R.; Papa, Y.; Yoshioka, K.; Matsumoto, T.; Kasama, T.; Inoue, D.; Brengel, M.; Backes, M.; et al. Sandprint: Fingerprinting malware sandboxes to provide intelligence for sandbox evasion. In Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses, Paris, France, 19–21 September 2016; Springer: Berlin/Heidelberg, Germany, 2016; pp. 165–187. [CrossRef]
66. Chailytko, A.; Skuratovich, S. Defeating sandbox evasion: How to increase the successful emulation rate in your virtual environment. In Proceedings of the ShmooCon 2017, Washington, DC, USA, 13–15 January 2017.
67. El Merabet, H.; Hajraoui, A. A survey of malware detection techniques based on machine learning. *Int. J. Adv. Comput. Sci. Appl.* **2019**, *10*, 366–373. [CrossRef]
68. Singh, J.; Singh, J. A survey on machine learning-based malware detection in executable files. *J. Syst. Archit.* **2020**, *112*, 101861. [CrossRef]

69. Souri, A.; Hosseini, R. A state-of-the-art survey of malware detection approaches using data mining techniques. *Hum.-Centric Comput. Inf. Sci.* **2018**, *8*, 3. [CrossRef]
70. Shaukat, K.; Luo, S.; Varadharajan, V.; Hameed, I.A.; Xu, M. A Survey on Machine Learning Techniques for Cyber Security in the Last Decade. *IEEE Access* **2020**, *8*, 222310–222354. [CrossRef]
71. Aghaeikheirabady, M.; Farshchi, S.M.R.; Shirazi, H. A new approach to malware detection by comparative analysis of data structures in a memory image. In Proceedings of the 2014 International Congress on Technology, Communication and Knowledge (ICTCK), Mashhad, Iran, 26–27 November 2014; pp. 1–4.
72. Arfeen, A.; Asim Khan, M.; Zafar, O.; Ahsan, U. Process based volatile memory forensics for ransomware detection. *Concurr. Comput. Pract. Exp.* **2022**, *34*, e6672. [CrossRef]
73. Lashkari, A.H.; Li, B.; Carrier, T.L.; Kaur, G. VolMemLyzer: Volatile Memory Analyzer for Malware Classification using Feature Engineering. In Proceedings of the 2021 Reconciling Data Analytics, Automation, Privacy, and Security: A Big Data Challenge (RDAAPS), Hamilton, ON, Canada, 18–19 May 2021; pp. 1–8. [CrossRef]
74. Xu, Z.; Ray, S.; Subramanyan, P.; Malik, S. Malware detection using machine learning based analysis of virtual memory access patterns. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Lausanne, Switzerland, 27–31 March 2017; pp. 169–174.
75. Bozkir, A.S.; Tahillioglu, E.; Aydos, M.; Kara, I. Catch them alive: A malware detection approach through memory forensics, manifold learning and computer vision. *Comput. Secur.* **2021**, *103*, 102166. [CrossRef]