

Volatools: Integrating Volatile Memory Forensics into the Digital Investigation Process

Aaron Walters Nick L. Petroni, Jr.
awalters@komoku.com npetroni@komoku.com
Komoku, Inc.
College Park, MD, USA

Abstract

In this work, we demonstrate the integral role of volatile memory analysis in the digital investigation process and how that analysis can be used to help address many of the challenges facing the digital forensics community. We also provide a look at some of the shortcomings of existing approaches to live response. Finally, we provide the technical details for extracting in-memory cryptographic keying material from a popular disk encryption application without knowledge of the password.

Keywords: computer forensics, digital evidence, digital investigation, incident response, volatile memory analysis

1 Overview

Recently, a growing amount of attention has been given to research and advancement in the area of volatile memory forensic analysis. Despite this increased attention, we have found that very few investigators feel they have the time, resources, or expertise to include volatile memory analysis in their digital investigation process. Some investigators, many of whom are faced with a backlog of cases, view volatile system memory as yet another “substance” that needs to be analyzed. As a result, volatile memory is often not being collected during incident response and, in instances when it has been collected, analysis typically focuses on searching for context-free strings. The purpose of this work is to emphasize the message that volatile memory forensics should not be thought of as an “extra task” for the overwhelmed digital investigator, but rather an integral part of the digital investigative process. We will also demonstrate how this type of analysis can be an easy transition for those who are already leveraging the context provided from live system response techniques.

There are a number of important forensic principles to consider when extracting data from a live system. These considerations focus on trying to minimize the changes to the system, understanding the effects of the changes, and minimizing the trust that is placed in the system [7]. Unfortunately, little work has been done to evaluate how well current practices in live data collection adhere to these principles. We will provide some initial insight into the limitations and obtrusiveness of various tools and techniques that are typically used for live response. We will also introduce Volatools, a toolkit for Windows XP SP2 memory dumps that will allow digital investigators to extract useful information that was previously only available on a live system. These tools were designed to help digital investigators in the Survey Phase of the investigative process by allowing response teams to quickly identify runtime system context and investigative leads that maybe useful in later analysis phases.

Finally, we will demonstrate how integrating volatile memory analysis into the Survey Phase of the digital investigation process can help address a number of the top challenges facing digital forensics. Volatile memory analysis tools and techniques can be used to complement the tools and practices that are currently

utilized during digital investigations. For example, we will demonstrate how investigators can leverage the context found in volatile memory to focus investigations with large volumes of evidence and triage temporally volatile information before it dissipates. We will also demonstrate the ability to extract keying material from a system memory dump in order to later mount an encrypted filesystem without knowledge of the password.

2 Related Work

One of the most thoroughly discussed topics among academics and practitioners in the field of digital forensics is the importance of a process model for digital investigation [7, 28, 29, 32, 36, 37]. A process model allows us to organize the way we think about, discuss, and implement the procedures that are typically performed during an investigation. It also forces us to think about all phases of an investigation and how the tools and techniques that are used during a digital investigation fit together to support the process. In this work, we will frame our discussion around the Integrated Digital Investigation Process model (IDIP) proposed by Carrier and Spafford [7].

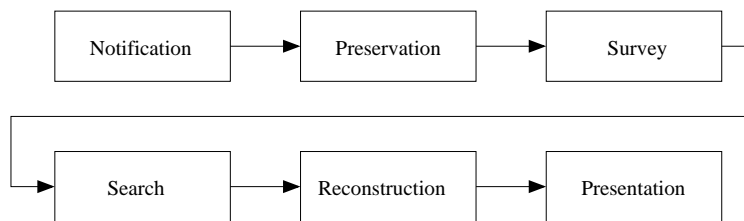


Figure 1: Digital investigation phases in IDIP model [8]

As seen in Figure 1, the IDIP model breaks down a digital crime scene investigation into six phases: Notification, Preservation, Survey, Search, Reconstruction, and Presentation. While the actual phases of a digital investigation are often dependent on the specific characteristics of the incident, the general nature of this model allows it to be applicable to many types of investigations. This model also focuses on the computer as another crime scene that must be investigated, as opposed to just another piece of evidence. Similar to a physical crime scene, Carrier and Spafford argue that a digital crime scene is composed of many different types of evidence that construct the digital environment. Based on this model, we view volatile memory as a critical aspect of the digital environment and discuss how volatile memory analysis can influence the Survey Phase of this process.

The first paper to discuss the possibility of reliably and accurately extracting evidence from volatile memory focused on the Preservation Phase of this same model [8]. Since the Tribble paper, there has been a substantial amount of work discussing the importance of volatile memory analysis and hypothesizing about the possible types of data that can be extracted. This work began with the DFRWS Memory Analysis Challenge in 2005, which was intended to motivate research and tool development in this area. Both of the winning submissions [3, 17] relied on list-walking and table-crawling approaches to extract information from volatile memory images of Windows 2000. Similar techniques were also being used by Mariusz Burdach [4] to extract digital artifacts from both Linux and Windows memory images. In contrast, others have begun to use techniques that perform a linear-scan of memory images in order to also find both linked and unlinked objects [19, 44]. Still other work has discussed technical details such as address translation [2, 34, 26], pool-tags [43], swap integration [34, 26], and carving binaries out of memory [2, 19]. While this research has exposed many interesting facets of an opaque operating system, many investigators want to know how these developments in volatile memory analysis can improve their own methods and techniques

in the field.

3 Live Response

For many investigators, live response has become an important aspect of the digital investigation process [10, 29, 23]. During live response, a responder collects information from a running machine about active objects. Live response offers the ability to peer into the runtime state of the system providing valuable context for an investigation that had been historically lost with “snatch and grab methods.” It also allows a digital investigator to survey the digital crime scene to determine the most economically justified response. An investigator typically performs live response using a response toolkit [30, 29, 31, 25]. These toolkits are frequently automated wrapper programs or scripts that run a series of command-line programs and redirect the output to a forensic workstation or peripheral media. In many cases, the tools run during live response are generic system administration tools that are utilized with few, if any, changes to support digital investigations. There are also a couple of commercial agent-based systems that are designed to collect similar information.

Live response is an important step in the advancement of digital investigation capabilities. Through it, investigators are able to gain access to information and form hypotheses faster than they have been able to in the past. Additionally, through the development of commercial and freely available tools, live response has enabled users and system administrators without forensics training to be more effective when they suspect an incident has occurred. However, there are also some disadvantages of live response and it is important to make informed decisions, particularly in an environment where a simple query into suspicious activity can quickly unravel into a full-fledged crime scene. The following is a set of drawbacks to consider when deciding whether live response makes sense in your situation or organization:

1. Live response does not fit cleanly in the IDIP model.
2. Live response disturbs the state of the machine under investigation.
3. The results of live response are unrepeatable.
4. Live response does not allow for new questions or clarifications to be asked later in the investigative process.
5. Most techniques for live response rely heavily on the trustworthiness of the target’s operating system.

The remainder of this section addresses each of these issues in more detail.

3.1 Live Response and IDIP

When viewed in light of the IDIP model, live response combines many aspects of both the Preservation and Survey Phases. The Preservation Phase involves securing and preserving the digital crime scene, while the Survey Phase relates to quickly identifying investigation leads based on the preserved evidence. In the case of live response, the crime scene has not been secured or preserved, but preliminary evidence is still collected. An attempt is generally made to preserve and secure the evidence (tool output), rather than the raw data provided as input to the tool. While building the IDIP model, Carrier made the following statement about the reliability of evidence collected from a live system, “the challenge is to minimize the changes, understand the effect of those changes, and minimize the trust that is placed on the operating system of the live system” [7]. While many have discussed the types of information that can be collected and the tools that can be used during live response [10, 29, 23], little work has been done to address or quantify responses to the challenges that Carrier laid out.

3.2 Impact on a Target Host

Best practice requires that an investigator must, when performing any act on a machine under investigation, minimize the impact of that act on the target system and understand the effects of that act. It is commonly understood that, when investigators performs live analysis, they are altering the digital environment in which the incident occurred. However, we are not aware of any work that quantifies or characterizes the actual impact of performing live response. In this work, we attempt to quantify this impact as the percentage of bytes of physical memory that were different after running a typical live response toolkit.

Our experiments were performed using a Windows XP SP2 virtual machine running in VMware 5.5.3 build-34685 on a Linux host. Table 1 provides detailed information about our test environment. The live response toolkit used in our experiments was the Windows Forensic Toolchest (WFT), which was run from an e-fense Helix compact disc (version 1.8-10-05-2006), with the output stored directly to a network share (also mounted at the time of response). The default Helix configuration of WFT was used and all the commands were run except those which would modify the local disk.

Base OS	Gentoo 1.12.6
Kernel:	Linux 2.6.16-gentoo-r7
Memory:	1GB RAM
Processor:	P4 Hyperthread 2.8GHz

Table 1: Test environment details

In order to evaluate the impact of performing live response with WFT, we needed to have an experimental baseline that would provide a useful frame of reference. This baseline was created by taking periodic snapshots at specific intervals to track the percentage of bytes of physical memory that changed as a function of time for the Windows XP SP2 machine performing no user tasks. (The only load on the system was due to standard background services, such as McAfee Antivirus and the default screen saver.) These baselines were generated for virtual machines that were given 256 MB and 512 MB of physical memory. As shown in Table 2, for the machine possessing 512 MB of physical memory, 96.7% of the bytes in memory remained unchanged after 60 minutes. After 900 minutes, 85% of the bytes still remained unchanged. In the case of the machine with 256 MB of physical memory, 90.4% of the bytes in memory did not change after the first 60 minutes. After 900 minutes, 73.8% of the bytes in memory remained unchanged. As expected, we can see from these results that the percentage of modified bytes increases as a function of time and the machine with less memory experienced a higher percentage of changes, which we hypothesize is due to a greater need for paging memory.

Minutes	256MB			512MB		
	Different	Total	% Same	Different	Total	% Same
0	0	268435456	100.00	0	536870912	100.00
5	14175676	268435456	94.7191	10059162	536870912	98.1263
10	16253700	268435456	93.9450	12272451	536870912	97.7141
60	25731960	268435456	90.4141	17804666	536870912	96.6836
120	54416944	268435456	79.7281	20542693	536870912	96.1736
900	70430438	268435456	73.7626	77252490	536870912	85.6106

Table 2: Bytes of physical memory that changed as a function of time (unused system)

Now that we have a frame of reference, we can evaluate the impact of running WFT on each of the

previously described virtual machines. This data was collected by taking a memory capture before the tool was run and another one immediately after it finished. For the machine with 256 MB of physical memory, only 67.2% of the bytes in memory remained unchanged after the tool had finished. The machine with 512 MB was impacted slightly less, with 69.4% of the bytes remaining unchanged. We also evaluated the case where `dd` was the only tool run from the Helix CD. After running `dd` on the machine with 256 MB of physical memory, 76.9% of the bytes in memory remained unchanged. In the case of the machine with 512 MB, 89.8% bytes remained unchanged. As a point of comparison, it is interesting to note that there was a larger percentage of bytes changed in both the 256MB and 512MB after running WFT than if we had let the machine continue to run unused for at least 15 hours.

While these results are preliminary and only provide a high-level perspective, we can see that running live response toolkits can substantially alter the digital environment. More research needs to be done to understand the effects of those changes and to determine if those changes are destroying potential evidence. These results do emphasize that, if we are going to rely on these tools for live response, more analysis needs to be done to understand their effects. The results also exemplify why taking a volatile memory dump should be the first task performed on the system, particularly if live response is to be utilized. Finally, the results also demonstrate that reducing live collection to a simple volatile memory capture can reduce our impact on the system.

Our experiments demonstrate that live response may change the underlying environment significantly in some cases. However, this is not, at least in our view, the most compelling reason to take caution with live response. The remainder of this section addresses the three most significant challenges facing live response.

3.3 Repeatability of Live Response

For some situations, having complete information about what really happened in an incident is far more important than collecting indisputable proof or being able to convince others of your findings. However, in many cases, particularly in those of interest to a court of law, the ability to provide proof of how a conclusion was reached is essential. Generally speaking, automated tools that perform a deterministic procedure to aggregate, summarize, or display data can be objectively verified or reproduced by a third-party expert. However, in the case of live response, while the tools themselves can be verified, the input data on which they are run can never be reproduced. Live response tools, by their very nature, are run on an unknown system that is in an unknown state. Furthermore, even if we could somehow argue that, because the tool itself is well-formed, the output could only have been produced based on a particular set of inputs, each action performed by the tool clearly modifies future inputs to the same tool or other tools used on the same system. Simply put, we can never reproduce the exact same inputs to the exact same tools, thereby making it difficult to prove the correctness of any results that have been gathered.

3.4 Asking New Questions Later

As discussed above, current live response procedures do not provide a mechanism to collect the exact same input state as it existed when the tool was originally run. An additional result of this property is that it is impossible to examine the same data in a different way, either to ask for more detail or to ask a different question altogether. Consider, for example, a tool suite that lists a large amount of information about a running system. This information might include all open network connections, running processes, loaded shared libraries, etc. If, at a later point in the investigation, more detail is needed, for example some program-specific property of a given process, it will be impossible to retrieve that information.

3.5 Trust

One of the fundamental prerequisites for gaining a true understanding of what occurred during an incident is having accurate data. Having confidence that the evidence under consideration is, in fact, a true representation of the underlying system is paramount. In terms of live response, we have already pointed out the importance of input data. Unfortunately, because all live response tools of which we are aware run directly on a potentially compromised system, they rely on the underlying operating system, and in some cases system libraries, for accurate access to resources. Even tools which attempt to determine the integrity of the operating system may be fooled if the attacker has perfect knowledge of the tool and control of the system before it is installed. Furthermore, on the majority of commodity systems, it is impossible to even know if the live response tool itself has been run in an unmodified way. This means that, even if the tool itself has been verified, the executing instance of that tool may be untrustworthy.

3.6 Summary

This section has pointed out some of the advantages and caveats of live response. The takeaway is not that live response should be abandoned, but rather that, as with any forensics tool or technique, it is important to understand both the costs and benefits of each mechanism to be utilized. As always, understanding your goals and environment, having a plan, and being well-informed is the best strategy before taking action. Live response is an incredibly valuable tool. However, as we demonstrate in the remainder of this paper, direct analysis of volatile memory may be a much better fit for some investigators.

4 Volatile Memory Analysis

In this section, we describe how the direct analysis of volatile memory images can help address some of the challenges facing live response. Volatile memory analysis can be used instead of or in addition to live response, depending on the goals of a particular investigation.

4.1 Integration into IDIP

Volatile memory analysis can be cleanly separated into two phases. First, in the Preservation Phase, volatile memory can be captured using a number of possible mechanisms, as described in the literature. Next, in the Survey Phase, the preserved data can be verifiably copied and analyzed in a more trusted, well-defined environment. Unlike live response, analysis does not itself put in jeopardy the evidence being collected.

4.2 Impact on Target Host

Because collection is the only task performed while the system is live, we are much more optimistic that mechanisms can be improved to minimize impact on the collected evidence. As shown in our experiments regarding live response, even simple mechanisms such as copying memory from an operating system virtual device do not impact the system as much as live response. Additionally, as volatile memory analysis becomes more mainstream, reliable memory capture is a feature that could be integrated directly into system design. Fundamentally, the impact on the host is strictly a function of the mechanism used to perform capture and there is good reason to believe this can be minimized.

4.3 Repeatability

When volatile memory images are analyzed directly, the raw data used by the tools and the tools themselves can be made accessible to any third-party reviewer who can then reach his or her own conclusions about

the validity of the evidence. Furthermore, the same operations can be repeated and even reimplemented for comparison and tool testing. Even collection mechanisms might be verifiable by utilizing multiple at or near the same time and then comparing the low-level data for bitwise differences.

4.4 Asking New Questions Later

Because low-level memory data is available, tools can be modified or new tools written to extract more patterns, information, or evidence from the same underlying data. Unlike live response, volatile memory analysis allows for questions that remain unanswered at the time of initial analysis to possibly be answered at a later date with more expertise, understanding, or additional evidence. For example, if a particular malware technique is unknown to authorities at the time live response is performed, it is unlikely that later analysis will be able to determine whether the malware existed on the target machine. However, if direct memory analysis is performed, new detection techniques can be implemented and used on the original data.

4.5 Trust

Volatile memory analysis provides a clean separation between data collection and data analysis. As a result, more time can be spent characterizing the effectiveness of data capture mechanisms, which are likely to be simpler and better understood, while data analysis can be run on trusted machines in the presence of trusted personnel. Furthermore, the repeatability of memory analysis lends itself to a higher degree of confidence in the evidence produced by it.

4.6 Summary

While we believe the above are strong reasons to favor volatile memory analysis, or at least augment live response with some direct analysis, memory analysis is not without its challenges. First, it requires a toolset and expertise that the vast majority of investigators do not currently possess. Our intention is to directly address this problem by releasing Volatools, described next. Still, there are other challenges facing volatile memory analysis. We describe these later in this paper.

5 Volatools

In this section, we introduce the Volatools reference implementation in more detail. Volatools is a command-line toolkit intended to assist with the Survey Phase of a digital investigation, similar to tools that are currently used for live data collection. In fact, it is our intention to demonstrate that the information investigators are currently collecting from a live machine [28] can also be extracted offline from a volatile memory image. As a result, these investigators are able to reduce their obtrusiveness on the state of the system and produce repeatable results. This toolkit is intended to provide an open-source tool set that can introduce digital investigators to volatile memory analysis and allow investigators to begin to integrate these techniques into their digital investigation process. As it is only a reference implementation, this version of the tool set only supports analysis of valid memory pages and is focused on support for Windows XP SP2. This implementation also only supports the list-walking and table-crawling approaches previously mentioned, the limitations of which will be discussed at the end of the paper. All programs in the Volatools toolkit are currently implemented in the Python programming language [35]. The following are a representative subset of the type of information that can be collected from a volatile memory image with the Volatools toolkit:

- *Date and Time.* System date and time information can play a very important role in volatile memory analysis and the digital investigation. It can prove useful for both documenting when the memory

acquisition was performed and for correlating with other times to create a timeline of events that occurred within the digital crime scene. For example, kernel objects associated and sockets and processes occasionally have creation times associated with them. This information is also useful for correlating information between multiple machines that may be involved in the incident and establishing an estimate of relative clock drift. During live data collection, this information would typically be generated by running the `date` and `time` commands on the system under investigation [23, 10, 29].

- *Running Processes.* Information about running processes on the system also has the potential to provide valuable information. It may provide the investigator the ability to detect rogue or suspicious processes on the system that an adversary or piece of malware is using to persist on the system. During a live data collection, this information would typically be generated by running `PsList` [41] on the system under investigation [23, 10, 29].
- *Open Ports.* Open ports may also provide useful investigation leads about suspicious activity on the system. As with enumerating processes, it could also provide insight into an adversary's attempt to persist on the system. Open sockets could be listening for incoming connection and provide a backdoor into the system. It also provides information about possible attack vectors into the system. In Section 6.1, we will discuss how this information can be useful for volatile targeting. During a live data collection, this information would typically be generated by running `netstat` on the system under investigation [23, 10, 29].
- *Process to Port Mappings.* Sometimes finding an open socket does not provide enough context to determine if something suspicious is happening on the system and if that socket is being used legitimately. Thus it can be extremely useful to enumerate which process and associated executable is responsible for having that particular port open on the system. During a live data collection, this information would typically be generated by running `fport` [14] on the system under investigation [23, 10, 29].
- *Strings to Process Mappings.* Until recently the primary form of analysis performed on volatile memory dumps involved searching for contiguous strings across the physical address space representation found in the file. One of the limitations of this method is the fact that this is a context free search. In Volatools, we now provide the ability to map those string back to the virtual address space in which they are actually utilized. In essence, providing the ability to map the physical offset of the string back to its corresponding virtual address.
- *Process to File Mappings.* Files often play a critical role in a digital investigation and the ability to extract extra context about those files can prove valuable during the investigation. For example, the ability to determine which process is accessing a suspicious file can often provide very useful investigative leads. Likewise, the ability to enumerate the files and directories a process is accessing can be helpful for determining what the process is doing or elucidating suspicious activity. During a live data collection, this information would typically be generated by running `handles.exe` command [39] or `FileMon` [42] on the system under investigation [10, 29, 23].
- *Process DLLs.* Microsoft Windows supports the ability to dynamically load and unload routines at runtime using shared objects called Dynamic-Link Libraries (DLLs). These loadable objects are then mapped into the address space of a running process. While this is often considered an important feature of modern operating systems, a malicious adversary can exploit this dynamic linking to manipulate the functionality of a process. In certain circumstances listing the DLLs that are loaded by running processes may provide an indication of suspicious activity [47]. During a live data collec-

tion, this information would typically be generated by running `ListDLLs` [40] on the system under investigation [10, 29].

- *Open Connections* Another area of interest for digital investigators is the current network connections that are established with the system under investigation. For example, a malicious adversary may be maintaining a backdoor into the system or be using the system as a stepping stone to other systems. Thus, open connection information can provide useful context about the state of the machine and be helpful for determining other machines with which the system was communicating. During a live data collection, this information about the current network connections would typically be obtained by running `netstat` on the system under investigation [23, 10, 29].

We must take this opportunity to remind you that these are only Survey tools and are not intended to provide advanced or exhaustive analysis. As with their live collection counterparts, they only have the potential to provide quickly obtainable investigative leads. Later in the paper, we will even discuss techniques an adversary may use to subvert these types of survey tools. On the other hand, by collecting volatile memory and becoming comfortable with this type of analysis, we create a base for more advanced tools and techniques that are much more difficult to subvert.

6 Challenges in Digital Forensics

In this section, we intend to address how integrating volatile memory analysis into the Survey Phase of the digital investigation process has the potential to help address some of the challenges currently facing the digital investigation community. In particular, we will focus on how investigators deal with the ever increasing volume of evidence and the increased usage of encryption technologies.

6.1 Volume of Evidence

One of the major challenges facing the digital investigation community is dealing with the increasing large volume of evidence associated with incidents [27, 38]. This often overwhelming volume of evidence can be attributed to a number of different causes. Modern computers are used to store an ever increasing amount of data. These computers are interconnected via high speed links allowing them to share large amounts of data. The number of incidents involving computers is growing as is the number of computers associated with each incident [18]. As a result, investigators are often overwhelmed by the amount of evidence that needs to be collected, stored, and analyzed. This is also coupled with the growing complexity of the analysis techniques that are being performed. Thus, it is understandable why an investigator inundated with filesystem images and traffic dumps would be apprehensive at the thought of a yet another data store, the volatile memory image, that needs to be analyzed.

One strategy people have suggested for addressing the volume of evidence problem is to start their investigation by using methods that attempt to target evidential leads. Unlike previous context-free methods which focused on using statistical clustering techniques to enumerate suspicious outliers [9, 6], we suggest leveraging the context from volatile memory to focus these initial efforts. We refer to this techniques as volatile targeting. By leveraging volatile memory analysis techniques during the Survey Phase of the investigation process, it becomes possible to leverage the contextual artifacts that can be extracted from volatile memory to focus the investigation leads for later phases of the process. This extra context can facilitate the investigators initial triage and allow them to quickly obtain time-sensitive data while it still has the potential for evidentiary value. Meanwhile, we are protecting evidential integrity of the volatile memory image so that it can be sent back to the lab for a more in-depth and thorough analysis.

Volatile targeting can prove extremely useful when dealing with large volumes of filesystem images or network traffic. For example, it is common practice to perform a forensic duplication of at least one or more hard drives during the digital investigation process, regardless of whether it is an intrusion incident or a contraband incident. As previously mentioned, these drives could contain a voluminous amount of data. In the example of the intrusion incident, the digital investigator can begin by targeting those executables that are currently loaded into memory, the DLLs that were loaded by those processes, and the files and directories the processes were accessing. On the other hand, if an investigator has detected a suspicious file during the filesystem analysis they can go back to the volatile memory image and determine if a process was accessing that file when the image was taken, and if a process was, what other files it was accessing as well. In some circumstances, the information from volatile memory can even be used to determine what types of files the intrusive code or adversary had been interested in. In the example of a contraband search, it can help the investigator target those files and directories that were open when the drive was seized or recently accessed in the moments before the drive was seized.

There are also a number of digital investigators who are often faced with large data stores of packet captures. These are generally the packet captures of a large number of machines that were collected for an extended period of time (e.g., days, weeks, months). During this period, there was an intrusion incident and the investigator is asked to determine the extent of the damage. By capturing volatile memory of suspected systems and leveraging volatile memory analysis in the Survey Phase of the investigation, the investigator can begin by targeting the volatile leads. For example, if the investigator found a suspicious process running on the system, they could target traffic flowing to any sockets the process had open or connections that were established. On the other hand, if the investigator knew that a particular traffic flow was suspicious, they could immediately map it back to the process and find any other open connections, files, or sockets associated with that process. There also may be times when an investigator is using volatile targeting techniques and discovers an encrypted network flow or an encrypted file. In the next section, we will discuss how an investigator can use volatile memory analysis to help address this challenge as well.

6.2 Encryption

An area of growing concern for digital investigators is the increasing use of encryption technologies [13, 11]. While encryption is not a new concern for cyber investigators, the growing ubiquity, accessibility, and usability of these technologies has the potential to seriously disrupt conventional methods of digital investigation. We have found the two areas of particular concern are encrypted network traffic and encrypted filesystems. Assuming strong encryption algorithms, the conventional methods investigators have to deal with this technology including coercion, brute force, and ephemeral files are time consuming and often have low return on investment. We intend to emphasize how including volatile memory analysis in the digital investigation process has the potential to help address some of these issues.

By leveraging volatile memory analysis, it is often possible to extract a number of volatile artifacts that would typically be lost, including unencrypted content, keying material, and sometimes passwords. Often the ability to extract each of these types of data depends on implementation details of the particular technology and the cryptographic properties the applications are designed to support. In this section, we will focus on a case study of a popular software based filesystem encryption technology that supports “On the fly encryption” (OTFE), and we will demonstrate how volatile memory analysis can be used to extract the keying material from memory enabling us to mount the disk offline.

The challenge of filesystems that support OTFE has recently been garnering a substantial amount of attention from digital investigators [31, 48]. This concern has been heightened with Microsoft’s incorporation of BitLocker Drive Encryption technology in upcoming versions of Microsoft Vista [46]. A data protection feature that attempts to leverage the increased deployment of Trusted Platform Modules (TPM 1.2) to protect user data. Microsoft claims the motivation behind this technology is to prevent offline ac-

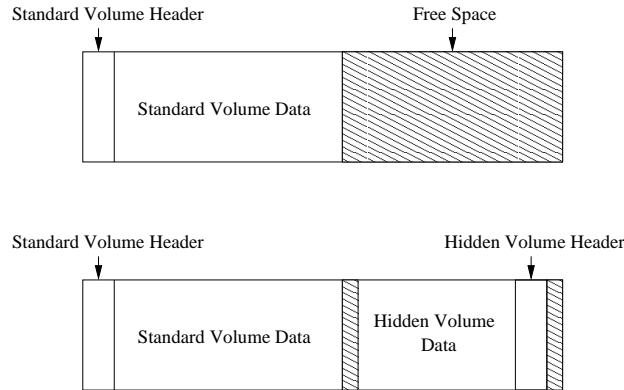


Figure 2: Examples of TrueCrypt volumes format [1]

cess to stored files on protected volumes, thereby helping to prevent data theft from stolen or improperly handled systems [45]. Another related technology that concerns digital investigators is TrueCrypt, a free open-source cross-platform disk encryption software that has a number of advanced features and a usable interface [1, 48]. Despite the benefits of these technologies for data protection, they have also proven to be an extremely effective method of hiding incriminating evidence [33].

In order to demonstrate the importance of incorporating volatile memory analysis into the digital investigation process and demonstrate the details of this technology, we decided to explore TrueCrypt 4.2a, a popular open source filesystem encryption utility that supports a number of advanced features [1]. TrueCrypt is a cross-platform system that supports “On the Fly Encryption” (OTFE), whereby an encrypted container can be mounted and transparently accessed. Both encryption and decryption are performed in RAM and unencrypted data is never stored on disk (not including swap). The TrueCrypt website clearly states that, while performing these operations, cached passwords, encryption keys, and sensitive content can be found in RAM [1]. We intend to demonstrate that, by integrating volatile memory analysis into the digital investigation process, it becomes extremely easy to both detect the usage of TrueCrypt and extract the keying material to support later phases of investigation.

In this example, we will focus our attention on TrueCrypt’s file-hosted volumes, which are files that contain encrypted virtual disk drives [1]. While on the host filesystem, these files contain no “magic” numbers or identifiable characteristics but appear as large files of random data. As seen in Figure 2, in the standard configuration the addressable sectors of a file-hosted volume are divided into three main areas: the standard volume header, the standard volume data, and free space. The standard volume header is the first 512 bytes of the file-hosted volume and is encrypted using the user generated password. The encrypted standard volume header also contains the master key which can be used to decrypt the standard volume data. Finally, the free space of the standard configuration is generally filled with random data. TrueCrypt can also be configured to support hidden volumes in which the addressable sectors of the free space are divided into a hidden volume header and data. This configuration can also be seen in Figure 2. This allows the user to continue to hide data even if they are forced to reveal a password for the standard data volume. In this example, we will focus on the standard configuration since once they are mounted the techniques for extracting the keying material are similar. For the sake of the example, we will also assume that passwords are not being cached and key files are not being used.

When a TrueCrypt volume is mounted, the standard volume header is read into volatile memory and the user is prompted for a passphrase. Using the salt extracted from the first 64 bytes of the volume header and the user supplied password, the mounter attempts to determine the encryption characteristics of the volume using a process of trial and error. This involves trying to decrypt the header using each of the possible

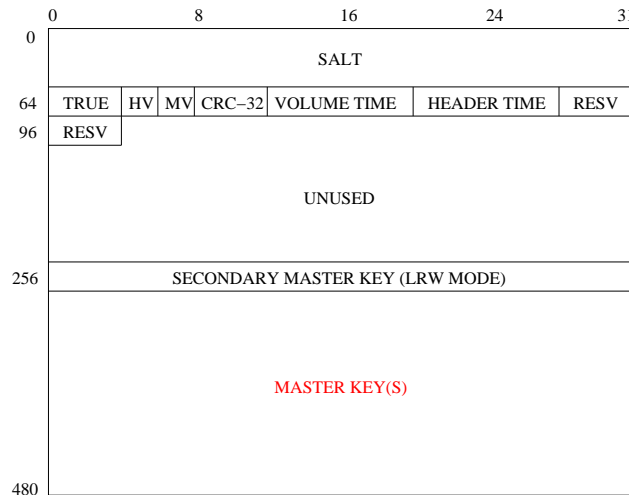


Figure 3: TrueCrypt standard volume header format

encryption combinations. Upon success, the magic value, the ASCII string “TRUE,” will be found in first four bytes of the header and a CRC-32 checksum of a subsection of decrypted data will match its stored value. The format of the decrypted standard volume header can be seen in Figure 3. As seen in the figure, once the header is decrypted correctly, it is then possible to extract the master key, which can be used to decrypt the volume sectors.

At this point, the details become operating system dependent. However, in most cases the high level principles are the same. As previously mentioned, TrueCrypt treats file-hosted volumes as virtual disk drives. As a result, the processes on the system are able to interface with the disk as if it is an ordinary block device that transparently supports OTFE encryption. This means that writes to this random access storage device will be encrypted and reads will be decrypted. On Linux, TrueCrypt is implemented as a Device Mapper target, which means that it leverages the logical volume management support provided by the Device Mapper to treat the standard volume data as a virtual block device. Once the master key is extracted from the standard volume header, the mounter relies on the TrueCrypt target constructor to properly initialize the Device Mapper target.

In virtual memory, each Device Mapper target is represented by a **dm_target** object. The **dm_target** object has a *private* member, which is a void pointer used to store local context information about the particular target. As the TrueCrypt target is instantiated, the *private* member is initialized to the local context of the TrueCrypt target. This context is stored in the **target_ctx** object, which includes information such as the targets volume path, virtual memory page pool, flags, times, and a pointer to its cryptographic context, *ci*. The cryptographic context object, **CRYPTO.INFO**, is used to store the cryptographic information extracted from the standard volume header including the *master_key*. This structure also includes information about the algorithms being used, the mode of operation, the initialization vector, and the hidden volume state. While the TrueCrypt volume is mounted, this information is stored in volatile memory to allow encryption and decryption to be performed transparently.

By integrating volatile memory analysis in the digital investigation process, it becomes possible to not only determine if TrueCrypt is being utilized, but also extract the memory resident keying material. However, we need a method to automate the process of finding this information. As with non-virtualized block devices, the Linux operating system uses major and minor numbers to determine how to interface with a particular device. In the case of a Device Mapper, the major number is used to distinguish the device as a Device Mapper target and the minor number allows Device Mapper to quickly determine the specific device.

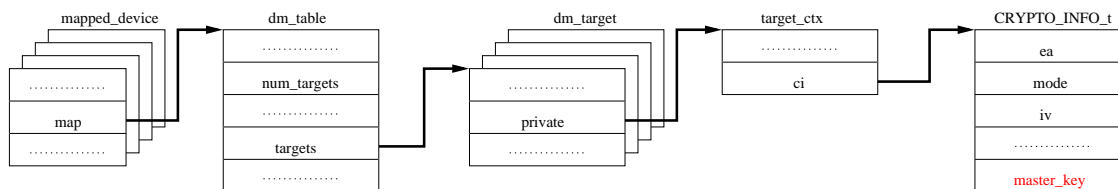


Figure 4: A graphical representation of the relationships between the data structures used to extract the keying material

In order to perform these lookups quickly, the Device Mapper kernel module relies on an IDR structure to keep track of allocated minor numbers. The IDR library uses a radix tree like structure as a sparse array used for mapping integer IDs to pointers without requiring fixed-size tables [21]. In this structure, the minor number is the integer ID that is mapped to a pointer. When in use, this pointer references a **mapped_device**. In order for the Device Mapper to quickly perform lookups it leverages the `_minor_idr` symbol.

Using the symbols exported by the Device Mapper kernel module, we are able to find the `_minor_idr` symbol in memory. The value stored in this symbol is a pointer to a kernel object of type **idr**. Using the `top` member of this object, we are able to find the root of the IDR tree, which is object of type **idr_layer**. The `layers` member of the **idr_layer** object can be used to find the upper bound on the number of minor numbers that are being stored in the IDR tree. By incrementing through these ids until we reach the upper bound, we are able to find the ones that are populated with pointers to **mapped_device** objects. Each **mapped_device** object in turn maintains a pointer to **dm_table** object, which contains the function table mapping for this device. The **dm_table** object also contains a `targets` member, which points to the Device Mapper target that makes use of this particular function table mapping. As previously mentioned, Device Mapper targets are instantiated in memory as **dm_target** objects that stores the TrueCrypt targets local context information, **target_ctx**. Using the `ci` member of this object, we can find the `master_key` member of the **CRYPTO_INFO** object. Once this information is found in the image, it can be extracted for later use. With a few minor changes to the mounter, we can use the extracted cryptographic information to mount the volume offline without the password. A graphical representation of the relationships between these data structures can be seen in Figure 4.

The purpose of this example was to demonstrate the feasibility of extracting keying material from a volatile memory image and emphasize the importance of including volatile memory analysis in your digital investigation process. Another point of this exercise was to introduce the reader to some of the important concepts necessary for performing this type of analysis. On the other hand, it is important to give the caveat that there are a number of different aspects that have the potential to make this process substantially more difficult. In order to facilitate discussion, we chose a easy example in which we have access to the application's source code and the source code to the kernel. In many cases, especially in when dealing with Windows or malicious code, we will not have the luxury of the source code or symbols. But, if you have an understanding of the functionality provided by the application and how that depends on the underlying operating system, you can often find how the cryptographic context and keys are being stored. For example, if you understand how device drivers store auxiliary information, the process is very similar once those objects can be located in the volatile memory image. There are also many characteristics of this application and its cryptographic properties that make the example amenable to volatile memory analysis. For example, the TrueCrypt application allows you interact transparently with virtual disk drive while providing OTFE. Performance is also a big concern so, until the encryption is done in hardware, these types of applications can ill afford to constantly poll a static hardware device like a Trusted Platform Module (TPM). As a result, it stores the keying material in volatile memory at least as long as the disk is mounted and the master key

for the standard volume data is rarely changed. On the other hand, if you consider some network protocols that employ cryptographic principles like perfect forward secrecy and that update session keys regularly, you may only be able to decrypt a limited subset of the data. While there are a number of challenges facing the volatile memory analysis community, recent incidents have demonstrated the importance of including volatile memory analysis in your digital investigation process [33].

7 Challenges and Anti-Forensics

In addition to the lack of legal precedent associated with volatile memory analysis, there are still a number of technical challenges and limitations as well. The first issue is temporal proximity. This relates to the fact that volatile memory analysis will only be able to detect those objects that are active on the system or inactive objects whose resources have not been reclaimed for other use. As a result, it will be most effective at detecting persistent changes to the system or in circumstances where acquisition is performed in close temporal proximity to the incident. Another challenge is that in many situations the processor cannot be halted in order to perform a volatile memory acquisition. As a result, the image collected is similar to a blurry snapshot of a moving object. Thus, there may be semantic inconsistencies between data objects that need to be accounted for during analysis.

Another major challenge for the volatile memory analysis community is to consider the malicious adversary and anti-forensics. For example, most of the publicly available acquisition techniques for both volatile and non-volatile data either depend on the assumption that the operating system or some hardware component of the system can be trusted [8, 15, 16]. Recently, people have begun to attack these mechanisms and corrupt the acquisition process [12, 22]. This is being addressed both by virtualization and new hardware acquisition support that allows acquisition to be performed by interfacing directly with the hardware.

It is also important to emphasize that volatile memory analysis needs to be integrated into both the Search and Reconstruction Phases of the digital investigation process as well. Many of the volatile memory analysis tools and techniques designed for the Survey Phase of the digital investigation process [19, 44, 4], including Volatools, are susceptible to a malicious adversary. The first form of attacks relates to creating false negatives or hiding. Using some very simple techniques, it is possible to hide from both list-walking and linear scanning techniques. For example, many of the list-walking techniques are susceptible to Direct Object Manipulation (DOM) attacks similar to those presented by Butler [20]. As a result, simply unlinking the **EPROCESS** object from the list of *ActiveProcessLinks* will often allow it to hide from list-walkers.

In response to these forms of attack, people have suggested linear scanning techniques [19, 44]. They often perform a linear scan of physical memory in search of objects that match what they consider “reliable patterns” [44]. Unfortunately, a reliable pattern may rely on characteristics of an object that are not essential. As the filesystem community has realized [5], essential and nonessential members of an object can only be judged with respect to a particular purpose. For example, the *Type* and *Size* members of the **DISPATCHER_HEADER** object, used by the synchronization mechanisms, may help provide a reliable pattern for an **EPROCESS** object but are not essential for a process to be scheduled. Thus, these values can be changed and the process will not be seen by the majority of the linear scanning tools. By using simple techniques, it is possible to hide from both list-walking and linear scanning Survey tools collectively. Based on these same notions of essential and non-essential, there are also a number of techniques that can be used to the sanitize the non-essential members of an object in memory [47].

Another area of concern is the susceptibility of these tools to false positives or decoys. It is possible for a malicious adversary to dramatically increase the noise to signal ratio and keep the digital investigator busy. Many of the linear-scanning tools available perform a context insensitive scan of the physical address space in search of their reliable patterns. Unfortunately, using this method makes it extremely easy for a malicious adversary to create “life-like” decoys. In fact, it is even possible to fool many of these tools by

creating an decoy operating system, since the majority of the available tools do not acquire or make use of CPU register values. Unfortunately, many of the aforementioned attacks will not be addressable within the Survey Phase of the investigation process and will require advanced analysis in both the Search and Reconstruction Phases.

8 Future Work

As it is obvious to see, a substantial amount of work still needs to be done in the area of volatile memory analysis. Our work currently focuses on building new tools and techniques for integrating volatile memory analysis into the more advanced Search and Reconstruction Phases of the digital investigation process. This work is primarily being done by the FATKit project, which attempts to focus on the goals of automation, reuse, and visualization [24, 34, 47]. We are also collaborating with other groups to improve the correlation with other data stores and update current forensics resources to support the needs of the volatile memory analysis community. Finally, we are also working with, and avidly seeking, practitioners who provide the useful feedback and testing that drives our work.

9 Conclusion

Despite the increased attention given to volatile memory analysis, many investigators still feel they are too overburdened to analyze another “substance,” volatile memory. The purpose of this paper was to emphasize that volatile memory is a critical component of the digital crime scene and, as such, should also be integrated into every phase of the digital investigation process used to analyze that crime scene. In particular, this paper focused on the Survey Phase of that process and introduced a tool set, Volatools, that can be used to address some of the limitations associated with popular tools and techniques for live response. Finally, the paper discussed how leveraging the context provided by volatile memory analysis during the digital investigation process has the potential to help address some of the main challenges facing the digital forensics community. We hope people will walk away with the impetus to start collecting this valuable piece of evidence.

10 Acknowledgments

The authors would like to thank those who contributed to and reviewed this white paper. We would also like to thank MISSL, DS², Komoku, and all the monkeys across the world.

References

- [1] TrueCrypt, January 2007. Available at: <http://www.truecrypt.org/>.
- [2] Andreas Schuster. International Forensics Blog, June 2006. Available at: <http://computer.forensikblog.de>.
- [3] Chris Betz. *memparser*, 2005. Available at: <http://www.dfrws.org/2005/challenge/memparser.html>.
- [4] Mariusz Burdach. *Windows Memory Forensic Toolkit (WMFT)*, 2006. Available at: <http://forensic.seccure.net/>.
- [5] Brian Carrier. *File System Forensic Analysis*. Addison Wesley, 2005.

- [6] Brian Carrier and Blake Matheny. Methods for Cluster-Based Incident Detection . In *IEEE Information Assurance Workshop (IAW)*, Charlotte, NC, USA, April 2004.
- [7] Brian Carrier and Eugene H. Spafford. Getting Physical with the Digital Investigation Process. *International Journal of Digital Evidence (IJDE)*, 2(2), Fall 2003.
- [8] Brian D. Carrier and Joe Grand. A Hardware-Based Memory Acquisition Procedure for Digital Investigations. *Journal of Digital Investigations*, 1(1), 2004.
- [9] Brian D. Carrier and Eugene H. Spafford. Automated Digital Evidence Target Definition Using Outlier Analysis and Existing Evidence. In *Proceedings of the 2005 Digital Forensic Research Workshop (DFRWS)*, 2005.
- [10] Harlan Carvey. *Windows Forensics and Incident Recovery*. Addison Wesley, 2005.
- [11] Eoghan Casey. Practical Approaches to Recovering Encrypted Digital Evidence. In *Proceedings of the 2002 Digital Forensic Research Workshop (DFRWS)*, 2002.
- [12] Darren Bilby. Low Down and Dirty: Anti-forensic Rootkits, October 2006. Available at: <http://www.blackhat.com/presentations/bh-jp-06/BH-JP-06-Bilby-up.pdf>.
- [13] Dorothy E. Denning and William E. Baugh Jr. Encryption and Evolving Technologies as Tools of Organized Crime and Terrorism, May 1997. Available at: <http://www.cosc.georgetown.edu/denning/crypto/oc-abs.html>.
- [14] Foundstone. Fport, 2002. Available at: <http://www.foundstone.com>.
- [15] George M. Garner. Forensic Acquisition Utilities, October 2006. Available at: <http://users.erols.com/gmgarner/forensics/>.
- [16] George M. Garner. KNTDD, 2006. Available at: <http://users.erols.com/gmgarner/KnTTools/>.
- [17] George M. Garner Jr. *kntlist*, 2005. Available at: <http://www.dfrws.org/2005/challenge/kntlist.html>.
- [18] III Golden G. Richard and Vassil Roussev. Next-generation digital forensics. *Commun. ACM*, 49(2):76–80, 2006.
- [19] Harlan Carvey. Windows Incident Response, June 2006. Available at: <http://windowsir.blogspot.com>.
- [20] Greg Hoglund and Jamie Butler. *Rootkits: Subverting the Windows Kernel*. Addison-Wesley, July 2005.
- [21] Jim Houston. Linux Kernel: idr.c, October 2002. Available at: linux-2.6/lib/idr.c.
- [22] Joanna Rutkowska. Beyond The CPU: Cheating Hardware Based RAM Forensics, February 2007. Available at: <http://blackhat.com/html/bh-dc-07/bh-dc-07-speakers.html#Rutkowska>.
- [23] Keith J. Jones, Richard Bejtlich, and Curtis W. Rose. *Real Digital Forensics*. Addison Wesley, 2005.
- [24] Nick L. Petroni Jr. and Aaron Walters. FATKit: The Forensic Analysis Toolkit, February 2006. Available at: <http://www.4tphi.net/fatkit/>.
- [25] Jesse Kornblum. Preservation of Fragile Digital Evidence by First Responders. In *Proceedings of the 2002 Digital Forensic Research Workshop (DFRWS)*, 2002.

- [26] Jesse Kornblum. Using Every Part of the Buffalo in Windows Memory Analysis. *Journal of Digital Investigations*, 5(1), 2007.
- [27] Ted Lindsey. Challenges in Digital Forensics. In *Proceedings of the 2006 Digital Forensic Research Workshop (DFRWS)*, 2006.
- [28] Kevin Mandia and Chris Prosise. *Incident Response: Investigating Computer Crime*. McGraw-Hill Osborne Media, 2001.
- [29] Kevin Mandia, Chris Prosise, and Matt Pepe. *Incident Response and Computer Forensics*. McGraw-Hill Osborne Media, 2 edition, 2003.
- [30] Monty McDougal. Windows Forensic Toolchest, January 2007. Available at: <http://www.foolmoon.net/security/wft/>.
- [31] Jim Moeller. Windows Vista Forensic Jumpstart Part I and Part II, January 2007. DoD Cyber Crime Conference 2007.
- [32] NIJ Technical Working Group for Electronic Crime Scene Investigation. *Electronic Crime Scene Investigation: A Guide for First Responders*, July 2001.
- [33] David Pallister. Three guilty of identity fraud which netted millions, December 2006. Available at: <http://www.guardian.co.uk/crime/article/0,,1961441,00.html>.
- [34] Nick Petroni, Aaron Walters, Tim Fraser, and William Arbaugh. FATKit: A Framework for the Extraction and Analysis of Digital Forensic Data from Volatile System Memory. *Journal of Digital Investigations*, 3(4), 2006.
- [35] Python Software Foundation. *The Python programming language*, 2006. Available at: <http://www.python.org>.
- [36] Mark Reith, Clint Carr, and Gregg Gunsch. An Examination of Digital Forensics Models. *International Journal of Digital Evidence (IJDE)*, 1(3), Fall 2002.
- [37] Marc Rogers, Jim Goldman, Rick Mislán, Tim Wedge, and Steve DeBrotá. Computer Forensics Field Triage Process Model. In *2006 Conference on Digital Forensics, Security and Law*, Arlington, VA, 2006.
- [38] Vassil Roussev and Golden G. Richard III. Breaking the Performance Wall: The Case for Distributed Digital Forensics. In *Proceedings of the 2004 Digital Forensic Research Workshop (DFRWS)*, 2004.
- [39] Mark Russinovich. Handle v3.20, November 2006. Available at: <http://www.microsoft.com/technet/sysinternals/ProcessesAndThreads/Handle.mspix>.
- [40] Mark Russinovich. ListDLLs v2.25, November 2006. Available at: <http://www.microsoft.com/technet/sysinternals/ProcessesAndThreads/ListDlls.mspix>.
- [41] Mark Russinovich. PsList v1.28, December 2006. Available at: <http://www.microsoft.com/technet/sysinternals/utilities/pslist.mspix>.
- [42] Mark Russinovich and Bryce Cogswell. FileMon for Windows v7.04, November 2006. Available at: <http://www.microsoft.com/technet/sysinternals/utilities/filemon.mspix>.

- [43] Andreas Schuster. Pool Allocations as an Information Source in Windows Memory Forensics. In *International Conference on IT-Incident Management & IT-Forensics*, Stuttgart, Germany, 2006.
- [44] Andreas Schuster. Searching for Processes and Threads in Microsoft Windows Memory Dumps. In *Proceedings of the 2006 Digital Forensic Research Workshop (DFRWS)*, 2006.
- [45] Robert Ellis Smith. Laptop Hall Of Shame, September 2006. Available at: http://www.forbes.com/columnists/2006/09/06/laptops-hall-of-shame-cx_res_0907laptops.html.
- [46] Microsoft TechNet. BitLocker Drive Encryption: Technical Overview, April 2006. Available at: <http://technet.microsoft.com/en-us/windowsvista/aa906017.aspx>.
- [47] AAaron Walters. FATKit: Detecting Malicious Library Injection and Upping the Anti, July 2006. Available at: http://www.4tphi.net/fatkit/papers/fatkit_dll_rc3.pdf.
- [48] Ronald Weiss. Forensic Analysis of Open Source Disk Encryption Tools, January 2007. DoD Cyber Crime Conference 2007.