CERIAS Tech Report 2017-5
Deceptive Memory Systems
by Christopher N. Gutierrez
Center for Education and Research
Information Assurance and Security
Purdue University, West Lafayette, IN 47907-2086

DECEPTIVE MEMORY SYSTEMS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Christopher N. Gutierrez

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2017

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Eugene H. Spafford, Co-Chair
Department of Computer Science
Dr. Saurabh Bagchi, Co-Chair
Department of Computer Science
Dr. Dongyan Xu
Department of Computer Science
Dr. Mathias Payer
Department of Computer Science

Approved by:

Dr. Voicu Popescu by Dr. William J. Gorman
Head of the Graduate Program

This work is dedicated to my wife, Gina. Thank you for all of your love and support. The moon awaits us.

ACKNOWLEDGMENTS

I would like to thank Professors Eugene Spafford and Saurabh Bagchi for their guidance, support, and advice throughout my time at Purdue. Both have been instrumental in my development as a computer scientist, and I am forever grateful. I would also like to thank the Center for Education and Research in Information Assurance and Security (CERIAS) for fostering a multidisciplinary security culture in which I had the privilege to be part of. Special thanks to Adam Hammer and Ronald Castongia for their technical support and Thomas Yurek for his programming assistance for the experimental evaluation. I am grateful for the valuable feedback provided by the members of my thesis committee, Professor Dongyen Xu, and Professor Mathias Payer. I am thankful for the financial support provided by the National Science Foundation, under award number 1548114.

Personal thanks to my Mom, Dad, and Grandma for instilling me the value of hard work and supporting my passion for computing at an early age. Thank you for allowing me to forgo a year of birthday and Christmas gifts for my first computer. A heartfelt thank you to my wife, Gina, for her unwavering support throughout my time as a graduate student. Thanks to Jeff Avery, Paul Wood, Mohammed Almeshekah, Oyindamola Oluwatimi, and Abram Magner for their valuable feedback and support. Thank you to all of my colleagues in the Dependable Computing Systems Laboratory for their comments on the preliminary results. To all of my friends and family who have shown me nothing but love, I thank you. Finally, huge thanks to my cat Buntu for making me laugh and smile every day.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Gutierrez, Christopher N. Ph.D., Purdue University, December 2017. Deceptive Memory Systems. Major Professors: Eugene H. Spafford and Saurabh Bagchi.

Unauthorized data destruction results in a loss of digital information and services, a devastating issue for society and commerce that rely on the availability and integrity of such systems. Remote adversaries who seek to destroy or alter digital information persistently study the protection mechanisms and craft attacks that circumvent defense mechanisms such as data back-up or recovery.

This dissertation evaluates the use of deception to enhance the preservation of data under the threat of unauthorized data destruction attacks. The motivation for the proposed solution is two-fold. (i) An honest and consistent view of the preservation mechanisms are observable and often controlled from within the system under protection, allowing the adversary to identify an appropriate attack for the given system. (ii) The adversary relies on some underlying I/O system to facilitate destruction and assumes that the components operate according to a confirmation bias based on prior interactions with similar systems. A deceptive memory system (DecMS) masks the presence of data preservation and mimics a system according to the adversary's confirmation bias.

Two proofs of concepts and several destructive threat instances evaluate the feasibility of a DecMS. The first proof of concept, DecMS-Kernel, uses rootkits' stealth mechanisms to mask the presence of DecMS and impede potential destructive writes to enable preservation of data before destruction. The experimental results show that DecMS is effective against two common secure delete tools and an application that mimics crypto ransomware methods.

Based on the results of DecMS-Kernel, several improvements are incorporated into a DecMS that uses virtual machine introspection. DecMS-VMI places the preservation mechanism out of reach from the system under protection. The virtual machine under protection does not undergo any changes or need additional software to support the deception, thus improving stealthiness. The results for DecMS-VMI demonstrate the ability to preserve data under a wide range of destructive methods: 13 different secure delete methods, four wiper malware, and one timestamp fabrication tool. Under both prototype systems, all of the detected data under destruction is successfully preserved. The overall results indicate that it is feasible to create deceptive systems to enhance data preservation methods on interactive computing systems.

1 INTRODUCTION

In the information age, digital data is a critical asset. When adversaries destroy digital information, it is challenging to recover it without a copy stored elsewhere. Physical security can prevent adversaries from physically destroying storage devices. However, it is possible to create malicious software to destroy data on a persistent storage medium[1]. Early examples date back to 1988 with programs such as the Jerusalem Virus, which destroyed files on Fridays that fall on the 13th of each month [1]. Such malware is referred to as *wiper malware* because it destroys files and makes them unrecoverable. In the present day, loss of data, through an adversarial attack on data availability, utility, and integrity [2], is more than a nuisance and can cause substantial damage.

In 2017, high profile Wiper Malware, such as Shamoon and Stonedrill, have been responsible for the destruction of files on many tens of thousands of computer systems [3] [4]. In August 2012, the Shamoon Wiper Malware destroyed critical system files and digital documents on about 30,000 corporate computers at Saudi Aramco [3]. The US Secretary of Defense at the time, Leon E. Panetta, stated that the affected machines "were rendered useless" and that "the Shamoon virus was probably the most destructive attack that the private sector has seen to date" [5]. Wiper malware target critical systems, such as those connected to the energy sector to cause power outages [6].

Once an attacker gains access to a system, files are at risk for unauthorized data destruction. Frequently modified user files, such as documents, images, or spreadsheets, are targets for data destruction attacks. Adversaries may also target files that require effort to replace, such as operating system files that are necessary for system

[1] A persistent storage medium is a device to store information that persists between machine reboot cycles.

stability and operation [6]. Destroying OS kernel files causes a system to become unbootable, requiring the system administrators to rebuild the system.

Common among all software data destruction attacks is the reliance on underlying system components to facilitate I/O requests. Both adversaries and users rely on several layers of abstraction that translates a semantic I/O request (such as writing to a named file on a file system) into machine code that controls the hardware components of a computing system. Both users and adversaries assume that the underlying layers work correctly to handle I/O requests. The system defenders can use the adversary's confirmation bias [7] to deceive.

In particular, this work proposes and evaluates Deceptive Memory Systems (DecMS) to enhance the protection of digital assets located on file systems under the threat of unauthorized data destruction. DecMS relies on existing security monitoring techniques which observe suspicious activity. Rather than denying suspicious writes that appear to be destructive, a DecMS system attempts to deceive an attacker into believing her destructive actions are successful while providing the system defenders with valuable information about the attacker.

Prior work in deception for defending computing systems relies upon isolated systems to lure attackers away from real systems. However, there are several advantages to deceive an adversary [7] who is conducting unauthorized data destruction on real systems. If the defenders can preserve the data under destruction without the adversary's knowledge, the defenders gain information about the adversary's targets and motivation. The defender observes the adversary from the context of a real system rather than a honeypot system that may contain synthetic data, users, and context. System defenders can craft deception into production systems to fool an adversary into incorrectly believing that destructive actions are damaging real protection systems.

Some data destruction attacks bring systems offline as quickly as possible. Another viable deceptive strategy is to impede the speed of destructive actions. Forcing the adversary to slow down provides a strategic advantage, allowing the system defenders

more time to react appropriately to an adversary. The system defenders may choose to impede the attacker while preserving the data under destruction in an isolated location.

Existing methods to recover from or mitigate unauthorized data destruction fall short in several ways. Data replication, such as a complete replica of a storage device, is taken periodically. An adversary can destroy data between backups, causing a loss of information. Some file systems track all changes and allow users or system administrators to recover from an accidental loss of data. However, several of these systems are not resilient to adversaries who disable the backup schemes before destroying data. Other techniques attempt to recover data after destruction occurs, which are time-consuming, expensive, or unreliable in retrieving the data [8]. Integrity monitoring may be difficult to maintain on user files that frequently change and are targets for destructive adversaries. Access control also fails to protect against unauthorized data destruction. The adversary may gain a high level of privilege before executing the destruction or may target user-level files, which may be difficult to replace.

The deceptive strategies explored in this dissertation are not alternatives to the existing data preservation and security monitoring techniques, but instead, enhance existing techniques through the use of deception. The system defenders may choose to deny an adversary's data destruction action or may use other data preservation strategies, such as periodic backups of user files. However, this dissertation focuses on the design and use of deception to enhance protection against data destruction attacks.

1.1 Thesis Statement

It is feasible to use deception to enhance the preservation of digital assets against unauthorized data destruction.

The dissertation demonstrates the feasibility by designing and implementing a proof of concept system that monitors for destructive writes, preserve the data un-

der destruction, and presenting a false reality to the adversary that indicates her destructive actions are successful.

1.2 DecMS Goals

Throughout this thesis document, the experiments are designed to demonstrate the following enhancements:

DecMS Preservation Goal It is possible to preserve digital assets when destruction occurs;

DecMS Impedance Goal It is possible to impede data destruction to provide time for defenses mechanisms to adjust;

DecMS Reduction Goal It is possible to reduce the effectiveness of data destruction attacks;

DecMS Identification Goal It is possible to gain information about adversary motivations by identifying targets for data destruction.

A successful proof of concept for any of the first three enhancement goals will demonstrate the feasibility of DecMS. The final goal, DecMS Identification Goal, is a secondary enhancement that will provide a benefit for system defenders. Evidence of feasibility for the enhancement goals is given for each proof of concept system considered.

1.2.1 Preserve

Rather than denying a potential destructive write, the action can appear successful from the adversary's perspective. The deception must persist throughout the attack to observe all of the targets of an adversary. However, the data under destruction must also be preserved to recover from an attack quickly. The preservation of data is deceptive and is an action taken at the end of the attacker's OODA loop but before

the attacker can observe if the destructive action is successful. The DecMS proof of concept preserves data under destruction and hides the existence of the preserve data.

1.2.2 Impede

Several attacks attempt to destroy data quickly to prevent system defenders reacting to data destruction. There are advantages to slowing the attacker down to allow the system defender time to reconfigure the system and mitigate the severity of an oncoming attack.

Impedance allows more time for the system defenders to orient, decide, and act to subsequent unauthorized data destruction attempts. During the unauthorized data destruction, the DecMS proof of concept system impedes the unauthorized writing of files to preserve the data under destruction without the adversary's knowledge. Impedance in the DecMS proof of concept system demonstrates the feasibility of slowing down a threat to allow the defender to observe and act on unauthorized data destruction.

1.2.3 Reduce

If an adversary is aware that the data destruction methods are not successful, then she may adjust her strategy. The defender's deception feeds false information to mislead the adversary into believing her data destruction is successful. The deception causes the adversary to incorrectly orient, decide, and act on subsequent unauthorized data destruction attempts, reducing the effectiveness of her techniques. In the DecMS proof of concept system, the feasibility of protecting against several data destruction attacks is demonstrated.

1.2.4 Identify

Rather than denying suspicious behavior, the defenders may choose to learn about an adversary by observing what the attacker is attempting to destroy. The configuration of the system deceives the attacker into believing her data destruction methods are useful while the defenders observe her actions. Identifying the targets of data destruction allows system defenders to learn about their adversaries and react more quickly to threats that follow a similar attack strategy. The information gained may help protect other systems more quickly than capturing samples and conducting a static or dynamic analysis. The motivation for the Identify Goal is that the portion of attackers who can quickly break into systems is much higher than the portion of defenders who can quickly discover breaches on their systems [9].

If the system suspects unauthorized data destruction for specific files, then the system can react accordingly. Deception for defense provides an advantage in the Observe, Orient, Decide, and Act loop (OODA) [7]. The deceptions may cause the adversary to act incorrectly or induce confusion to slow down the decision process. The Identify goal is relevant to the observation step in the OODA loop because the defenders gain threat intelligence regarding targeted files. The DecMS proof of concept system demonstrates feasibility by collecting the data assets that adversaries target for data destruction and runtime information such as the process that requests the destructive write.

1.3 Contributions

1. Chapter 3.2 describes the deceptive planning necessary to support a DecMS System. In particular, the deceptive benefits, risks, and potential impact of using a DecMS System are detailed. Several types of computing systems are potentially suitable for DecMS enhancement, and the impact on performance is detailed.

2. Chapter 4.1 details the various methods to integrate DecMS within a computing system. Various layers of abstraction, deceptive strategies, and a discussion on the advantages/disadvantages are provided.

3. Chapter 6.1 evaluates a DecMS system through the use of kernel modules to enhance system calls relating to I/O. The results indicate that it is possible to protect against data destruction and hide from the adversaries through the use of several stealth techniques.

4. Chapter 6.2 evaluates a DecMS System by enhancing virtual machine monitoring by observing for data destruction indicators. The proposed system can successfully protect against various data destruction techniques. The systems also demonstrate an effective attacker impedance to provide additional time for adjusting defense strategies.

2 BACKGROUND AND RELATED WORK

In 1965, Daley and Neumann [10] described a general purpose file system, which organizes files through a hierarchy of directories. References to files within the file system are done so through symbolic references rather than their physical location on the storage device. Directories are containers of files (or other directories), and a file is an ordered sequence of data elements (e.g., bits, words, and characters). An entry name references a file or directory. An entry name that points to a file is called a branch, which also contains additional information about a file. Branch information (henceforward referred to as metadata), consists of information about a file, such as the physical address (location) of the file on the storage medium, and the creation or modification time of the file, and other information about the state of the file.

Daley and Neumann's work inspired several file systems such as New Technology File System (NTFS), the File Allocation Table File System (FAT), Hierarchical File System (HFS), and Extended File System (ext), to name a few. All of the above file systems organize files through a hierarchy of directories, consist of files, and contain metadata.

This thesis considers the protection against unauthorized data destruction on digital assets stored file systems. Further, the scope of data destruction is limited to software mechanisms that render data unrecoverable and does not consider the physical destruction of data (e.g., pulverizing a hard drive). A device controller may provide a method to destroy (also referred to as data sanitization) the contents of the entire drive [11, 12]. However, initiating the data sanitization may be inaccessible adversaries because it may require physical access or access to the Basic Input-Output System (BIOS) or Unified Extensible Firmware Interface (UEFI) [11]. This thesis does not consider attackers who have physical access or access to data sanitization methods on persistent storage devices.

Software data destruction is the overwriting of a digital asset such that the digital asset is no longer recoverable. A digital asset is any information related to accessing the file (including symbolic references, file system data structures, or directories), file metadata, or the file itself. Unauthorized data destruction can be partial or complete such that it produces a loss of information either temporarily or permanently. There are a variety of techniques for unauthorized data destruction. For instance, a threat may overwrite data arbitrarily to make data assets unavailable or replace data with misleading information to trick users. In Section 2.2, a description of the threat space is given.

Before describing the threat space, a discussion of the data assets that need protection is necessary. As computing resources are bounded, some information may have priority for preservation over other data assets. The severity of unauthorized data destruction is dependent on the importance and effort necessary to replace or repair the file. Several files, such as software binaries, are easily replaced because of online software distribution. Binaries for widely deployed software are available through various software repositories, such as the Advance Package Tool (APT) for Debian Linux[1]. User-produced files are difficult to replace if no backup copy is available.

2.1 Digital Assets to Protect

Digital assets are any files within a persistent storage medium, organized by a file system, and accessed through an operating system via a set of system calls. A file system is an organized structure of files on the storage medium and consists of metadata, directories, and the files. Programming interfaces use a file system to translate the logical organization of the files (directory path and file name) to a physical location on the storage medium.

Unauthorized data destruction may target any of the above components. A file may become unavailable if the file system data structures are corrupted or destroyed.

[1] `https://wiki.debian.org/Apt`

Metadata, which contains information about a file, may also be a target of data destruction. File destruction may occur without modifying the metadata or file system, allowing the file to be accessible from the operating system but may produce unexpected results.

The effort to recover from data destruction is dependent on the preparation before destruction occurs. As storage space is finite, files are prioritized based on importance and ease of replacement if destruction occurs. Attackers also consider the ease of replacement when identifying digital assets for destruction.

For illustrative purposes and henceforward, threats target two categories of assets: Replaceable Asset (RA), and Non-Replaceable Asset (NRA). The definition of NRAs are assets that only exist within a given system and not replicated elsewhere. Examples of NRAs may include user created media files such as digital images and videos, or system log files. The definition of RAs are files that are replicated elsewhere and readily accessible to users or system administrators. Examples of RAs include files such as widely deployed system libraries or applications.

Note that RAs and NRAs are perceptions that are subject to change based on (mis)information available. An attacker may destroy a backup copy of a file, causing the system administrator to misperceive an RA. A file that a system administrator believes to be RA is in fact NRA. Likewise, a system administrator may mislead an attacker into believing that she is destroying NRAs but is wasting effort by destroying RAs. The latter example is explored in subsequent sections and is fundamental in deceiving a destructive adversary.

The perception of RAs and NRAs guides the strategies an attacker chooses in data destruction. The destruction of RAs requires less effort to recover compared to the destruction of NRAs. However, the attacker may decide to target RAs for other motivations, such as hiding the presence of files, compromising authenticity. Another possible motivation is to render the target system unusable by overwriting RAs that are necessary for system stability, jeopardizing the utility of the operating

system. Authenticity and utility are some security elements that unauthorized data destruction threaten. Other threats and definitions of security elements are below.

2.2 Threat Space

The Parkerian Hexad is a framework which describes security with six elements: Availability, Utility, Integrity, Authenticity, Confidentiality, and Possession, each of which are defined by Parker [2]. While data destruction, in conjunction with other malicious actions, can compromise any combination of the six elements of the Parkerian Hexad [2], this dissertation defines data destruction as the loss of at least one of the security elements, with emphasis on the necessary conditions.

The last two elements listed above, confidentiality and possession, are not necessary for unauthorized data destruction. A loss of Confidentiality requires more than data destruction. A breach of confidentiality requires an attacker to read confidential information. However, an attacker could destroy data without reading the data. A loss of Possession overlaps with a loss of the other four remaining security elements. As previously defined, data destruction overwrites data assets to produce a loss of information. Therefore, the information at the location where the attacker overwrites always results in a loss of Possession of that given information. The loss of possession, henceforward, is implicit if not otherwise mentioned.

For each of the remaining four security elements, a brief description or summary is given from "Toward a New Framework for Information Security?" [2]. Examples for each security element are provided and are formulated with minimal overlap to illustrate the threat boundaries of the data destruction. In practice, data destruction combines the loss of a combination of security elements, which is discussed in Section 2.3.

Data destruction causes a loss in availability when, at the minimum, the overwrite causes a *misplacement of information*. In Loss Scenario 1 [2], Parker illustrates a loss of availability, without suffering a loss to other security elements, by an adversary

who renames a file without notifying valid users. When the attacker causes a loss in availability for RAs such as drivers or kernel objects, a system may become unusable until replacing or repairing the given files. A loss of availability on a NRAs is overcome by methods to uncover or search for the misplaced file. Other examples include typical uses of a delete command that cause a file to be misplaced.

The loss of authenticity through unauthorized data destruction causes, at the minimum, *misleading information*. Data assets should remain consistent between valid use. Unauthorized changes to a file that replaces information are misleading because it violates the user's perception of the file. Misleading information may not cause a substantial loss in the other three security elements considered for data destruction, as illustrated in Parker's Loss Scenario 4 [2]. To clarify, overwriting portions of files with misleading information always causes a partial loss of availability and integrity. The overwritten parts of the file are no longer available. The data asset also suffers a loss of integrity because the missing portions leave the file in a state that is incomplete.

Replacing file metadata may produce misleading information, but the file itself could still be available, have integrity, and have utility. For NRAs, it may be difficult to recover the original data without replicating the original data. In practice, correlation analysis can help overcome misleading information. For instance, file timestamp metadata can be cross-checked with application logs to help clarify misleading timestamp information [13].

The loss of integrity occurs when, at the minimum, a destructive action produces *incomplete information*. Parker [2] illustrates in Scenario 3 that the other three security elements (authenticity, availability, and utility) may not suffer a loss when integrity is lost. For NRAs, it may be difficult to recover portions of the files without replications. Cryptographic checksums can help determine if a file is complete. However, the creation of the checksum must occur before an attacker alters the file. Further, the checksums must be in isolation from the attacker. Mechanisms such as like Tripwire [14] can help detect a loss of integrity and other security elements.

The loss of utility occurs when, at the minimum, *information is transformed then replaced*. Information transformation may have minimal impact on other security elements. Parker illustrated in Loss Scenario 2 [2], that it is possible to lose utility, through unauthorized encryption of files, without suffering a loss of other security elements. Some loss of utility is recoverable for NRA if the attacker uses a poorly implemented or weak encryption algorithm. For instance, an attacker may fail to adequately destroy the decryption key, allowing the system administrator to recover the key and decrypt the transformed data.

The recovery from unauthorized data destruction for NRAs is dependent on the specific techniques that the attackers use.

2.3 Data Destruction Methods

There are several data destruction methods to consider. The work presented considers software data destruction where the physical device is usable, but the data is no longer available to the user. The data destruction methods may overwrite the data within the file itself, the metadata associated with the file, or any symbolic information that references the physical location of the file on the storage medium.

2.3.1 Delete

The definition of delete is "To remove or obliterate a record or item of data, such as by overwriting data on disk or tape with new data or null characters" [15]. A distinction between removing a record and removing an item is necessary. A record is a reference that points to the location of a file within the file system. File deletion overwrites the symbolic references (i.e., records) to specific file points to the physical location on the storage medium.

The distinction is necessary. Deleting a file does not typically overwrite the contents of a file. A deletion typically overwrites the record that points to the location of the file on the physical media. "Modern file systems associate the deletion of a file

with the release of the storage related to that file" [16]. As the record that references the file is overwritten, the file is no longer accessible from the file system. The actual contents of the file exist within the storage medium but are inaccessible without special software that scans portions of the disks that are considered free.

Deleting a file may still cause issues for a user. The inability to find a file may cause system instability. For instance, removing shared libraries or the binaries necessary for booting the operating system may cause the system to be inaccessible until the deleted files are replace or "undeleted." The definition of undelete is to restore a file previously deleted [17]. If the deleted file is a NRA, then it may be necessary to use "undelete" tools to recover the deleted file, which is discussed in Section 2.5.3.

The effect of deletion is that a user who wishes to access the deleted file will require additional effort. Data deletion is an attack on the availability security element. The deletion causes a loss of availability of a file or may cause a system to become unavailable if critical system files are deleted.

2.3.2 Secure Delete

In addition to or an alternative to unlinking a file from the file system, secure delete overwrites the entire file so that the undelete tools cannot recover any portion of the file. Secure deletion is necessary to preserve privacy. Certain files may contain sensitive information, so they are securely deleted to ensure that the information within a file is no longer available. However, when combining secure delete with unauthorized data destruction, the results may be devastating.

To completely remove a file from a system, one must consider several other components of a system, such as the physical storage medium, data replication, caches, hardware/software redundancy, file system, or metadata [18]. The National Institute of Science (NIST) provides guidelines for secure deletion of data concerning the type of physical media in question [11].

The NIST recommendation includes overwriting portions of memory with random bits, several times, depending on the storage medium. Alternatively, secure data deletion could also write over portions of memory with a fixed pattern such as all zeros, all ones, or some predefined pattern. This approach achieves secure data deletion without physical access to the storage space and without destroying the physical medium. However, the NIST recommendations consider the type of physical medium to ensure the destruction of residual data that may be present. For example, for magnetic storage devices, NIST recommends that several rounds of random bits be overwritten to render recovery with magnetic analysis impossible [11]. Some algorithms for secure deletion are detailed by the United States Defense Security Service [19, Section 4.4] and Schneier [20].

Methods to securely delete data assets are typically available for users. However, the secure delete method may not overwrite all the possible locations of a data asset [18]. File system and data caches may complicate secure deletion if the goal is to remove data assets from a system completely. The attacker must over-write the logical storage location of the data object as well as temporary locations where the data object may reside. Section 2.5.3 briefly describes how file caches and other systems may help in the restoration of destroyed data assets.

Another method to securely delete data is to use a "free-space filling" service [12, 18] that periodically identifies unallocated portions of a storage device and overwrite the free space rendering the data within the free space unrecoverable. An attacker who wishes to destroy data only needs to delete (e.g., unlink from the file system) and wait, or initiate, the free-space filling tool.

The overall process is similar to both delete and secure delete action with a different order of execution. Usually, the secure delete over-write occurs first before deleting the file. The Free-space filling tools delete the file first and then overwrites the space that it occupies. The difference between space filling tools and secure delete is delete-then-wipe (disk-filling tools) as opposed to a wipe-then-delete (secure delete). The minor distinction, henceforward, is ignored and will both be referred

to as "secure delete" as it has negligible impact in designing defense and deception systems to protect against unauthorized data destruction.

Secure delete causes a loss of availability, utility, and integrity. A securely deleted file is no longer available because the file is unlinked from the file system. Further, in addition to causing the file to go missing, the securely deleted file has no utility. Irrelevant information replaces the file. As the securely deleted file is not whole, the file suffers a loss of integrity. Secure delete, on its own, is not designed to mislead the user. However, replacing the file with information, rather than securely deleting, causes a loss of authenticity.

2.3.3 Data Replacement

An adversary may securely delete data objects that a forensic examiner can use as evidence of illegal activities. However, that is apparent when secure delete methods destroy files on a storage medium [21]. Rather than replacing the file with randomly distributed bits or other bit patterns, the attacker may replace the data.

Data replacement is advantageous for the attacker. Data replacement is more stealthy than typical secure delete methods, especially if the replaced data appears to be innocuous or indistinguishable from valid user data. Further, it may allow the attacker to place misleading information to confuse or deceive a user. For instance, file system metadata such as timestamps are critical for a forensic examiner to create a timeline of illegal activities. An adversary may change the timestamp metadata to obfuscate the order of events that took place or to cover evidence of file access or modification.

An attacker may use tools such as timestomp [2], an anti-forensic tool that replaces file system timestamp metadata with fabricated date and time entries. Timestamp fabrication can increase the difficulty for the forensic investigator. Data fabrication

[2] `https://www.offensive-security.com/metasploit-unleashed/timestomp/`

is an attempt to impede the examination process, thus wasting time, money, or personnel resources [22].

Data replacement causes a loss of authenticity at the minimum. Replacing timestamp metadata may mislead forensic analysts into believing a false chronologic order of file modification. Data replacement, when used with secure delete, may deceive the system defenders. If the replaced data is similar to free space on the disk, the defender may not be aware of the missing data assets. The data that is overwritten suffers a loss of availability, as mentioned in Section 2.2.

2.3.4 Transformation

Data transformation obscures data objects. The transformation may be benign, such as data compression to save storage space, or to hide from observation, such as encryption. When combining with data replacement and unauthorized data destruction, the results can lead to a loss of utility.

An example of unauthorized data destruction through transformation is crypto ransomware. Crypto ransomware [23, 24] replaces the data object with an encrypted version of the data. The encryption key is not available to the owner of the file, but rather, the key is in possession by an adversary. If the attacker is successful, the user is forced to pay extortion to recover the files. Other transformation techniques, such as steganography, can also transform data to produce a loss of utility. The attacker possesses the extraction key and may extort the owner to recover the original file. Files may also be transformed and replaced to appear to be corrupt, but in reality, are corrupted by an attacker. Data transformation can be combined with the other destruction techniques to produce a combination of a loss of security elements.

2.4 Data Destruction in Anti-Forensics

The adversary's ultimate goal of anti-forensics is to minimize the forensic *quality* and *quantity* of incriminating information [25]. An attacker may use anti-forensics to

induce doubt and confusion through the use of fabricated evidence [26,27], resulting in a loss of authenticity, integrity, or utility. Reducing the quantity of forensic evidence is a loss of availability or integrity.

Anti-forensics may also be designed to increase the difficulty of forensic analysis, relative to the resources available. Rather than replacing or altering evidence, an attacker may place additional false evidence. The difference is that the addition of evidence does not necessarily destroy data assets of interest [3]. The forensic investigator must then distinguish between fabricated and truthful evidence. This thesis effort does not consider cases of false evidence that do not destroy data.

Anti-forensics overlaps with unauthorized data destruction. The difference is that anti-forensics is the destruction or the alteration of evidence that indicates malfeasance. Unauthorized data destruction may have alternative goals or motivations, such as removing the availability or utility of data assets. Both unauthorized data destruction and anti-forensics may produce a loss of integrity, availability, utility, or authenticity, but the difference in motivation guides the selection of data assets to protect and the policies to enforce.

2.5 Defending Against Unauthorized Data Destruction

There are several methods to defend against unauthorized data destruction. For each method, a description of the protection method is given. A short discussion is also present which details the limitations to protect against certain types of unauthorized data destruction attacks.

2.5.1 Access Control

To limit the unauthorized destruction of a file, access control "protects shared resources against unauthorized accesses [...] according to an access control policy" [28].

[3]The false evidence placed on persistent storage may overwrite previously deleted data assets of interest

Access control may enforce one of the Parkerian security elements, such as confidentiality (read permission) or integrity (write permission). For unauthorized data destruction, integrity models such as the Biba Integrity model [29] or the Clark and Wilson model [30] are relevant. Integrity Models protect unauthorized modifications of data assets by explicitly assigning integrity levels for both subjects (e.g., users) and objects (e.g., data assets). Integrity levels for subjects are effectively the trustworthiness that a subject will properly modify an object [31]. However, it is difficult to be certain that an application on a computing system is trustworthy [32]. Both subjects and object integrity levels must be properly defined to protect against unauthorized data modification effectively.

Ultimately, if the attacker can gain the highest level of privilege, access control methods may fail. Other methods, such as data preservation or recovery strategies can help mitigate against cases where attackers gain high levels of access. Nonetheless, an attacker may cause significant damage even without a high level of access. For instance, an attacker who has access to a shared directory may destroy files, causing a loss of availability for other users. A malicious user within the group may securely delete a file, causing a loss of availability to other users in the group. Tricking a user to run a malicious application is another attack vector that can cause a loss of data. A malicious application may run at the same level of access to the user who ran the application, allowing the malicious application to destroy user-owned files.

Some systems may prevent overwriting altogether, even if the user gains high levels of privilege. For instance, immutable and append-only files [33, Chapter 20] or write-once, read-many (WORM) storage media [12] only allow file-appending and no other type of write access. An attacker cannot overwrite any information on immutable write-only systems. The immutability may be enforced by disabling the access permission modification or through the use of storage media that does not allow overwriting of files, such as optical disks [12].

2.5.2 Data Preservation Strategies

Mitigation and recovery strategies for unauthorized data destruction consists of data replication and data logging. If successful, either strategy can reduce the severity of unauthorized data destruction by recovering data that suffers a loss of Parkerian Hexad security elements.

The data replication strategy consists of replicating data in a location that is isolated from the attacker. If a file suffers a loss of integrity, availability, utility, or authenticity, the replicated data can replace or repair the file in question. Observing for invalid changes to a file may also help detect unauthorized data destruction.

The data logging strategy achieves the same goals with an alternative approach. All changes to files are saved and indexed. If at some point a loss of a security element is observed, the changes to the files are reverted to a valid state.

Data Replication

Data replication strategies define data objects to be preserved and therefore replace files if they succumb to a destructive attacker. As long as the adversary does not have access to the backup storage medium or the systems that manage the backups, the files in the backup are considered RA.

There are several systems to create replicates of data objects for recoverability purposes in case of destructive adversaries or accidental destruction. However, unless otherwise noted, these systems are designed for accidental destruction, or hardware/software failures and often fail to protect against motivated adversaries who seek to destroy data.

Chervenak et al. [34] survey backup techniques for file systems and serves as the foundation for the subsequent section. Definitions for each system from [34] are given, and advantages/disadvantages of each backup technique under the threat of unauthorized data destruction is detailed.

A *full or incremental backup* replicates the entire storage medium to a backup location, usually stored separately from the original [34]. In the case that a system suffers a loss, the backup is referenced to recover files. Incremental backups create copies of data that have changed from a prior backup, which saves storage space relative to a full backup.

Full and incremental backups are viable when a system suffers a loss in integrity, availability, or utility for user files or some system files that are infrequently updated. Another advantage is that full or incremental backups can complement other data backup systems. A copy of a storage medium placed offsite can supplement local backups if the attacker can compromise local backups. The offsite copy may require the attacker to compromise additional systems or have physical access to the backup copy to destroy data.

The creation of full or incremental backups is periodic, which the adversary may consider when destroying data. The adversary may compromise a security element between the creation of backups. A most recent backup copy of frequently modified files may not exist if the file is created and destroyed between a full or incremental backup. Another disadvantage is the lack of details when a system suffers a loss of a security element. A comparison of the storage medium and the full backup is necessary to identify losses of security elements. Further, the comparison may not yield details on losses to authenticity or integrity for newly created files. A system may have new files relative to the backup copy, and the administrator cannot determine if the new file contains misleading information or if the file is incomplete. Full or incremental backups may require the users and applications to halt during backup, to prevent file modification during the backup process [34]. Online backup, described below, handles the creation of a backup while users or applications are live on the system.

Online backups must preserve files that are active without disrupting the end user and without file inconsistencies [34]. The backup copies must be a valid copy of the

files or else the backup may introduce losses to security elements if an administrator uses the backup.

Online backups may provide valuable information regarding the losses of security elements during an attack. It may be possible to observe what security elements suffer a compromise for specific files. Further, supplemental information may accompany the online backup, such as active users and processes during the creation of the online backup.

If not properly handled, file inconsistencies may be introduced by online backup systems [34]. It may be difficult to distinguish inconsistencies introduced by the online backup and the attacker. Another disadvantage is that an attacker may be able to introduce temporal inconsistencies because backups do not occur instantaneously and the users on the system may modify files as the backup is in progress.

File base backups [34] are copies of files and directories. File-based backups may not include the data that may exist between files and directories, such as slack space used for malicious hiding of data [35].

File base backups allow for flexible policies for specific types of files and directories. Frequently modified files may require more frequent backups compared to more static files (e.g., system files, binaries, libraries).

File base backups do not capture information of previously deallocated files, which may provide valuable information in reconstructing assets that suffer a loss of a security element. Files that are partially destroyed, deleted, and no longer available on the file system are not backed up. A loss of availability or integrity, through partial destruction or deletion, may not be recoverable with file base backups.

Physical or device based backups [34] create a bit-by-bit replica of the disk, without relying on the file system to replicate the data.

Physical level backups capture changes to slack space [35], which may help recover from a loss of integrity (e.g., partial destruction of files) or a loss of availability (e.g., delete).

Partially destroyed files are not necessarily recoverable. Undelete and file carving tools may help recover files. However, if an attacker has overwritten the files, the file may be lost. Further, a defender cannot recover files that an attacker has completely overwritten.

Tracking Changes

A simple solution to counter unauthorized data destruction is to save all writes to persistent storage. An attacker cannot destroy data if it is impossible to overwrite previous data. However, there are several challenges in saving all writes to persistent storage that have limited storage capacity. In practice, such a system must address management of accessing the most up-to-date version of a specific file, the ability to recover prior versions of a file, and the policies to react to when the persistent storage is full.

Snapshots and copy-on-write (CoW) systems [34] allow for a file system to track changes after a snapshot of the file system is taken. A snapshot is a read-only state of an entire file system. Subsequent writes copy the portions of the data before applying the writes (copy-on-write). The written copies are also indexed to keep track of changes. If unauthorized data destruction occurs, the administrator can recover the original contents of the file by referring back to an older copy of the file.

Unification File Systems, such as UnionFS [36], support the ability to sandbox potentially untrusted entities. A sandbox is a "forked"' snapshot in which suspicious actors are placed until their actions are verified as benign. Changes to persistent storage within the sandbox are isolated from trusted users and processes. If the entity within the sandbox is valid and trustworthy, changes within the sandbox can then merge with the persistent storage. If the entity within the sandbox is malicious, the changes within the sandbox do not affect the persistent storage, and a system administrator can observe the files that the untrusted entity modified to gain knowledge of attack patterns and motivation.

The unifying feature of the above systems is the creation of a read-only view of the storage medium at some point in time. Changes to files after the snapshot are saved and tracked.

After snapshot creation, the attacker must compromise the controls for CoW (e.g., remove the snapshot, disable CoW) to destroy data. The defenders can design systems with a security perimeter to isolate the CoW snapshots and controls.

One disadvantage of snapshot and CoW systems are situations when a valid user wishes to destroy a file. Destroying a file may require additional steps to remove the file from snapshots, compared to the other backup schemes described above. Another disadvantage is that snapshots may need to be updated to free up space of unneeded copies of data. As all writes to existing files create copies, an attacker may attempt to fill the disk by repeatedly overwriting files. Rather than overwriting the same portions of the disk, as in the systems previously described, a CoW file system create copies, which is a path for an attacker to fill the disk quickly with useless data.

Append-only, log, or continuous file systems write all modifications to storage without overwriting existing data, which allows for users to roll back changes for files [16]. As storage media are bounded, append-only, log, or continuous file systems typically use a garbage collector or other mechanisms to free unwanted file versions. The unifying feature of these systems is tracking the changes throughout the lifetime of the file systems, and then the removal of obsolete information is decided later.

There are several version control systems specifically designed to recover from faults or user-specified version recovery. Version files systems, such as Elephant [16], retains important versions of specific files for recovery. The elephant system is designed to *keep landmarks*, versions of files specified by the user or through automation based on a heuristic. The elephant file system may also *keep all* version of a file. Other file systems such as for Plan 9 [37] take periodic snapshots of the file system for later recovery (if necessary). NILFS [38] continuously creates snapshots of the file system as users write to the disk, which allows for recovery of files immediately after they occur.

For digital forensics and security auditing, it may be necessary to have an exhaustive record of all files modified within an operating system to capture accurate timelines. Systems like Elephant and NILFS provide resiliency to unauthorized data destruction. However, sophisticated attackers may simply modify their strategies to subvert systems such as Elephant or NILFS. Elephant and NILFS were not designed to deceive attackers and the recovery, and checkpointing functionality is not isolated from the end user. Further, the designs of append-only file system do not adequately isolate the control mechanisms from the operating system, which may be untrusted.

As with Snapshot and CoW systems, if the controls and garbage collector for the append-only file system (and the like) are isolated from the attacker, it is not possible to destroy data directly. The attacker must rely on the garbage collector to remove data. The policy of the garbage collector or related mechanism must be carefully designed not to remove old data that may be useful to recover from a loss of a security element.

The attacker's actions may influence the garbage collector to remove old data, even if the attacker is isolated from the garbage collector. For instance, an attacker may attempt to fill the disk, similar to the strategy on CoW systems, to prompt the garbage collector to remove old records to free space. A patient attacker may be able to destroy data by waiting for the garbage collector to remove old versions of overwritten data. In practice, the garbage collector and control mechanisms for the above systems are accessible within the system that the user and, potentially, the attacker uses.

2.5.3 Data Recovery and Repair

Data destruction may occur on systems that do not use any data redundancy systems. If no backups are available to recover the original data, data recovery attempts to reconstruct the destroyed data assets. Several of the techniques presented below overlap with the field of digital forensics [39]. Specifically, data remanence,

the "retain[tion] previously stored information beyond its intended lifetime," [40] is helpful in recovering data that has suffered from unauthorized data destruction.

It may be necessary to identify the missing data assets or identify data assets that are illegally modified. A subsequent section, Section 2.5.4, discusses identification methods for unauthorized data destruction.

There are several methods to *recover or repair* previously deleted files or data. Sluethkit, for instance, recovers "orphan" files that are inaccessible through the file system but recoverable by analyzing file metadata to backtrack the location of the file [41].

File carving may be necessary to reconstruct files that are fragmented and spread across multiple locations on the storage medium [42]. However, if portions of the deleted are missing because of a different file allocating the space of the deleted file, it may be impossible to recover the file.

Data Caches

While not specifically designed to mitigate data destruction, data caches may contain portions of data assets that have suffered a loss of a security element.

Journaling file systems duplicate data to repair file system inconsistencies caused by an unexpected power loss [18]. Furthermore, file systems may store metadata associated with a data object in a separate location from the data itself [18].

Hardware components may also contain portions of a destroyed data object. For instance, a flash-based storage device may contain spare memory cells to degrade the device uniformly, [18]. Hybrid storage devices [43], which combine both solid-state drives and hard disk drives, cache frequently access files within the solid-state portion of the disk. An attacker may fail to destroy data assets located on both the solid-state and hard disk.

System caches may also pose a problem as portions of a file may still exist elsewhere. Temporary directories, swap space, or in-memory file caches may contain file fragments.

A system administrator may use all of the above to recover portions of a destroyed file. However, as with undelete and file carving methods, full recovery of a destroyed data object is not guaranteed, especially for in-memory file caches that contain highly fragmented files [44].

Digital Forensics

It is infeasible to anticipate and store *all* forensically relevant states that are generated by a computing system without negatively impacting performance [45]. CoW and Append-only storage systems help track changes to storage over time, but there is always an upper bound on the amount of storage available on real systems.

Forensic examiners use information such as system log files, checkpoints, backups, and metadata in an investigation. However, forensic quality of such information is not guaranteed. Furthermore, specific data objects, such as metadata, were not designed for digital forensics [45]. It is possible for an adversary to edit audit trails, logs, or relevant metadata as the adversary may have access to such items or gain access to such items through the use of rootkits [45–47]. Thus, the preservation of quality data objects is further complicated under an adversary who seeks to destroy or alter incriminating evidence through anti-forensic techniques. Prior work mitigates these issues by isolating the logs outside the operating system [48] or secure audit logging [49] to inhibit the attacker's ability to corrupt or view log files.

An adversary may replace the incriminating data objects with random binary bits, as suggested by NIST, to ensure that recovery is infeasible. However, Savoldi et al. [21] indicate that using such techniques are not stealthy as the distribution of securely deleted data objects is distinguishable on the disk. Further, the tools for secure deletion may generate additional evidence of malfeasance [27]. Nonetheless,

the trade-off between stealthiness and secure deletion is a decision that an adversary must make.

2.5.4 Unauthorized Destruction Detection Strategies

Detection strategies attempt to detect destruction and react according to some policies to mitigate or stop destructive actions or malware.

Signatures-Based Detection

Signature-based detection monitors behaviors and features for known destructive malware [50]. The features can be (i) dynamic, derived from runtime interaction with the suspected application; (ii) static, derived from code sequences that represent malicious behavior; (iii) a hybrid of both [50].

The advantage of static signature-based detection is that it may protect a system against known malware. However, there are several limitations in signature-based detection. Some malware employs packers, metamorphic, or polymorphic techniques [51] to obfuscate a binary, to hide features that signature-based detection schemes rely upon. Further, generating signatures relies on careful analysis of malware and the creation of signatures potentially by human experts [50]. Producing malware signatures is outpaced by the rate at which malware is written. Another issue is that signatures fail to detect previously unseen malware (zero-day malware) [50].

There are several examples of signature-based rules to detect destructive software. For instance, device drivers that access the raw storage medium, which bypassing the file system, is a sign of potentially destructive malware. Another example is the detection of static features in known anti-forensic binaries such as timestamp obfuscation or secure delete tools.

Dynamic signature-based features may observe destructive behaviors at runtime, which may be more flexible in identifying destructive malware static-based detection. Several standard secure delete algorithms contain distinguishing features that may

indicate data destruction attempts. Secure delete features may include sequences of fixed patterns, such as zero bits, one bits, or other standard secure delete patterns. Integrity monitoring software such as Tripwire [14] observe for any modification of a file as an indication of an adverse modification of data assets. Overwriting portions of a file that are not frequently updated, such as a file signature, may also indicate a destructive action [23].

The advantage of dynamic signature-based detection for data destruction is that it forces the attacker to modify behavior rather than modifying its source code/binary. There are several methods [51] to obfuscate the static features in a binary file. Another advantage is that the detection of destruction is not unique to a single instance of malware, as in static binary features, but may also detect known destructive malware that behaves similarly.

The disadvantage of dynamic features is false positive detection. Some benign software may behave similarly to destructive malware. For fraudulent timestamp modification, files from an archive (e.g., zip, tar, or rar files) may preserve the original timestamp metadata. When extracting onto a new system, the discrepancy of the newly created files and the timestamps written to the file system may cause a false positive, as shown by Pennington et al. [52]. Similarly, updating system binaries may trigger integrity monitoring software.

Anomaly-Based Detection

Anomaly-based detection observes and learns normal system behavior, without the presence of an attacker, and detects derivations from the normal system behavior [50]. The advantage of anomaly-based detection schemes is the possibility of detecting zero-day attacks, and the disadvantage is high false alarm rates and the challenge of identifying features for detecting normal/anomalous behaviors [50].

Several anomaly-based detection schemes can help detect data destruction attacks. For instance, a system may observe the entropy of I/O write buffers. Anomalously

high entropy on a write buffer may indicate unauthorized encryption of a file, indicating the potential presence of Ransomware [23, 24, 53]. Anomalous I/O patterns may also indicate data destruction attacks. Prior work observes that crypto ransomware has distinct I/O patterns that may indicate an unauthorized encryption [23,24]. Contienlla et al. in SheildFS [53] use anomalously high rates of read, write, or file rename requests as an indication of unauthorized data encryption.

High false positive rates are a disadvantage of anomaly-based detection schemes. For instance, false positive detection is observed in ShieldFS [53] when applications such as Explorer and Visual Studio access a large number of files in short period. The confusion may cause an incorrect insertion of deception or cause the system to slow down at the incorrect time, both of which are undesirable for benign users on the system.

2.5.5 Combining Detection and Preservation

Early works [52,54] have inspired the design of systems that prevent the accidental or unauthorized data destruction or modification of files on persistent storage. The Self-Securing Storage System (S4) [54] keeps all versions of changes to files for a fixed window of time. The work builds on log-structured file systems but at a finer granularity of file changes. Further, S4 uses security perimeters, which separates access to the recovery and configuration mechanisms from the OS in case the OS becomes untrusted. Without the security perimeter, an attacker can remove versions of a file, effectively destroying the data. For an attacker to destroy data on S4, she must compromise the OS and the management interface. If the attacker only destroys data on the OS, the administrator of S4 can recover the data.

An extension to S4 is a Storage-Based Intrusion Detection Systems (SBIDS) [52], which observes disk access patterns for unauthorized actions. SBIDS observe for unusual access patterns that indicate that a machine may be under attack. The suspicious actions include erroneous data/time modifications, unexpected update patterns

for critical files (e.g., system binaries or log files), or the hiding of files and directories. By combining S4 with SBIDS, an administrator can react to attacks, which are observed and analyzed outside of the untrusted OS, and recover files that suffer a loss of a security element.

SBIDS and S4 provide an excellent solution for attacks against system binaries, configuration files, logs, libraries, kernel objects, or unauthorized data and timestamp modifications. Specifically, SBIDS demonstrates the ability detect unauthorized modifications to date and time stamps (a loss of authenticity), deletion of log records in log files (a loss of integrity), and the creation of hidden files (loss of availability). However, SBIDS does not address the problem of detecting unauthorized actions on critical user files that are frequently updated. User critical files can vary and do not have a rigid structure and access pattern compared to critical system files.

Other related work that combines detection and protection is ShieldFS [53]. By using a CoW system, ShieldFS protects a Windows OS against Crytpo Ransomware. If a process behaves like Ransomware, the files under unauthorized encryption are recoverable by revoking the changes. The system monitors write patterns through a custom driver. Features are extracted from disk activity, through the custom driver, along with in-memory cryptographic features, referred to as CryptoFinder. The features feed into a multi-tier classifier, which examines various changes to the storage medium over time. ShieldFS mostly relies on the CryptoFinder, which contributed to 69.3% of all the malicious samples in their experimental evaluation.

The results indicate that the SheildFS approach works for cryptographic ransomware, which causes a loss of utility by data transformation. Unfortunately, there are a variety of methods to destroy data. For instance, wiper malware may not use ciphers to destroy files. As discussed in Section 2.3, data destruction methodologies may vary, and crypto ransomware focuses on a narrow set of methods (a loss of utility through unauthorized data transformation). The experiments in this dissertation examine a broader set of data destruction methods. The wide variety of methods contribute to the challenge of unauthorized data destruction.

The ShieldFS [53] work inspired some of the design and architecture decisions of DecMS. In short, the experimental evaluation improves upon the design of ShieldFS [53]. The difference in the experimental design is that the monitoring, analysis, and data preservation mechanism are isolated from the attacker, and the OS where the attacker resides is assumed to be untrusted. SheildFS provides some defense against an attacker who attempts to disable SheildFS. Although the ShieldFS drivers are designed to be immutable, the authors describe a path to disable ShieldFS under an attacker who has administrative privileges or by compromising the OS kernel.

The Taser Intrusion Recovery System [55] uses taint analysis with filesystem snapshots and audit logs to help identify and revert file changes after an attack discovery. Taser relies on the use of an IDS or a system administrator to identify "detection points," which may be a file or process that is suspicious. The recovery algorithm references both the detection point and audit log and backtracks to find all relevant files and reverts the changes. Benign changes replay from the snapshot point to recover the changes that are not affected by the tainted files. It may be difficult to manage and revert the effects of destructive actions when benign applications are affected by the unauthorized data destruction. Taser attempts to resolve the conflicts that may occur when a tainted file or process interacts with a benign file or process.

2.6 Deception for Defense

Cybersecurity deception is described in detail by Almeshekah and Spafford [7]. A summary of relevant topics and definitions for this dissertation is given below.

Deception, as defined by Almeshekah and Spafford [56] is "Planned actions taken to mislead and confuse attackers and to thereby cause them to take (or not take) specific actions that aid computer-security defenses." However, defensive deception when implemented as an isolated system has limitations, and there are benefits in using deception in conjunction with other security tools [7].

In this dissertation, we explore deception as an enhancement of data preservation (and other security mechanisms) under unauthorized data modification. The deceptive strategies are designed to work on systems with real users and not as an isolated system for interacting with adversaries (such as honeypots).

Based on Almeshekah and Spafford's taxonomy for deception [7], this dissertation describes deceptive components with Bell and Whaley's terminology of dissimulation (hiding the real) and simulation (showing the false). Dissimulation components consist of (i) masking, (ii) repackaging, and (iii) dazzling. The simulation components consist of (i) mimicking, (ii) inventing, and (iii) decoying.

Bell and Whaley order the above deceptive components from strongest to weakest form of deception, where (i) indicates the strongest and (iii) is the weakest form of simulation/dissimulation. Almehsekah and Spafford [7] build on the Bell and Whaley's work, applied to information security:

> We can *deny* the target access to the truth and show him the deceit. When we cannot stop the truth from being observed, we can *misdirect* the target's focus to the deceit. When we cannot influence the target's focus, we can confuse the target by presenting him with the truth and one or more plausible deceits.

When designing deceptive systems, this dissertation follows the ordering given above. Deception can also be consistent or inconsistent [57]. Consistent deception presents a consistent but false view of reality. The deception remains persistent despite the adversary's effort in finding inconsistencies. The alternative is "inconsistent deception" that presents deceptive information to an adversary, but the deception is ephemeral or is easily verified to be false with some effort. Inconsistent deception attempts to disorient or confuse the adversary, which may be beneficial even if the adversary is aware of the ruse. The system defender, who controls the fictional reality, can indirectly influence the adversary's actions. The adversary's perception of a system is unclear because the defender introduces inconsistent information which requires time

and effort for the adversary to disseminate. If the attacker becomes aware of the inconsistency, she may conclude that the (i) defender is using deception, (ii) the system is faulty, or (iii) her perception of the system is incorrect. Neagoe and Bishop [57] argue that consistent perception is difficult to achieve because of the variety of paths that an attacker can use to examine if some information is consistent or inconsistent. While consistent deception is used in sandbox systems such as honeypots, inconsistent deception can induce confusion, frustration, or impede the attacker from achieving her goals.

Both consistent and inconsistent deception provide advantages and disadvantages. The challenge with consistent deception is to present the deception regardless of the adversary's strategies to reveal the truth. In comparison to inconsistent deception, consistent deception requires more effort for the defenders of the system. Alternatively, the inconsistent deception may still disrupt the attacker who does not attempt to unveil the deception and may require less effort by the defenders to inject the deception. This dissertation considers both consistent and inconsistent deception in some of the experimental evaluation.

In defending computing defense system, there are several benefits in using deception with other security technologies. The benefits [56] are (i) "increases the entropy of leaked information," (ii) "increases the information obtained from compromise attempts," (iii) "give defenders an edge in the [Observe, Orient, Decide, and Act] OODA loop," and (iv) "increases the risk of attacking computer systems from the adversaries side." This thesis does not explicitly explore the of goals (i) and (ii). Information leakage is a loss of confidentiality and is not a necessary condition in unauthorized data modification as mentioned in Section 2.2. It is possible to obtain information about an adversary from failed unauthorized data destruction attempts. However, the work here only gathers such information and does not attribute or attempt to understand the attacker strategies.

This thesis explores the benefits of (iii) and (iv). The OODA loop models how adversaries and defenders behave in a conflict. The player who acts first has a tactical

advantage, and deception may disrupt a player's view (e.g., observe and orient) of the conflict, causing impedance during decision or selecting a suboptimal action [7, 56]. The advantage for the defender is additional time to decide and act when faced with an adversary.

For unauthorized data modification, the defender may be able to prepare data redundancy in anticipation to a destructive action. If the redundancy or replication is isolated and hidden from the attacker, the attacker may believe that she is destroying data without knowing that the assets are under protection. Through careful planning and a successful deceit, a defender can recover from unauthorized data modification quickly, giving the defender an advantage in the OODA loop. Injecting false responses in the system may also cause the attacker to asset the situation, find alternative attack strategies, and possibly confusing the attacker [7]. In the former case, the defender can react more quickly to unauthorized data destruction than what the attacker perceives, and in the latter case, the attacker is slowed down which gives the defenders more time to react to an attacker.

Disrupting the attacker's destruction through the use of false information may increase the risk for the attacker (benefit (iv)). For instance, anti-forensics techniques destroy incriminating information on a system. Feeding false information to an attacker, such as showing the attacker's expected "successful" outcome when she is destroying incriminating information, gives the attacker a false sense of assurance. Further, some data destruction attacks attempt to be stealthy until the attack is unleashed to maximize the effectiveness. If the attacker believes that she is destroying data, but the system is resilient to the attack, the adversary risks revealing her presence within a system.

Almesheka and Spafford describe a model to plan and integrate deception into computer security defense [7]. The planning requires strategic goals and how an attacker should react to deceit. The deception should exploit an attacker's bias and should reflect the simulation and dissimulation strategies. A feedback channel mea-

sures how an attacker reacts to the deception. Further, the risks of using deceptions should be well-understood.

In particular, the costs of using deception may impact the end user, especially if resources are shared between the deceptive system and the system in need of defense. The deceptive system may have to compete for resources, which may be undesirable for the system under protection. There is a need to understand how deceptive systems effect "normal users' activities" and identify if the risks of using deception are acceptable [7]. Distinguishing between truths and lies is also a challenge in some computing applications Forensics attempts to ascertain the truth, and deceptive information may lead an examiner to invalid conclusions. The deception introduced for defensive purposes must be identifiable for a forensic examiner but difficult for an attacker to identify as deceitful. Details for the deception planning, integration, risks, benefits, and costs for the proposed system is found in Chapter 4.

Building on the work by Neagoe et al. [57], Sun et al. present Chameleon [58], which explores consistent and inconsistent deception to vary the behavior of operating systems. The authors present a case study to demonstrate Chameleon's feasibility against botnets. Chameleon injects deceit in several system calls, which is designed to disrupt fragile software. The results indicate that the unpredictability of the system calls slowed down the botnet's ability to send spam without significantly degrading the performance of standard desktop applications. The results indicate that malware is sensitive to minor operating system misbehavior.

A viable deceptive strategy evaluated in this dissertation is mimicking a system that an adversary expects and mask the adversary from the truthful nature of the system. There are several techniques, primarily used by rootkits to hide from system administrators and anti-malware software, to hide components of a system [25,59,60].

Isolation is necessary to preserve the integrity of the preserved data that is under destruction or any forensically valuable information. Isolation relies on compartmentalization or security parameters such that the user is unable to modify critical

components or data. The existing techniques are designed to isolate rather than hide the presence of the system.

For instance, FreeBSD could be configured to run in a secure mode which ensures that flags for immutable and append-only files cannot be changed and kernel modules cannot be loaded after boot [33, Chapter 20]. It may be possible to write logs and preserve the data under destruction into an immutable and append-only file to prevent an attacker from destroying files, even at the highest attacker privilege level [4]. While isolation techniques are critical to preserving the integrity of logs or preserved data, attackers could simply look for the presence of such files as evidence that a system is using the above configuration and modify their attack strategy.

There are some existing data hiding techniques to increase the challenge of discovering critical components of a system. Several of the data hiding techniques are in use in malicious software, such as rootkits. Kernel Object Hooking (KOH) and Dynamic Kernel Object Manipulation (DKOM) are well-known strategies that rootkits utilize to stay hidden from anti-malware software and system administrators. There exists a rich body of literature ([59, Chapters 3-6]) that details how rootkits remain hidden from monitoring. The same techniques can hide the presence of deceptive technologies.

For instance, files associated with a deceptive system in persistent storage could be inaccessible to any users within the system by modifying system calls (through DKOM or KOH) to identify critical files and hide their presence from the operating system [59, 61]. Other techniques, such as unlinking processes from the process lists, could also hide the presence of deceptive components within the operating system.

Another technique that may combine both isolation and hiding is Virtual Machine Introspection (VMI). VMI allows a host machine to access the contents of memory within a guest virtual machine by examining the "hardware states and events and uses this information to extrapolate the software state" of the virtual machine [62]. A deceptive system could be placed within the Virtual Machine Manager (VMM), intro-

[4] Assuming that the attacker cannot access the system before boot to disable immutable files.

spect a guest virtual machine, preserve information, and respond deceptively. VMI provides some distinct advantages over the previously discussed hiding techniques. One disadvantage of DKOM and KOH is that the core functionality exists within the same system where a potential attacker resides. An attacker could compromise the system and uncover the deception. A challenge of VMI is to extrapolate the semantic meaning of low-level hardware changes [62]. However, techniques exist [63] to address the challenge of the "semantic gap" [63].

Other isolation and data hiding techniques place system components near the hardware, which are inaccessible unless an administrator has physical access to the machine. Another design alternative is placing the deceptive system on the network, which intercepts, analyzes, and modifies network traffic outside of the system under protection. All of the above data isolation and hiding techniques are considered in designing the architecture for the experimental evaluation in Section 4.1.

2.7 Other Related Work

Barik et al. describe a system that enhances the logging mechanism for MAC DTS (Modified, accessed, created, Date and timestamp) for the Ext2 File system [64]. The proposed system preserves MAC DTS for key files through the use of Loadable Kernel Modules for virtual file systems (VFS). The system uses two unused `inode` fields in the VFS to (i) indicate the file is critical and (ii) a pointer to the MAC DTS trail blocks. The goal is to preserve MAC DTS against an adversary who attempts to purge evidence by modifying critical timestamps.

While the work by Barik et al. [64] is similar to the work presented in this dissertation, there are some crucial distinctions. First, Barik et al. do not explicitly use deception to trick an adversary into believing that their evidence wiping technique is working correctly [64]. Secondly, the authors do not attempt to hide the presence of the of their proposed system. If an adversary is aware of the scheme proposed by Barik et al. [64], they can trivially check to see if the two unused inode fields are in

use. Third, the logging mechanism is not resilient to unauthorized modification. If an adversary is aware of the scheme proposed by Barik et al. [64], she can merely enumerate through the MAC DTS Trail Blocks and modify the entries.

Milkovic [47] describes how a rootkit could subvert memory forensic tools[5] that dump the contents of memory for forensic analysis and possibly exposing the rootkit to an examiner. Milkovic proposes a method that intercepts system write calls and examines the parameters to determine if a memory acquisition tool is dumping memory. If a memory acquisition tool is detected, the rootkit removes evidence of its existence from memory before the memory acquisition tool can dump the contents of memory for later examination. From the forensic investigator's perspective, the memory acquisition tool appears to be behaving normally but in actuality is failing to capture malicious processes that are memory-resident. The same methodology applies to defense. DecMS could be designed to be memory-resident. If an attacker attempts to observe portions of memory that reveal DecMS, intercept the call and produce plausible results to induce attacker doubt regarding DecMS.

Kuperman and Spafford [65] describe `AUDLIB`, a method to wrap libraries to produce application-level audit information. `AUDLIB` uses Library Interposition to intercept library function calls to generate detailed audit information with a higher degree of fidelity than typical kernel logging. `AUDLIB` can produce a detailed audit trail that includes timestamps, PID, PPID, the caller, and the library that the caller invoked. Further, `AUDLIB` can identify misuses such as string, overflow, and return-to-libc attacks with little overhead to the end user.

`AUDLIB` is capable of detecting anti-forensics actions and other items of forensic interests. However, `AUDLIB` is not designed with deceptive capabilities. An attacker could simply check to see if `AUDLIB` is installed on a given operating system and disable it. Library Interposition is a valuable method that DecMS could use to identify actions of forensic interest but must be combined with hiding and isolation techniques to be effective.

[5] `http://www.volatilityfoundation.org/`

Nance et al. discuss the research challenges involved in digital forensics and VMI [66]. In particular, they point out the challenges, from a digital forensics perspective, of developing a VMI tool. They discuss the methods in which the goals VMI can be achieved while considering the risks of unauthorized entities who may use VMI for malfeasance. The work presented in this dissertation assumes that the interposition, regardless of the method, is trusted and not compromised. Further, the authors discuss the ability to conduct VMI for covert operations. The authors point out that covert VMI is applicable in deception, where one can simulate hardware failures or in general, influence the actions of processes. The discussion motivates the research presented in this dissertation. The authors point out that such a system is challenging because of the issues of "monitoring non-quiescent systems." Finally, the authors also discuss the possibility of detecting VMI on a virtual machine. The authors discuss that an attacker could analyze system wall time or page faults to determine if she is running within a VM with VMI. While the research challenge of exploring VMI detection is relevant to DecMS with VMI, it is out of scope with the thesis statement. However, design choices of DecMS are driven from the discussion presented by Nance et al. [66], as it is more challenging to determine if a system has VMI compared to determine if a system runs deceptive software within the host operating system.

Savoldi et al. [21] evaluates the feasibility of identifying regions on a disk where a secure deletion action *may have* occurred. The same methods can be used to identify where destruction took place to determine destruction *as they* occur. As shown by Savoldi et al. [21], the statistical test suite for pseudorandom number generators [67] and the entropy-based classifier could be used to identify secure delete actions.

More recent work uses similar methods to detect secure deletion before reaching the disk. Unveil [24] and CryptoDrop [23] use features to detect the presence of destructive actions. Unveil monitors for stealthy Ransomware within an anti-malware analysis system and monitors the entropy of writes to the disk as a detection feature. CryptoDrop uses file signatures and entropy as part of its detection features.

Bacs et al. [68] design an intrusion detection system called Storage-Level Intrusion ChecKer (SLICK) to protect virtual storage devices. SLICK monitors write accesses to specific regions of the disk that are critical, such as the master boot record (MBR), bootloader, or space beyond the region of the file system. Particularly, the bootloader and MBR are locations that an attacker replaces to execute code before the OS starts or causes the OS to fail to boot, respectively. SLICK monitors for any modifications in those regions and successfully detects all modifications.

The Drakvuf system [69] uses VMI to analyze malware through VMI. The goals in Drakvuf that related to this thesis document include data collection fidelity, stealthiness, and isolation. Data fidelity in VMI is a challenge [66]. Drakvuf uses active VMI to insert breakpoints at key locations to halt the virtual machine to gather runtime information, thus avoiding the non-quiescent inconsistency problems [66]. Drakvuf is a malware analysis system designed to test and identify malicious applications. With some effort, it may be possible to use Drakvuf to protect live virtual machine systems with real users. Some of the experimental results use Drakvuf to observe for data destruction actions. One of the prototype systems, DecMS-VMI, extends Drakvuf to analyze for data destruction, preserves files, and injects deceptive responses to protect a guest virtual machine.

3 THREATS AND DECEPTION

Designing a DecMS requires an understanding of the threats, the cost/risks of introducing deception, monitoring for the desirable conditions to introduce deception, and the methodologies to achieve the deceptive goals [7]. As with any security system, trade-offs guide in the design of the architecture. The design of the architecture explored in this dissertation is guided deceptive planning detailed in [7].

First, an explicit threat model is necessary to design deception for defense. Building on the threat space introduced in Section 2.2, a definition of several threats are given. Details regarding the destructive methods are presented, which is critical in identifying destruction. Later, in Chapter 5, instances of each of the threats described here are given to evaluate prototypes of DecMS. The threat models will help support the DecMS Identification and Reduction Goals.

Second, the deceptive planning in Section 3.2 follows the guidelines provided by [7] to plan a defensive deception that is effective in deceiving an attacker. The deceptive goals and attacker biases guide the design decisions in subsequent sections. Based on the assumed threats, the deception planning details the needs for monitoring for a DecMS. Several services are identified to help facilitate the deception. The contributions include a deceptive plan to produce plausible false reality and support the DecMS Reduction Goal.

Third, the costs of deception within an I/O system are given. A crucial part of planning for deception is understanding the costs and risks of using a deceptive system [7]. Efficiency degradation is the cost of placing deception within the I/O system. System defenders should determine the acceptable trade-off for gaining resiliency of data destruction at the cost I/O performance. A set of costs for deploying DecMS is defined. Given the costs, identifying the computing systems where a DecMS can provide defense benefits is also considered. The contributions include the cost factors

associated with deception on I/O systems and several computing systems that may be suitable for a DecMS. Further, the costs of deception and suitable systems guide the design choices for the Impedance and Preservation Goals.

Fourth, the design space and architectures to exploit the adversary's biases are detailed. The architectures vary in complexity and effort necessary to protect against unauthorized data destruction. Some design and integration methods provide a balance of achieving the deceptive goals with an acceptable cost increase. The contributions include several design strategies given the threats, assumptions, and the cost factors in using deception to protect against unauthorized data destruction.

Finally, the placement of DecMS is also considered. The benefit of placing the monitoring and deception services within the system under protection is a simplistic design but at the cost of stealthiness. Alternatively, DecMS can be placed outside the system under protection but may require additional work to monitor the system from the outside. The challenges of both designs are detailed, and the contributions include the trade-offs of both design. Further, the monitoring and protection policies for identifying when the DecMS should impede data destruction to protect critical assets are also considered.

3.1 Threat Model and Assumptions

The selection basis for the threats to evaluate the hypothesis statement is as follows. First, the threats should be instances of real unauthorized data destruction and not synthetically created. Using real malware samples will demonstrate the effectiveness of exploiting the confirmation bias of current malware threats. To adequately assess the threat space, the evaluations use several threats to span the possible losses to Parkerian Hexad security elements caused by unauthorized data destruction. The selection of threats evaluates the feasibility for DecMS to protect against all security elements that are relevant to data destruction, as identified in Section 2.2.

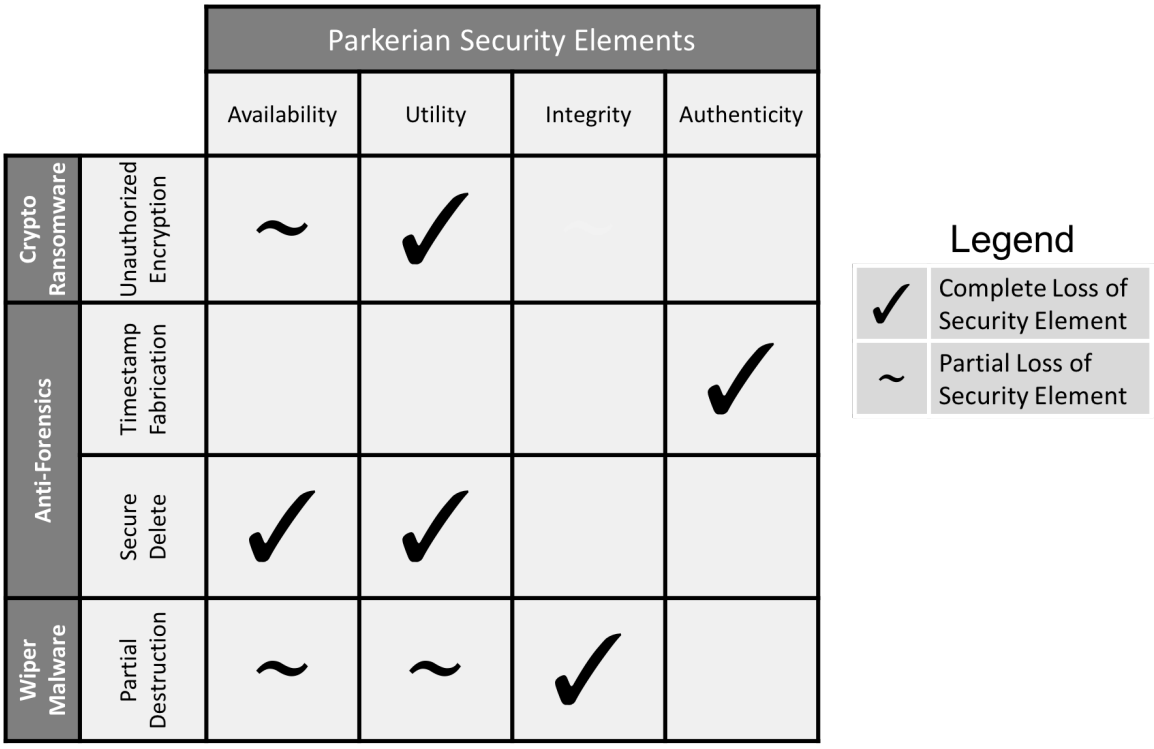


Figure 3.1.: Threats and loss of Parkerian security elements for various data destruction attacks.

Figure 3.1 illustrates three types of threats, their respective data destruction method, and the loss of a Parkerian security element. Note that between the anti-forensics and wiper malware threats, the entire threat space is covered. Below, each of the three threats from Figure 3.1 is detailed.

The definitions of assets that DecMS protects are in Section 2.1, which include RA and NRA files and metadata stored on a file system. Each threat focuses on a subset of assets to meet their end goals. The goals, motivations, and targeted assets are necessary to guide the deception design and defense, which refined in subsequent sections.

A definition of benign users is also necessary as the addition of deception relating to I/O may affect legitimate users, by limiting the resources available to the users or

by incorrectly presenting deception to benign users. The benign user is also defined below and will serve as a baseline in the experimental evaluation.

3.1.1 Wiper Malware Threats on Integrity

Wiper malware destroys digital assets to render computing services and data unusable. Other wiper malware may have different goals and methods. Below, a description of a particular type of wiper malware that targets the loss of integrity is detailed for evaluating DecMS.

Goals

The goal of wiper malware is to destroy data assets to render data recovery difficult and to cause the system to become unusable. The motivation is to bring computing resources offline or to halt users from completing tasks. The specific reasons may include financial gain, failing to pay extortion or to cause chaos.

Targets

Wiper malware targets both RA and NRA objects on a file system. RA objects may include partition tables, master boot records (MBR), or other assets that are critical for system stability. Several recovery tools exist to repair MBR [70] or partition table corruption [71]. The NRA includes frequently updated user files, such as those found in a home directory.

Methods

Wiper malware attempts to destroy files quickly with the goal of making a recovery difficult. Portions of files, such as file signatures, are overwritten so that recovery software cannot easily distinguish the beginning of a file. Overwriting file system

data structures such as the MBR or partition table also complicates file recovery mechanisms as the indexes to the files are no longer available to the system defenders.

The wiper malware threat focuses on causing a loss of integrity of RAs and NRAs. The entire contents of a file are not entirely lost to the above methods. It may be possible to recover portions of a file because of the methods above only target a small portion of a file rather than the entire asset. Figure 3.1 shows that the wiper malware threat causes a complete loss of integrity from the partial destruction of files. A partial loss of availability and utility is the result of the possibility of partial recovery NRAs.

Alternative Goals and Methods

Wiper malware may also target evidence of malfeasance to hide the methodology of the data destruction from forensic investigators or system defenders. Wipers may also attempt to overwrite the entire disk or file but at the cost of the speed of exploitation. A full destruction of a particular data asset, as defined for the evaluation of DecMS, is closely related to anti-forensics threats to Availability and Utility.

3.1.2 Anti-Forensics Threats to Availability and Utility

Anti-forensic techniques are specifically designed to overcome forensic analysis software [35] or destroy digital evidence rendering it unrecoverable [27]. Data destruction is a particularly difficult problem to overcome in forensic investigation [27]. An attacker will not simply delete incriminating evidence, but rather, *securely delete* incriminating data segments on the disk.

Goals

The goal of the anti-forensic attacks on Availability and Utility is to purge NRA from a persistent storage medium completely. The files under destruction should no

longer be available, and information that replaces the targeted files should have no utility.

Targets

The assumed targets of Anti-forensic attacks on Availability and Utility are NRA. Nonrecoverable files represent forensic evidence that is local to the storage medium. The evidence is assumed not to be replicated elsewhere.

Methods

The methods for data destruction are known secure delete algorithms. Several of the algorithms follow standards that are designed to destroy targeted data completely and may include multiple overwrites. For example, tools such as `shred` [72] or `srm` [73] are designed to erase data in a way that renders recovery impossible. The secure delete methods the file, the associated metadata, and any information stored within file system data structures.

Alternative Goals and Methods

Secure delete methods overlap with anti-forensics and wiper malware. It is possible to securely delete data from a file by overwriting the complete file with arbitrary data. Anti-forensics may target specific records in a log file rather than destroying the whole file. The partial destruction of a file overlaps with the assumed methods of the wiper malware threat.

3.1.3 Anti-Forensics Threats to Authenticity

An attacker may destroy forensically valuable information such as file timestamps, which may be overwritten with misleading information causing a loss of authenticity.

Goals

The goal of an anti-forensic attack on Authenticity is to overwrite data with misleading information to obfuscate the evidence of a crime. Forensic examiners use file system timestamps to create a forensic timeline, and replacing timestamps with fraudulent times is one method to reduce the quality of forensic evidence.

Targets

The assumed targets for threats against authenticity are replacing the MAC DTS for the file system metadata associated with a given file.

Methods

There exist libraries or system calls that handle changes to MAC DTS file system metadata. The assumption is that the attacker uses the well-documented library to modify MAC DTS to change the timestamp of a file. The MAC DTS replacement can be innocuous, blending in with other files without raising suspicion.

Alternative Goals and Methods

Rather than replacing the MAC DTS, other file system metadata can be replaced with misleading information. Rather than removing a log entry, for instance, the adversary may replace the log with an entry that is innocuous. Other attacks that cause a loss of authenticity may include replacing executable programs with ones that contain hidden functionality such as a backdoor into the system.

A threat that modifies MAC DTS satisfies the requirement of exploring an attack on authenticity. The alternative goals and methods are protected by other mechanisms, such as integrity monitoring for binaries and log files.

3.1.4 Crypto Ransomware Threats to Authenticity

Rather than destroying critical data, Crypto Ransomware renders a target's computer or data unusable unless the user pays a fee. The focus of ransomware is to cause a loss of data utility and availability to the user.

Targets

The targets for the Crypto Ransomware threats are user files that are NRA. RA are not considered as a user or system administrator can easily replace the file if backups exist.

Goals

The goal of Crypto Ransomware is to extort payment by transforming user files into an unusable state. Restoring files is not possible unless an extortion payment is received.

Method

The assumption of the transformation method is an encryption method with a key that is not in the user's or system administrator's possession. The key is assumed to be computationally intractable to recover.

Alternative Goals and Methods

Other data transformation methods are possible that render user files unrecoverable. For instance, a steganographic method could place user files within multimedia files. The extraction can use a key, which is out of the user's possession. However, steganography is not typically used in this way.

3.1.5 Benign User

A benign user is assumed to be well behaved and does not attempt do any unauthorized data destruction. The impact of the deceptive system should not hinder the end users' ability to complete their tasks. The benign user will serve as a baseline for the experimental evaluation. Several user tasks are simulated in the experimental evaluation to determine if the deception causes any disruption to the end user.

A user may write compressed files to the disk, which may have high entropy and potentially confuse a monitor for identifying destructive writes. High entropy writes may cause misclassification. Several data destructive methods (Crypto Ransomware and secure delete) overwrite assets with randomly selected bits. The false positive may cause unwanted disruptions for the end user or may indicate signs of the existence of DecMS for threats.

3.1.6 Assumptions

Assumptions for all of the above threats follows:

1. It is assumed that the threat compromises the machine and can execute destructive actions.

2. The threat adversaries destroys data by overwriting a data object, partially or completely, on a storage medium accessible from the compromised machine.

3. The attacker does not have physical access to the storage medium. Ergo, the attacker cannot physically destroy the storage medium and uses the software data destruction methods outlined above.

4. The attacker may have administrator access to the compromised machine but cannot avoid monitoring [1].

[1]For example, Kernel Object Hooking (KOH), Dynamic Kernel Object Manipulation (DKOM), or Direct Kernel Structure Manipulation (DKSM) [59, 74] could be used to remain hidden from security monitoring from within an OS.

3.2 Planning Deception

In this section, the plans for the deception is outlined to evaluate the hypothesis statement for the dissertation.

3.2.1 Strategic Goal

As detailed in [7], it is critical to specify the strategic goals of the deception. The goals of the deception are given.

1. To **gain information** about the targets of the destructive adversary while **preserving data** under destruction.
2. If preservation of data is not possible, **impede the destruction** with plausible but misleading faults that are difficult to attribute.
3. Decrease the effectiveness of data destruction while increasing the risk of revealing the adversary and her techniques.

The first goal is to provide the system defenders with a list of assets that adversaries target and aligns with the DecMS Identification Goal. The list of assets may provide insight regarding the data assets that interest the threats. Rather than preventing the destruction, the data under destruction is preserved and isolated from the attacker, aligning with the DecMS Preservation Goal. While the same benefits can be gained from dynamic malware analysis, some preliminary information is obtainable from the DecMS. Further, it may be challenging to capture the malware sample, as seen in some wiper malware that destroys itself before wiping files from persistent storage [75].

The second goal is to prevent a potential weakness of the deceptive strategy in use for the first goal. As storage space is fixed, there may be a situation where preserving data is not possible. It may be best to slow the attacker down or cause confusion to prevent the storage medium from becoming full. The impedance may allow time for

the system to remove useless information, compress data to make space to preserve data from potential destruction or allow time to analyze the system state, which aligns with the DecMS Impedance Goal. The deception that will react to such events is designed to appear plausible from the perspective of the end user on the system under protection to not raise suspicion.

The third deceptive goal is to reduce the effectiveness of an attacker's method to destroy data, satisfying the DecMS Reduction Goal. As the first two goals are to deceive the attacker, the third goal is successful if the deception can mitigate the destruction attempts. The more successful the data preservation is, the less reliable the tools and data destruction methods are for the adversary, even if the adversary can fool the analysis of data destruction. There should be a risk for the adversary even if DecMS produces a false negative when attempting to identify data destruction. The increased risk could be that the data under destruction may still be recoverable or the effort needed to destroy the data does not match the perception of the adversary.

3.2.2 Adversary Reaction

The adversary should assume that all of her destructive actions are successful, even if the adversary validates that the data under destruction is no longer available. Any impedance caused by the secondary goal should be innocuous and not raise the adversary's suspicion. The adversary should not suspect that the data destruction tools are not effective in meeting her goals. The threats should continue operating as if the data destruction are effective.

3.2.3 Attacker Bias

The bias that is exploited is the confirmation bias as mentioned in [7]. The adversary assumes that the information presented to her is truthful. Any detection of malicious behavior will cause a denial of access to the compromised system to prevent further disarray. The confirmation bias is the belief that most computing systems

report truthful information about the system state and not attempt to deceive the end user [7]. A destructive action should be successful, and the expected results should be verifiable. However, it may be unnecessary as there is typically no reason to validate a result of changes to persistent storage because past interactions have biased the user into believing results do not change unless explicitly modified by some entity within the system. Further, it is common to occasionally see a drop of performance in computing for a variety of reasons (e.g., network issues, memory leakage, waiting for I/O, scheduling, competing for shared resources).

3.2.4 Deceptive Components

Based on the goals, there are several viable options for the deceptive components. The deception should be in regards to the system's functionality or state [7]. Further, the deceptive system should "identify patterns and characteristics [...] and the conditions when to deceive" [7]. Identifying destructive patterns, characteristics, and conditions require DecMS to (i) monitor I/O and (ii) analyze the I/O.

There are several methods to monitor and analyze I/O. To provide flexibility in observing and analyzing the destructive methods outlined in Section 3.1, DecMS uses several policies that are defined by the system defenders. In DecMS, the Monitoring Policies define the methods to identify adversary destructive behaviors. The Monitoring Policies consist of an Observation Policy, which defines types of I/O requests that may be related to data destruction. The Observation Policy focuses on observing the I/O that may be of interest. The Analysis Policy takes the observable information per the Observation Policy and determines the conditions when to deceive and to preserve data that is under destruction. If the analysis policy identifies that a deception or preservation should take place, the Protection Policies detail the appropriate actions to take. The Protection Policies consist of a Preservation Policy, which preserves the data that is under destruction, and a Deception Policy, to obfuscate the true nature of

the system. For each of the Monitoring and Protection policies, an associated service actualizes the rules defined by the policies.

Patterns, Characteristics, and Conditions Of Deception

The deceptive system must identify unauthorized data destruction events to respond deceptively. Several prior works in misuse/intrusion detection may identify common unauthorized data destruction patterns. Work that can help identify unauthorized data destruction is given in Section 2.5.4.

Some write requests for specific files may always be unauthorized and the monitoring should identify when such requests occur. Some of the integrity monitoring methods can help identify unauthorized modifications to trigger deceptive responses. The conditions for a deceptive response may consist of black/whitelists where the blacklists provide a list of conditions, such as files or locations on a storage medium, where there should always be a deceptive response or a whitelist where the system should never inject deception. The rationale for a whitelist could be trade-offs in time/space, or there may be conditions where deception is undesired, such as requests from high privileged processes. Whitelists may also decrease the effectiveness of the deception if the attacker can modify her attack to take place within a whitelist directory or if she can whitelist her destructive actions.

Other conditions and events may also modify the deception to support the goals. For instance, if storage space is abundant, the system may preserve all I/O that modifies files and allow the users on the system to operate without impedance. If storage is scarce, the deceptive strategy may adjust to impede writes that appear destructive.

Deceptive System Functionality and State

The Protection Policies should provide adequate defense and deception necessary to deceive threats and protect the truth regarding the “system functionality”

or "system state" [7]. The deception may consist of any combinations (or multiple combinations of) simulations/dissimulations. A single simulation/dissimulation pair may not be sufficient to provide protection. Further, information regarding the true system functionality and state may also be protected by other security mechanisms, such as access control or integrity monitoring, but an emphasis is given to deceptive strategies in this dissertation.

Modifying the way I/O operates to support the deceptive goals requires that the adversary cannot uncover the true system functionality of overwriting a file by manipulating the adversary's perception through denial, misdirection, or confusion [7].

Examples to deceive the system functionality includes masking the presence of the services relating to the Monitoring or Protection Policies. A deception that protects that system state may consist of a mask that hides the internal state of the system.

Deception regarding the system I/O exploits the confirmation bias of the adversary's assumption that overwriting data is an effective way to achieve data destruction. Note that all of the assumed threats use data overwriting to destroy some portion of a file (or metadata) on a file system. The deception alters the overwrite functionality of the system by preserving the data under destruction while reporting to the adversary that the overwrite is successful.

An inconsistent deception for I/O may report that an overwrite is successful, but upon review, the adversary discovers that the overwritten data is not on the storage medium. Both strategies for I/O are viable, but the advantages/disadvantages require discussing implementation details, which are given in Chapter 6.

The state of the system should appear weak to a data destruction attack to deceive the adversary into using methods that DecMS is capable of handling correctly. The deception can take the form of decoy data preservation services that are located within the system under protection, without the attacker realizing the existence of DecMS. Further, the impact of the attacks should also be convincing to an adversary. As mentioned previously, a deception could convince the adversary that her data

destruction attacks are effective in destroying data by showing that the destroyed files are no longer available.

Several services are necessary to support the goals of the deception. The Observation Policy and service mechanisms watch the users within a computing system for destructive actions. Second, data preservation services are necessary to meet the first goal. The Analysis Policy and service identify data destruction and the Preservation Policy and service preserves the data from potentially malicious destructive actions. The data preservation service can incorporate one of the many preservation methods systems from section 2.5.2.

If the services that are necessary to support the goals degrade system performance, the attacker may uncover the presence of the deception. Deception regarding the system activity may help obfuscate or deny the presence of service activity within the system.

Specifically, there may be noticeable performance delay that occurs only during data destruction. An adversary may conduct a statistical analysis to determine if there are hidden services that are not visible from within the operating system.

The presence of the monitoring may be hidden from the users or may be isolated from the system under protection. There are several viable locations for the monitoring service, which are described in Section 4.1.

The system under protection may have mechanisms in place that attempt to preserve user data. However, the data preservation within the system under protection may be disabled after the attacker has compromised the system. If the attacker disables protection services, the attacker may perceive that the system is vulnerable to unauthorized data destruction attacks. However, the perception of the system is incorrect if the attacker is denied access to information that indicates that a protection system exists outside the system under protection. The attacker's false perception of the system state helps support the first goal of gathering information about attacker's targets for unauthorized data destruction.

Simulation and Dissimulation

Below are some simulation/dissimulation strategies for DecMS. The presentation is left at a high level as it is necessary to provide implementation details to describe the strategy adequately.

Mimicking The adversary's confirmation bias is that the system should behave based on previous interactions with a similar system. A viable strategy is to configure DecMS to behave in a indistinguishable manner to a system without DecMS. There are several challenges to mimic a system with the monitoring and protection services. If resources are shared between the services and the system under protection, then the attacker may observe delays at unexpected times.

For example, DecMS could show the results of a successful data destruction attack to convince the adversary that the attack was a success. The reality is that data is preserved elsewhere.

Inventing Rather than mimicking a system without DecMS, the alternative strategy is to invent plausible causes of discrepancies. Some services, such as preservation or analysis, may require additional delays that would not appear in a system without DecMS. Inventing a cause of the delay, such as a networking delay or waiting for a shared resource to become free may help deceive threat.

For instance, DecMS could invent arbitrary delays during benign writes to hide true delays that caused by a preservation service. Inventing hardware or OS related faults could also impede an attacker when data destruction is taking place.

Decoying If inventing is not convincing, then decoy services could lead an attacker away from the true services. Several decoy services may purposefully be visible or partially visible to an adversary. An attacker who disables or circumvents the decoy service could fail to investigate the true service.

For example, a decoy preservation service can be enabled on an operating system to show that a system has protection against unauthorized data destruction. However, a service outside of the operating system, such as a hardware component, could provide the true preservation.

Masking Masking the true configuration of the system could be achieved through isolation techniques in addition to more deceptive techniques. For instance, DecMS can insert deceptive responses whenever the adversary attempts to uncover information regarding the services of DecMS.

Repackaging If masking whole or parts of DecMS are difficult, then repackaging can help in deceiving an adversary. DecMS can be repackaged to appear as an innocuous service that is unrelated to the goals of DecMS. For instance, a driver or kernel module could be repackaged to support the goals of DecMS. The attacker may not bother to verify that the kernel module is different from her preconception because of her confirmation bias.

Dazzling If the repackaging is not convincing to an adversary, then dazzling the adversary is a viable option. For DecMS, dazzling may consist of producing several plausible instances of DecMS. The dazzling strategy should make it difficult for an adversary to identify the correct DecMS. Further, the risk should be high for an adversary who interacts with the incorrect DecMS.

3.2.5 Feedback Channels and Monitoring

The defender uses the feedback channels to monitor the actions of the adversary [7]. To protect against unauthorized data destruction, the feedback channel should determine when a destructive action is taking place to react accordingly. The reaction, as mentioned previously, should preserve the data under destruction to meet the first goal. The defenders should also react when the adversary is attempting to fill the

disk with meanless data to disrupt the system under protection in a way that may negatively impact users of the system. The disruption may be in the form of halting the system until sufficient storage space is available. Under such circumstances, the attacker may be aware of the deception so the adversary may react with a counter deception to potentially disrupt or impede the attacker.

The feedback and monitoring channel depends on the implementation of the deception. However, to protect against unauthorized data destruction, the following actions should be monitored to support the deception.

- Identify writes that appear to be destructive
- Identify writes to any sensitive files that should preserve, regardless of the information that is written.
- A mechanism to determine the state of the system to adjust strategies as necessary.

Identifying destructive actions relies on existing detection mechanisms and thus inherits the possibility of false positives and false negatives. As with other security monitoring tools, the accuracy may not be perfect. There is a non-zero chance that a benign user's write appears destructive. Further, the monitoring and feedback channels may also degrade system performance, which also affects benign users.

3.2.6 Risks and Countermeasures

The final step of planning the deception is to identify the risks and the potential impact on the adversaries and benign users on the system. The risks and impact are then examined to determine if the risks of using deception are worth the benefits.

As mentioned in [7] there are risks associated with changes to the system that affect the adversary and the benign users or computing services on the system under protection. The effort necessary to destroy data under the proposed system is quantified, but additional experiments with real attackers and systems are left for future

work. In addition to performance impact, real users may become confused when misinformation is presented. The confusion is subjective to the user who consumes the deception and quantifying the confusion is also left for future work.

Several experiments will quantify the performance and degradation of the specific deceptive strategies and systemization of the deception. The increase in write latency and the decrease in throughput are viable measurements to quantify the impact of the deceptive system. Likewise, the same measurements can also indicate the impedance on the adversary. Other measurements, such as the percentage of files preserved will quantify the accuracy of identifying destructive writes and the ability to safeguard the files under destruction.

There are several viable strategies to deceive the attacker. The specific strategies to deceive the adversary depend on the type of system and the amount of risk the specific system may take. Hardware, software, virtual machines, and networking solutions are viable places that may be modified to support the goals of the deception. The design choices for implementing the deception, the types of systems that are resilient to the risks introduced by deception warrant an in-depth discussion. Section 3.3 describes the types of systems that may support the risks of using deception and Section 4.1 and 4.2 details the viable design choices to meet the goals.

3.3 Cost of Deception

While providing benefits, deception comes at a cost. The system defenders must support additional computing systems to manage, deploy, observe, or analyze the effectiveness of deceptions. Depending on the implementation, additional computational overhead and latency may be observable in the defended system. Since security resources are bounded and often compete with other required services, deception may also compete for resources. Further, adding deception into the system may need additional hardware to reduce the overhead associated with deception.

Some of the costs include additional CPU cycles, storage space, and network transmission. For this dissertation, the costs for deploying deception to defend against data destruction are associated with storage medium metrics, which include storage space (for preserving data) and latency/throughput (for analyzing potential destruction). Further, any additional hardware/software introduced to a system increases the chance of failure because of increasing the baseline complexity and reliance on multiple systems. Misinformation may also cause user confusion or cause applications to misbehave. Additional hardware, software, and other technologies such as virtual machines may also be necessary to support the observation, analysis, and preservation of unauthorized data destruction.

Below are performance or stability metrics that deception may degrade. While this dissertation does not optimize for performance, the metrics are important nonetheless to provide insight to help improve deceptive systems in future work.

Latency - The cost, measured in time, of "how long it takes for a given job or piece of work to be completed" [76]. Latency metrics are typically given as a mean or median for a fixed amount of data written to or read from the storage medium.

Throughput "a measure of how much work gets done in a given time interval" [77]. Throughput measurements are a unit of information (e.g., bits, bytes, or kilobits) per a unit of time (e.g., milliseconds), typically taken as a mean or median.

Variation The measurement of the "amount of variation [...] sometimes described as spread or dispersion" [78]. The unit of measurement is a statistic such as a variance or standard deviation for any of the other metrics in this list.

Permanent Error Rate A catastrophic error that cannot be repaired by a recovery mechanism [79]. Permanent errors are detectable, either by a mechanism or because they cause a catastrophic failure. Measurements for permanent error rate include mean time to failure.

Undetected Error Rate An error that remains undetected caused by "inadequacy in the error check facilities" [79]. Undetected errors may cause the storage system to misbehave by corrupting data or causing the system to perform poorly (memory leakage).

Storage Space The measurement of information that occupies space on a storage device.

The impact of each cost will ultimately depend on the implementation. It may be possible, for instance, to use specialized hardware to optimize processing delays. Several types of systems are portable, so power is an important cost factor to consider. However, power constraints are not considered in this thesis and are assumed to be adequate unless a need for additional hardware is necessary.

Deception may also produce misleading information for adversaries on the system. As seen in [58], misleading information may take the form of false errors, which may cause transient, permanent, or silent errors for malicious applications. Transient errors [79] or fault tolerance [80] will ultimately impact the latency/throughput/variation (to analyze the fault) or storage space (e.g., error correcting codes) if the system shares resources with the system under protection.

Additionally, there is a non-zero chance that a benign user may interact with the deception. The deceptive information could disrupt benign users who are unaware of the presence of deceptive information, such as system administrators, to misconfigure the system to overcome a perceived error or performance degradation. Further, adding additional systems increases complexity and testing, thus increasing the risks of bugs and other errors.

The storage space increase to support the DecMS Preservation Goal may increase a variety of ways. For instance, a system may be configured to copy-on-write everything until the storage device is no longer in use. In the worse case, if every overwrite is unique, then the additional storage space necessary is $O(n^k)$ where n is the size of the drive and k is the number of complete overwrites of n before the drive fails or is

no longer in use. The above scheme is not practical; a more reasonable solution is to reuse portions of the storage devices when preservation instances that are no longer useful.

3.3.1 Types of Systems

The applicability of deception, given the costs above, may not be suitable for some systems. Introducing additional latency may be unacceptable for certain computing services, while perfectly acceptable for others. For instance, the additional overhead introduced by analyzing and injecting deception on a system primarily for user media consumption is acceptable given the degradation of storage performance. In contrast, it may be unacceptable to reduce the performance of real-time systems, which has strict deadlines to meet.

Four systems modes (interactive, batch, transaction, and real-time systems [81]) characterize the acceptable/unacceptable use of deception on storage mediums.

Interactive systems are primarily for active users who interact with computing resources continuously. A user is interactive with the system through input peripherals and observes changes through an output device such as a monitor or terminal [82]. Important aspects of interactive systems include low "response time" to allow a user to issue a command and observe output, and meet user's expectations of latency/throughput [81, Chapter 2.4.1]. Latency on interactive systems is a major factor and deception, ideally, should be imperceivable to the end user. Delays are imperceivable to a user when they are below 0.1 s [83]. Current storage devices, such as SSDs, write at a rate of about 500 Megabits per second. At 0.1 seconds, the amount of data written is about 6.25 megabytes. When writing files under 6.25 megabytes, the overhead from deception may be imperceivable.

Similar to an interactive system, transaction systems are designed to process a large number of transactions concurrently. A user interacts with transaction systems through queries (a transaction) with an expectation of a short latency of about one or

two seconds [84, Chapter 1.7]. For transaction systems, data consistency is important. Transactions must guarantee "atomicity," meaning the transactions cannot be in an incomplete state [84]. Thus, transaction systems are sensitive to any failures and faults.

Batch systems, contrasted to interactive and transaction systems, do not have an active user interacting with the systems [81, Chapter 1.4.1]. Instead, batch systems gather jobs, processes them, and save results. Batch systems maximize throughput, CPU utilization, and turnaround time, measured by jobs completed within a time interval [81, Chapter 2.4.1]. Batch systems are not as sensitive to errors compared to other systems. Faults are handled by discarding the results and relaunching the batch job.

Real-time systems provide guarantees that a given task will complete within a predictable time interval and by a certain deadline [81, Chapter 2.4.1]. Some real-time systems face catastrophic failures if the processing does not complete within the time interval [85]. Throughput and latency are crucial measurements for real-time systems. Any changes to real-time systems should not encumber the ability to complete tasks before the predefined deadline with predictable time intervals. A real-time system may disregard inputs to keep up with the time requirements, which is not a feature in the other systems considered [84].

For some systems, deceptive information may be unacceptable if it impacts certain resources. For instance, if a deception scheme impacts the latency for a real-time system, it may be unacceptable. Other systems are flexible for certain impacts caused by a deception. Storage space and throughput, for instance, may be flexible on interactive or real-time systems. A user may not notice if the increase in latency is imperceivable small [83] or mixed with other user interactions. Real-time systems have a narrow set of requirements, and thus storage requirements are known ahead of time. Below, the systems discussed above are compared to their tolerance of degradation for several cost factors.

3.3.2 Cost Impact of Deception on Systems

Given the above characterization of systems and potential costs associated with the use of deception, Figure 3.2 illustrates a comparison of the non-economical costs under three possible levels of impact: Tolerable, Undesirable, and Unacceptable. Systems that tolerate degraded performance are designed to handle the degradation, or the specific metric is not critical to delivering the system's end goals. The undesirable category describes systems that do not fail under instances of poor performance, but the degradation causes inconvenience. Finally, systems that have unacceptable degradation have little to no tolerance for reduced performance and risk catastrophe if observed.

For each type of systems considered in Figure 3.2, resources between the deception and the system under protection are shared. However, for illustrative purposes, each system type does not overlap. In practice, systems may service multiple computing roles, such as running batch jobs on an interactive system when a user is not present[2]

There are several caveats under all of the systems considered. Some real-time systems have soft deadlines rather than hard deadlines [85]. Some interactive systems have a lower tolerance than others. Some interactive systems, such as video editing require high storage throughput for rendering high-resolution media. We assume the typical case for each system based on the general design goals listed in the cited references. Further, the ordering (from left to right) from tolerable to unacceptable is relative to the systems (batch, real-time, interactive, and transaction) under consideration. Further, interactive, transaction and batch systems may also be cloud-based. The cloud systems may increase the allowable latency or variance as networking may introduce additional delays. Real-time systems are not cloud-based as networking delays are difficult to predict unless there is a dedicated communication line. Cloud systems are not explicitly considered in Figure 3.2, but may allow for more performance degradation for transaction systems and interactive systems, specifically, latency and variance.

[2]Seti@Home `setiathome.berkeley.edu`

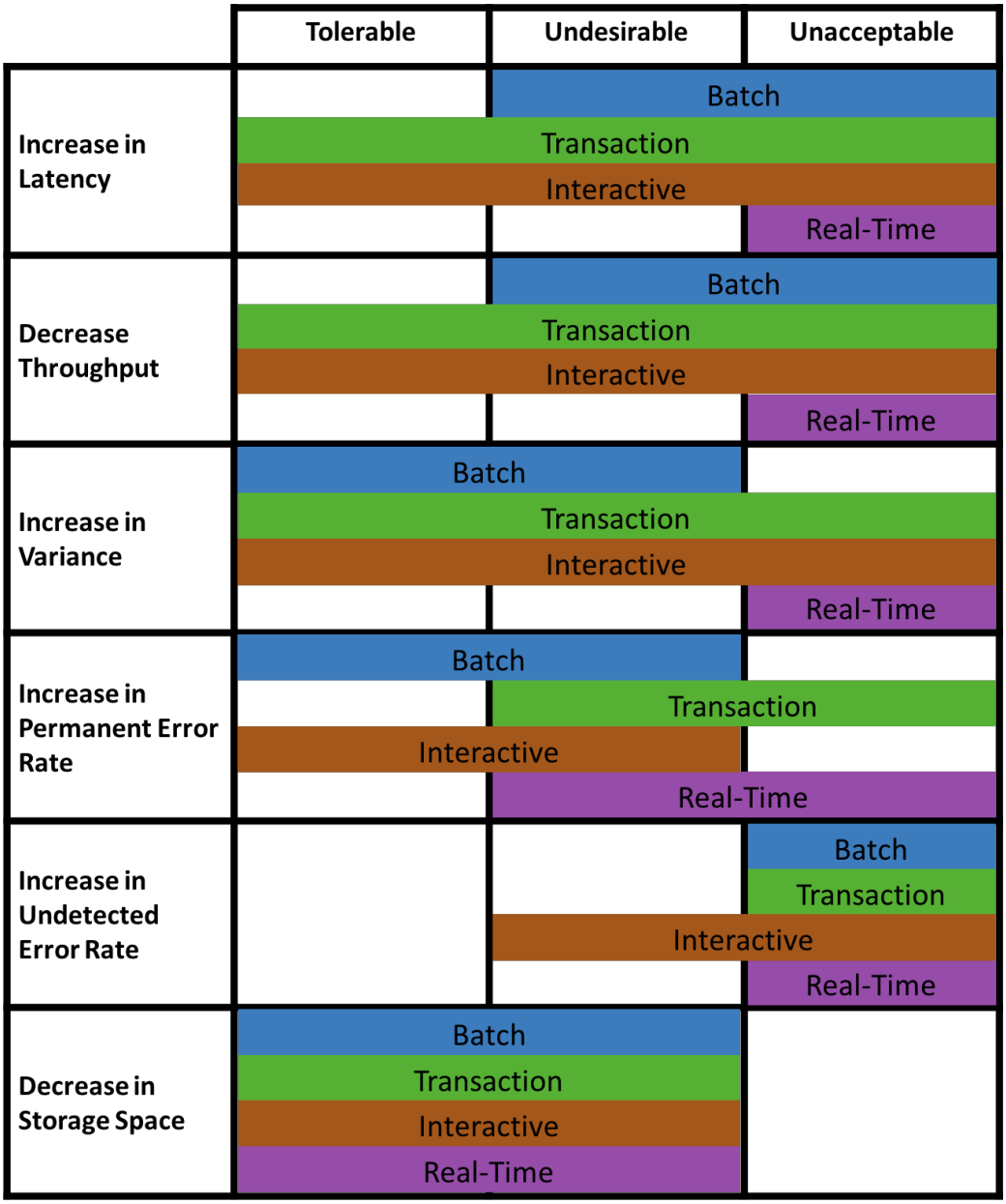


Figure 3.2.: The costs of using deception on various systems.

The increase of I/O latency may not impact batch systems. Batch systems typically queue jobs to run. Latency is, therefore, more acceptable on batch systems compared to interactive and transaction systems. For both interactive and transaction systems, a user is engaged in the system and expects tasks to complete quickly [81,84]. However, any delays below a certain threshold are imperceivable to users [83] so there

are acceptable increases in latency. Increasing the latency for real-time systems is unacceptable compared to the other systems considered. Any additional delay caused by or resulting from deception may cause the real-time system to miss tight deadlines [81].

A decrease in throughput caused by or resulting from deception may be tolerable in interactive systems. On an interactive system, users are typically consuming or producing a small amount of data from the storage medium. When high throughput is necessary on an interactive system, a user can work on other tasks, so the decrease in throughput is amortized. Compared to batch or transaction systems, a reduction in throughput is undesirable. Transaction systems are servicing concurrent users, so a reduction in throughput will cause fewer users serviced over time. Likewise, an important metric for a batch system is the number of jobs completed over time. For real-time systems, it is unacceptable for a decrease in throughput if it impacts the ability to complete tasks by the deadline [81].

An increase in variance may also be acceptable for some systems. Batch systems are designed to service jobs. The workload of batch systems depends on the number of jobs to process. The variance is dependent on system load, and batch systems are expected to handle a variety of workloads. As a live user is present in both transaction and interactive systems, an increase in variance is undesirable. Users operating such systems expect tasks to complete within a reasonable time frame concerning their expectation [81]. High variance means that some transactions or other user interactions may take a longer period than what is expected of the user. Likewise, a high variance will produce a wide degree of possibilities, which are not handled gracefully on real-time systems that have tight deadlines to meet [81].

For interactive systems, an increase error permanent rate is more tolerable compared to other systems. If an error occurs, the user can react and follow up with additional inquiries to resolve the issue or find alternatives. For batch and transaction systems, the errors will reduce the throughput, which is undesirable as the goals of those systems are high throughput. For batch systems, a system job can

be relaunched after resolving the issue [84, Chapter 1.7]. Real-time systems should gracefully handle errors by disregarding the issue to meet tight deadlines or halt the execution to prevent a catastrophic result. Depending on the type of error, a real-time system error may also be unacceptable. The assumption for permanent errors is that they are detectable, so a real-time system should be designed to handle the errors appropriately.

Silent errors are undesirable for interactive systems. Silent errors cause an application or operating system to misbehave in unforeseen ways. Identifying silent errors require manual investigation to find the cause. An interactive system, by definition, has a real user present and may intervene or determine the cause once the issue is observable. It may be as simple as restarting the machine to resolve memory leak issues or may require running diagnosis tools to help root problems like hardware failures. For real-time, transactions, and batch systems, client errors cause substantial issues. For a transaction system, storing invalid or incomplete transactions are unacceptable. Batch systems may report incorrect results for a job which is also unacceptable. Silent errors for real-time systems may disrupt the ability to process information within the deadline, which is acceptable.

Several systems can tolerate some storage overhead caused by deception. For instance, interactive systems storage is flexible. The user may manage the storage by removing unneeded files or saving files offsite. The user has control and can react to the lack of storage so that the additional overhead may have minimal impact. If a deception scheme impacts the latency for a real-time system, it may be unacceptable.

As real-time systems are designed for a well-defined space, the storage requirement should be defined ahead of time, so adjustments could be made to support the additional storage costs.

Other systems are flexible for certain impacts caused by a deception. Storage space and throughput, for instance, may be flexible on interactive or real-time systems. A user may not notice that a task takes twice as long if the increase in latency is

imperceivable small or amortize with other user interactions. Real-time systems have a narrow set of requirements, and thus storage requirements are known ahead of time.

4 ARCHITECTURE AND DESIGN SPACE

4.1 DecMS Integration Location

A computing system has several layers of abstraction to allow programmers and users to achieve their goals without focusing on lower layer mechanics. Rather than fully understanding the complexities of the system, a programmer or user can focus on high-level goals without worrying about the specifics of the OS and hardware. A user's goal may be to write a report and send it to her colleagues. She wants to focus on the contents of her report and not machine code that is required to operate the storage device. Likewise, adversaries who develop malware and exploits and rely on the consistency of the lower levels of abstraction to achieve her goals. As mentioned in section 3.2.3 the adversary's confirmation bias is that the lower levels will operate similarly to her prior interactions. The various layers of abstractions, where the attacker may not check and validate for consistency according to her perception, are viable locations to insert deception.

The deception exploits the adversary's bias that the code (or hardware) at a lower layer is trustworthy in completing an I/O request. As stated in Ken Thompson's Reflections on Trusting Trust:

> "you can't trust code that you didn't write yourself [...] No amount of source-level verification or scrutiny will protect you from using untrusted code [32]."

The above quote applies to adversaries as much as well-behaved users. At some point, the adversary relies on code at a lower-layer of abstraction to facilitate the I/O. The layers in which the adversary fails to check for deception are viable locations for facilitating the deceptive goals.

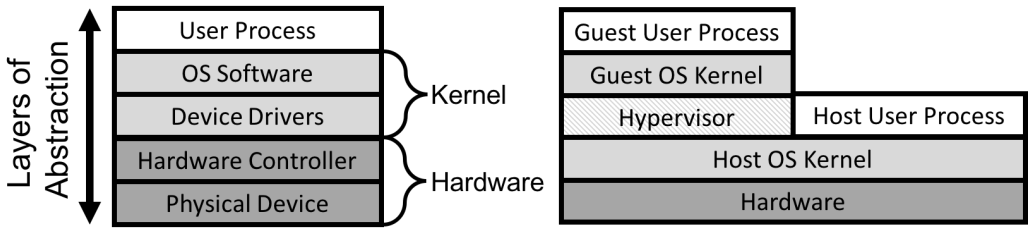


Figure 4.1.: Possible layers of abstraction and monitoring methods for deception, based on [81, Chapters 5, 7].

Figure 4.1 is an illustration of typical computing systems, arranged by layers of abstraction. The figure is based on the description of a multi-layer machine in [86, Chapter 1], where the highest layer of abstraction interacts with lower layers through a translation or interpretation of languages that are specific to each layer. As the focus of this dissertation is on I/O, the translations or interpretations are in the form of I/O requests. For instance, a high layer may request to write an English sentence to the end of a named file. Lower levels translate the name of the file to the physical location of the file on a specific storage medium. At the lowest layer, the I/O request executes on the electronic circuits within the physical hardware.

The layers in Figure 4.1 illustrate the various I/O software layers, inspired by the I/O software layers description in [81, Chapter 5]. Figure 4.1 contains additional layers to describe the hardware below the software and a few optional layers, such as the virtual machine layer and networking layer. The optional layers may be present in systems that use virtual machines or if the I/O request is for a file on a network storage system such as NFS.

The layers in Figure 4.1 are categorized as follows. At the top, user-space is where applications request I/O through a well-defined software mechanism such as an OS system call. User permissions control mechanisms limit the read/write/execute permissions of certain files and locations of the disk. Below the user-space level are the kernel layers, where I/O requests require administrative permission and have full access to the storage medium. However, some of the hardware functionality may be

inaccessible even within the kernel layers. Some functionality, such as hardware diagnosis information requires a special boot sequence that exists outside of the operating system. Furthermore, the hardware controller may be inaccessible without physically attaching a device to program the firmware.

The accessibility of the various layers requires a variety of effort for the attacker to gain access. At the user level, an attacker may be able to overwrite user files. If the attacker gains administrative privileges, the attacker may overwrite any files within the file system. Further, if an attacker has physical access to the device, she may able to reprogram the I/O controller to cause damage to files that may be inaccessible for security monitoring done at the OS level.

The deception may be hidden from view by placing the deceptive components below where the adversary has access. Further, the security tools, such as access control, may also supplement the deceptive components by denying access.

For each layer in Figure 4.1, a short description, advantages, disadvantages, and examples are given.

4.1.1 User-Level

At the highest level, the user level application may be a valid place for inserting deception. An application can be modified to observe for destructive actions and provide mechanisms to protect against the data destruction. There may be several tools within the system for data destruction. For example, some secure deletion applications such as `shred` or `srm` are standard tools used by benign end-users to delete sensitive data from the disk securely. One possible strategy is to simply replace these tools with versions that achieve the deceptive goals. However, the adversary may install her own data destruction tools or use other techniques that do not rely on the binaries on the system. Application level modification should, therefore, be supplemented with other security tools, such as denying the installation of tools and have a trusted procedure to modify and update system binaries. Additional mechanisms,

such as a method to preserve the data under destruction, may be necessary with user-level modification to support the deceptive goals.

Advantage

One advantage of inserting deception at the user-level is the simplicity. The states of the operating system and applications are available at the user-level without having to transform or interpret the information at higher levels. At lower levels, it may be difficult to make decisions and select the best strategy to deceive the adversary. For instance, monitoring for deception at the virtual machine level requires bridging the semantic gap, which is nontrivial.

Even with a simplistic approach, the deception may work well against automated tools or adversaries that do not check for deceptive strategies. It may not be necessary to insert deception at lower levels of abstraction if the goals can be met at the user level.

Disadvantage

It may be difficult to deceive the adversary if she can view the entire deception system. Users and potential adversaries interact with a computing system at the user level. The attacker may exist on the user level (or lower), so the simulation/dissimulation strategies may be difficult if the attacker has full access to the same level where the deception exists.

Several of the deceptive strategies involving hiding. The dissimulation strategy of masking may be particularly difficult to do convincingly, but other dissimulations strategies such as repackaging or dazzling may be more suitable at the user level. The mimicking simulation strategy may be difficult for a defender to achieve if the attacker can easily distinguish differences between a system with or without DecMS.

Nonetheless, even if the attacker can detect the presence of DecMS, she may decide to go elsewhere[1] and target a system without DecMS.

Inserting a DecMS at the user level may require additional security tools. Hiding a deceptive component may rely on tools that operate at a lower level. For example, a process listing tool may not report any processes that are in use for the deception.

Examples

Modifying user level applications to support a DecMS is one possible approach. Prior work, such as the embedded sensors/detectors in Diego Zamboni's work in [87] can help detect destructive writes and trigger the use of a deceptive strategy. The work, however, was not focused on achieving a deception but rather detecting attacks. Other techniques can help support deception but may require additional tools at lower levels of abstraction. For instance, rather than modifying the source code, a user process can be instrumented with deceptive mechanisms through control hijacking [88]. The advantage is that the binaries on persistent storage are not modified, and the evidence of the deception occurs only in memory. Code injection modifies the execution flow that attempts to write to storage and checks for destructive intent. However, the components to insert the deception into the user level may require administrative privileges or exist at a lower level, such as the kernel.

4.1.2 Kernel-Level

At the kernel level, an attacker must gain administrative privileges to view or modify the kernel. There are several viable places to insert deception within the kernel. The OS contains I/O related software that interfaces with user applications that request to read or write to a file on some storage device. I/O related system calls handle the requests. Other interfaces to hardware devices, such as drivers or interrupt handlers, may be instrumented or modified to support the deceptive goals.

[1] Finding an alternative attack may not be an option for a targeted attack.

Advantages

For an attacker to gain a clear view of the abstraction between the user level and the kernel level, the may need to gain administrative privileges. The access control mechanisms on interactive systems typically protect the OS kernel. If the deception requires portions of the system to remain inaccessible, then there is an advantage to placing the deceptive system within the kernel as there exist control mechanisms to deny users access to kernel modification.

The advantage of operating system modification is access to a rich collection of forensically valuable information, such as running processes, timestamp changes, file writes, and active users. The information can be later used for forensic analysis if the destructive writes are unauthorized and malicious.

Disadvantages

While it may be suitable to protect against user-level applications that are partaking in unauthorized data destruction, kernel level deception may be broken if an attacker gains administrative privileges. Security monitoring tools that exist within a kernel [89] are shown to be weak against a persistent attacker who is motivated to circumvent such security mechanisms.

In regards to ease of development, it may be harder to modify a kernel to support a DecMS. Further, kernel code is sensitive to changes and an increase in system instability is undesirable.

Examples

Modifications to the POSIX read, write, or open system calls may be configured to examine the write buffer for destructive patterns before reaching the storage medium. Shared libraries are another viable location to examine for destructive I/O. Prior work in function, library, or system call interposition [65, 90] could be the mechanism

used in monitoring for deception. Wrapping system libraries also provide a defensive advantage to hide the presence of DecMS. Recent work in [53] use custom drivers to observe for write patterns typical in crypto ransomware. In the domain of deception, prior work in honeypots [89] uses custom kernel modules to intercept system calls to gain an understanding of an attacker within the honeypot system. Work in [58] use inconsistent deception in modifying the system calls and show that malware operates poorly within these systems with some degradation to benign applications.

All of the above examples are viable strategies to monitor and react to destructive I/O operations.

4.1.3 Hardware-Level

Another viable layer of abstraction is to insert deception within hardware components. It may be difficult for an adversary who does not have physical access to a device to identify if there are undesired components attached to hardware interfaces and bus lines. Also, it may be difficult to verify that the controllers are operating correctly without physical access to the hardware controller. However, there are some challenges in understanding the semantic meaning of the I/O requests without context or additional information from the higher levels of abstraction. For example, it is non-trivial to determine the specific file under destruction by examining raw disk access only. Mapping low-level disk operations to files are considered a semantic gap problem for disk storage [91].

Advantages

At the hardware level, the deception may be hidden unless the attacker has physical access to the system. Deceptive dissimulations, such as masking, may be hard to unmask at the hardware level. To detect the presence of a hardware attachment, the attacker may have to conduct timing analysis to see if a specific task completes slower than expected. Hardware changes may also have some other effects that an attacker

may use to determine the presence of a hardware device without having physical access to the machine. If the hardware device that is in use for the deception alters data caches, then the attacker can measure the cache hit/miss rate and determine if the machine has additional hardware attached to the storage system. Both timing and cache analysis may be difficult to do compared to the detection methods at higher levels of abstraction. However, the timing and cache analysis may be simpler to conduct than compromising a system administrator account if the deception exists in the kernel or user levels. Nonetheless, masking and mimicking deceptive strategies may work well with hardware solutions if performance and cache hit/miss rates are not affected.

Some deceptive systems may be instrumented without affecting performance degradation. A data bus that is sending information from one device to another may be observed by physically attaching a wire to the bus. A deceptive system could simply relay all the information sent over the bus and preserve an audit log of all requests sent to and from the devices. Further, hardware solutions may increase performance in comparison to the software solutions at higher levels.

Modifying the physical layer to enable deception may outperform solutions that place the analysis within the software. For example, Spensky et al. interpose a SATA controller with a physical device and demonstrates a near identical read throughput compared to the same system without the instrumentation [92]. However, other experiments show a degraded write performance compared to a system without interposing at the SATA controller.

Disadvantages

One disadvantage of placing deception within the physical layer is the additional cost of using hardware. The layers above are all software solutions and do not require special hardware to support the deception. Reprogramming the hardware controllers is a valid strategy to insert deception within the hardware, however, reprogramming

hardware controllers is often nontrivial and may require a special hardware interface to access the firmware. Physical modifications and testing for consistency in hardware may also be more challenging than compared to software solutions.

It may also be necessary to gather other information to determine the best deceptive strategy when monitoring for deception. Simply logging and preserving audit logs for destructive writes may be meaningless for a user, system administrator, or forensic examiner. Because of the hardware semantic gap problem, it may be difficult, or impossible, to determine the specific process or the user who requested the destructive write by examining only the raw disk writes. Gathering such information may require examining higher levels of abstraction.

Examples

A hardware controller or a device connected to the data bus, for instance, may keep an audit of all writes to a storage device. The device may store the audit trail on a write-only device such as an optical disc.

Dione [91] interposes disk operations outside of the "system-under-analysis" with physical hardware or virtualize software. Dione then translates the raw disks requests to high-level data objects through the use of Sleuth Kit. Interposing at the hardware-level may be feasible through similar techniques proposed by Dione to monitor for destructive disk writes and to insert deception when necessary.

4.1.4 VM-Level

Virtual machine introspection (VMI) may also be used to inspect a virtual machine and examine system calls for file modifications [62, 63]. The monitoring for deception exists outside of the system under protection within the virtual machine monitor (VMM). VMI allows a host machine to access the contents of memory within a guest virtual machine, interpose events, and "extrapolate the software state" of the virtual machine [62]. The challenge of VMI is bridging the semantic gap, that is, gather

meaningful information about the guest OS from raw memory access. It is typically assumed that the VMM is isolated from the guest OS and several security tools take advantage of the isolation. For instance, IDS [62] and dynamic malware analysis [69] systems are some of the security tools enabled by VMI.

Advantages

There are several advantages of placing deception outside the protected operating system and within a VMM. First, if an attacker infiltrates a guest virtual machine, it is difficult for the attacker to reach the host. Virtual machines and the VMM are designed to be isolated.Secondly, VMI can be stealthy [63, 69, 93], increasing the difficulty for an attacker to uncover the deception.

Disadvantages

There are also some disadvantages of VMI. Similar to the challenges of inserting deception near the hardware, the challenge with VMI is bridging the semantic gap, i.e., determining which files are changed from outside the operating system, from information stored in system memory. Prior work [94] and an associated open source project [95] address semantic gap challenges. Rekall, a memory forensic framework, profiles modern operating systems and provides indexes to kernel objects as they reside in memory[2].

Placing DecMS within a VMM is not as flexible as placing DecMS within the operating system. One disadvantage is that commercial operating systems frequently update, so the indexes to in-memory kernel objects must be updated for the VMI mechanism as well. However, it is evident when updates are available thus the maintenance of a system with VMI can be managed accordingly.

[2] `http://www.rekall-forensic.com/posts/2014-02-20-profile-selection.html`

Examples

As referenced earlier, there are several tools available to inspect a virtual machine from a VMM. In particular, the Drakvuf VMI mechanism [69] works well because Drakvuf is stealthy and provides an advantage to deceptive strategies that attempt to mask the deception or mimic a system without deception.

4.1.5 Networking-Level

Networking is another layer where deception can be inserted into. Particular to I/O, there is network file system that sends read/write requests for files over the network. A deceptive system can inspect all the read/write requests and identify writes that may be destructive or the deceptive system may create an audit trail of commands that overwrite data, which are reversible if the writes are later determined to be destructive.

Advantages

As with other layers of abstraction, the attacker may not have access to routers or servers where the deception is taking place. The separation between the system under protection and the deceptive systems provides advantages when the deception calls for masking or mimicking strategies.

The other advantage is that interception I/O at the networking level is more well defined compared to analyzing raw disk accesses or VMI in which both require bridging the semantic game.

Another advantage is that user tolerance for latency may be up to 0.1 ms [83], allowing for some delays without significantly harming the user experience. Deceptive strategies that are designed to impede the attacker can be repackaged as networking delays caused by network congestion or other factors.

Disadvantages

A disadvantage of placing deception at the networking layers is that it may be difficult to partition the network so that the attacker is not able to view the deception. That is, relative to the partitioning of deceptive components that require hiding that is within the physical layer. It may be difficult to control the attacker's access certain segments of the networking where the NFS exists but it still may easier to control compared to placing the deception in the user or kernel layers. Another challenge with networking layer deception is the lack of information about active users within the system under protection, which is the same problem when placing deception in the physical layer. Additional tools may be necessary to gather information when reacting to an adversary with deception.

Examples

The work on storage based intrusion detection systems (SBIDS) [52] describe the inspection of reading/write requests on an NFS. The read/write requests are inspected for behaviors that are suspect. A similar strategy can check for destructive write patterns.

4.1.6 Summary

As discussed above, placing deception within each I/O layer has advantages and disadvantages. Some of the weaknesses that appear in several layers are the requirement of using other security mechanisms for protecting the deception. Some of the additional security mechanisms are standard in modern systems, such as access control or requiring physical access to program hardware components.

While the discussion focuses on individual layers of deception, it is possible to place deception within multiple layers of abstraction. For instance, the kernel level or VM level deception can work in conjunction with deception in lower layers, such

Table 4.1: Summary of possible layers to monitor and preserve data under destruction.

Layer	Advantages	Disadvantages	Examples
User	Simplicity.	Hard to hide from threat. Threat can disable.	Embedded Sensors/detectors [87].
Kernel	Protection under access control. Access to system state	Access control to kernel may fail. More complex to modify and test than user layer.	Library and system call interposition [65, 90]. Modified kernel or drivers [53, 89].
Virtual	Solution separates trusted and untrusted system. The adversary must compromise VMM. More stealthy compared to user or kernel layer	The semantic gap problem and maintenance.	Virtual machine introspection [69], virtual disk monitoring [91].
Hardware	Inaccessible to remote threats. May provide performance improvement.	Higher cost compared to software solutions. The semantic gap problem for hardware.	Modified firmware or hardware controller [92].
Network	Separates trusted / untrusted system. A higher threshold of acceptable delays and reduced throughput.	Observing system internal state and information. Not as flexible compared to other solutions	Network interposition or monitoring [52].

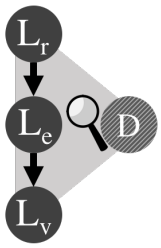
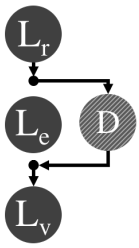
as the physical or network layers. The higher layers have a better semantic view of the system where the attacker operates and can monitor and provide information to the deception in lower layers.

The decision to place deception within certain layers is driven by attacker biases and assumptions of the attacker's capability. It may be sufficient to protect against unauthorized data destruction attacks through the use of a deceptive kernel module if the attacker does not gain administrative access to the system under protection. The other factor in deciding where to place the deception is the impact on well-behaved users on the system. As discussed in Section 3.3, some performance penalty may be acceptable for some systems (e.g., interactive or batch systems) but not others (e.g., real-time systems). In selecting the layer(s) to place the deception, one must consider performance penalties that may not be suitable. However, it may be possible to minimize some of the performance penalty if the dedicated hardware supports the deception. A pure software solution may not be suitable because of the performance cost on some systems, but a solution could split the deception between high layers (e.g., kernel or VM layers) and lower levels (e.g., physical or networking layers). Placing deception in lower layers, however, requires additional testing and costs compared to pure software solutions but the cost may be justified if the data within the system is protected.

4.2 DecMS Integration Methods

At each layer of abstraction, information is passed between layers to facilitate the I/O. At the highest layer, the user (or process) on the system requests to overwrite a file, and at the lowest layer, the hardware writes the specified data to a location determined by higher levels of abstraction. The information may transform between layers into a language that is suitable for the given layer of abstraction. For instance, the data flow at the high layer may be a string that represents a file and a buffer.

Table 4.2: Methods to support the goals of a DecMS

	Expected flow of I/O	Modification	Introspection	Interposition	Wrapper
Diagrams					
Description	L_r – Request I/O L_e – Execute I/O L_v – Verify I/O ⬇ – Information Flow	L_d is a modified L_e that supports deception and executes I/O	The introspection is passive and does not modify flow of information but may invoke actions D monitors for events of interests	The interposition alters the flow of information. D monitors and responds deceitfully to I/O requests	The wrappers D_1 and D_2 alters the information between layers or executes actions to support the deception
Example	L_r – (Process) overwrite file with all zero L_v – (Process) verify file contains all zeros	L_d – Preserve Data then Execute I/O	D – Asynchronous Data Preservation	D – Data Preservation then Execute I/O	D_1 – Alter to cause a write fault in L_e D_2 – Alter return code to "Success"

At lower layers of abstraction, the data flow between layers may be in the form of a smaller segment of the write buffer and a physical address of the storage device.

The deception modifies or observes the data that passes between layers. The deception may support the original requests (e.g., write to a specific location on a storage medium) but may also include additional tasks that are not part of the user's perception of the system (e.g., write audit information for every write request).

There are several strategies to observe or modify data flow between or within the same layer to support a deception. Figure 4.2 illustrates all the strategies to observe or modify I/O flow between (or within) layers. L_r represents the layer when the I/O request originates and the request passes to a lower layer. The layer below, L_e represents the execution of the I/O request. The specifics of the execution is dependent on the layer, but it can be a translation of the request to a language that a lower layer understands. The execution may also break the requestion into smaller segments, buffer the request, or any code execution that is necessary to facilitate the

request at the specific layer. Next, layer L_v verifies the I/O by either receiving the sought information or a status code that indicates the I/O is a success or failure. Note that L_r, L_e, L_v may exist on separate layers or the same layers. For instance, L_r and L_v may be a request and validation at the user layer. For simplicity sake, L_e is assumed to be in a different layer than L_r or L_v but in practice may all exist within the same layer. For instance, L_r a function call, L_e is the function, and L_v is the continuation of the code execution after the function call.

The layer itself may undergo a *modification* to support a deception, *introspection* can passively observe information, *interposition* can disrupt to data flow and route it to a different layer or component within the layer, or a *wrapper* can insert addition layers before and after a specific layer.

Each of the above strategies has advantages and disadvantages to consider. The discussion below explores and discusses the strategies. Each strategy is independent of the layer where the deception is placed, but the specific implementations may have a performance impact. For instance, wrappers may have a performance penalty that is higher than using introspection. The additional layers may cause additional performance overhead compared to a strategy that passively observes the I/O. Further, the specific strategies may require additional security mechanisms to protect the deception. For instance, the wrapper strategy may require additional mechanisms to deny the user access to the new layers inserted between layers.

4.2.1 Modification

A modification to a layer is when a layer is replaced to support the deception. The strategy is analogous to trojan horse applications that contain additional functionality hidden and undesired by the end user. The modified layer may support the functionality of the layer and only insert deception when necessary. In Figure 4.2, the layer L_e is replaced with L_d. The letter d or D is the nomenclature that represents a deception.

To help support the deceptive goals, D may impede requests, preserve data under destruction, gather information regarding the files under destruction, or respond deceptively with false errors.

Advantages

Depending on the layer, it may be simple to replace or modify the layer. For instance, a standard I/O library or drive can be replaced with a customized version to support the deception. If the modification is within a layer that is inaccessible to the user (e.g., within the physical layer), then it may be difficult to recognize the presence of a modified layer.

Disadvantages

If the adversary has access to the modified layer, she may compare the modified layer with her expectation. For instance, a library or drive can be compared with a cryptographic checksum, detecting the presence of the modification.

4.2.2 Introspection

Introspection, based on the definition from [96], "a technique for externally monitoring the runtime state of a system-level virtual machine." Rather than observing the state of a virtual machine, the definition is broadened to include any layer of abstraction. Introspection observes the state of the layers and the information exchange between layers without disrupting or altering the said information or layers. The observation is passive but still exploits the adversary's confirmation bias that the I/O request is not observed closely by the system defenders. Introspection is analogous to an eavesdropper on a channel who observes but does not modify the information exchanged between the communicating parties.

The introspection strategy can help in achieving the deceptive goals by observing for data destruction actions and preserve data before L_e executes. While D itself cannot inject false information or disrupt information flow, D may work with other strategies and trigger other deception mechanisms to react to specific events.

Advantages

It may be difficult for an adversary to observe the presence of introspection because the information flow is not disrupted and there are no modifications to any of the layers. Because of its passivity, there may be no noticeable performance impact, especially if there is no resource sharing between the introspection system and the system under protection. For preserving an audit log of all changes, the introspection method may work well as long as the introspection system can handle the I/O requests without dropping requests.

Disadvantages

There may be a race condition if the introspection observes then reacts to an event. Reacting to an event may require additional strategies such as a modification or interposition at different layers.

4.2.3 Interposition

Rather than passively observing the I/O passed between layers, the introspection strategy intercepts the I/O between layers and redirects the information to different location. In Figure 4.2, the D component receives the information that L_e should receive. After D facilitates the request, with possible deception, D then sends some information to layer L_v to verify the I/O request.

Interposition may help in the deceptive goals. The interposition may inspect all I/O request, properly facilitate the request, and only react deceptively under some

circumstance. If the I/O request is suspicious, D may respond slowly, preserve the data under destruction, or partially fulfill the request to confuse the adversary.

Advantages

By disrupting the information flow between layers, the interposition has more control over the introspection strategy. The interposition may require some tasks to complete before allowing the I/O to continue to other layers. For instance, D may need to write an audit log or copy some information before L_v executes. Another advantage is that the L_e remains unmodified so the adversary cannot simply check L_e for integrity.

Disadvantages

The Disadvantage with interposition is a decrease in stealthiness compared to introspection. The adversary may look for evidence of disrupting the data flow between layers, which is comparatively less stealthy than the introspection strategy which does not alter information flow. The challenge for the defender is to protect the mechanisms that disrupt the flow of information between layers.

4.2.4 Wrapper

Rather than changing the flow of information, the wrapper strategy changes the information that is passed between layers to inject deception into the I/O. The rightmost column in Figure 4.2, D_1 is placed between L_r and L_e. D_1 may simply modify the requests sent to L_e to meet some deceptive goal. Likewise, D_2 is placed between L_e and L_v. The D_1 and D_2 wrappers are either new layers that exist between L_v, L_e, or L_v or they may exist within the layers.

Modifying the information can help facilitate the deceptive goals. The D_1 layer can observe the specific targets, identify destructive behaviors, preserve the data under

destruction, or impede the speed with which attackers destroy files. The D_2 layer can also impede but may modify the values returned to L_v to confuse the adversary.

Advantages

The advantage of wrappers over the other strategies is that it allows L_e to execute while allowing modification of the I/O before and after. An attacker can verify that L_e executes and that the integrity of L_e is valid.

Disadvantages

The wrapper does not have as much flexibility as the modification or interposition strategy. The deceptive strategies are a modification to the I/O between layers rather than modifying how a specific layer behaves. Further, the defenders must protect D_1 and D_2, which could be challenging if the attacker has access to the layers.

4.3 Monitoring Methods and Policy

Section 4.1 describes the various layers of abstraction to place the deception, and in Section 4.2, the strategies to insert the deception are detailed. Each layer of abstraction and the strategies must also consider the location of the adversary and the location of where any processing, logging, and data preservation takes place.

The deception can share system resources with the adversary, however, if the deception is not adequately protected, the adversary may be able to uncover the deception or disable it.

There are several trade-offs to consider. If the attacker is not checking for deception, it may be sufficient to place the deception within the same layer of abstraction to which the adversary has access, referred to as *co-location* henceforward. The attacker may simply check for the deception, but the defenders may also use additional protection or deceptive mechanisms to protect the deceptive components, referred to as

co-location with protection. Finally, the deception may be *external* to the abstraction layer that the adversary has access.

4.3.1 Deception and Adversary Co-location

As the name implies, the adversary and the deception are within the same layer of abstraction. It is assumed that the adversary and deception components have access to the same set of resources. If the adversary and the deception exist within the user or kernel layers of abstraction, then they both share resources such as CPU, memory, networking, and storage space. For physical space, the deception and adversary share hardware channels, power, and other hardware components. For networking, the adversary and deception share networking bandwidth, routing, switches, or other network-related resources. Likewise, if the deception is co-located within the VMM, the adversary and the deception can both view guest OS and control the VM configurations.

There are some advantages in placing monitoring and analysis components within the same layer. First, it may be sufficient to protect against automated attacks that are designed on systems without deception. If the attacker does not bother to check for deception, then her automated exploits may not correctly handle events that may divert the adversary into the unexpected. Prior work in deceptive systems show that disrupting attackers with deception are effective in reducing their impact on the system under attack [58].

4.3.2 Co-location with Protection

Additional protection mechanisms can help aid in the deception. Access control can deny adversaries access to deceptive subsystems or audit logs. Additional deception may also help, for instance, by hiding the presence of interposition and introspection.

Several of the techniques are utilized by malicious software such as rootkits that hide the presence of processes and other tools for system administrators and security monitoring tools. At the user layer, access control can help deny users who wish to read/write/execute deceptive components of the system. At the kernel layer, the components of the system can be set to be immutable so that the adversary cannot change components even if she has administrative access on the machine. At the networking or VM layers, there may exist decoys to confuse the adversary.

However, the disadvantage is that an attacker can look for changes to the system that is out of the ordinary for evidence of deception. Further, the adversary can focus on moving to a lower layer of abstraction to circumvent the deception.

4.3.3 Deception in External Layers

Rather than placing the deception within the same layer that the attacker has access to, the analysis and monitoring can be inserted in a layer of abstraction to which the attacker does not have access. There are several viable options. However, a good understanding of the capabilities of the adversary is necessary. The deception, analysis, and monitoring must be with placed in a layer that reflects the adversary's capabilities. For instance, if the adversary exists only within userspace, then a DecMS at the kernel layer may be sufficient.

One challenge with placing deception external to the layers that an adversary has access to are the various effects that the deception can have in higher layers that may confuse benign users or cause the adversary to raise suspicion.

4.3.4 Identification Methods

4.3.5 Policies and Components of DecMS

DecMS consists of four main components, each consisting of a policy and a method to enforce or execute the policy: Protection, Detection, Deception, and Clean-Up.

Monitoring Policies

The Monitoring Policies handle the monitoring as stated in [7]. The monitoring policies determine what set of events should trigger a deceptive response or other protection mechanisms. For the experimental evaluation, the Monitoring Policies consists of a Observation Policy and Analysis Policy.

Observation Policy

Purpose The Observation Policies should instrument I/O and observe for circumstances that require additional analysis by the Analysis Policy and service. The Observation Policy determines the set of files, libraries, users, or processes that DecMS interacts with. The Observation Policy indicates which user files (e.g., all files in a user's home directory) to protect with DecMS and which to ignore. The Observation Policy also indicates what specific events require additional analysis. The overall purpose is to determine which files or events need further processing under the Analysis Policy. The Observation Policy consists of blacklist/whitelist users, files (directories or specific files), events such as system calls relating to a storage medium or other identifiers that can help identify data items of interests.

Analysis Policy

Purpose The Analysis Policies should specify the methods to identify destructive I/O, define the conditions to deceive a potential attacker, and define other conditions that warrant a defensive action, as determined by the Protection Policies.

The Detection Policy defines the strategy, given the situation, that is best suited to determine if a deceptive response is necessary. The Detection Policy dictates the set of methods needed to detect if a deceptive response is warranted correctly. The Detection Methods can be included signature-based, behavior-based, or heuristic-based detection schemes. The purpose of Detection Methods is to determine if a

Stratagem should be used or to let the system call continue without modification or interruption.

A commonly deployed method to detect known malicious applications is signature-based detection. The core idea is to examine the binary of the malware for key patterns of known malicious code. Malware counters anti-malware software through polymorphic techniques that modify distinguishing features before execution. Another approach that can be used to identify key actions or data objects is through behavior-based detection schemes. The core idea is to observe the actions of some applications for some period. If the actions fall outside the scope of what is expected, fail the application.

4.4 Protection Methods and Policies

The goal of the protection policies is to facilitate the deception and provide general protection or resilience against attacks targeting one of the Parkerian Hexad for data destruction attacks. The Deception Policy is responsible for injecting deceit into the protected system at some given time. It may take the form of a deceptive response for some I/O request or the configuration of the system to remain hidden from the threats.

4.4.1 Deception Policy

Purpose To define the deceptive response given the conditions reported by the Monitoring Policies and insert deceit into the system under protection.

The goal of the Deception Policy is to decide on the deceptive strategy to execute, given the policy decisions and observations in the previous two components. The Deception Methods are simulations/dissimulations that are designed to deceive an adversary.

For example, the selected stratagem may attempt to deceive an adversary, who is destroying incriminating evidence, by pausing the write system call, logging the

incident, and creating a copy of the data elsewhere. The Preservation Policy and service are responsible for handling the data preservation.

4.4.2 Preservation Policy

Purpose To preserve any data that is overwritten by destructive I/O and define the conditions to identify, retain, migrate, compress, or remove stagnant data to optimize storage space utilization.

The Preservation Policy should provide a strategy for tracking deception and anticipate how the users will react to the stratagem. For example, a policy could be defined to log all changes to a file of interest or simply log the final change to a file. Several challenges in designing a preservation policy when storage space is fixed. The Preservation Policy should also determine how long data under preservation should be preserved before moving elsewhere, compressed to save space, or to delete so that space can be used.

4.4.3 Other Protection Policies

In addition to the above policies, several additional policies may exist to help protect data from unauthorized data destruction attacks. Some protection mechanisms, such as access control, may be implicit as modern operating systems use access control. While not explicitly explored in this thesis, it is possible to further protect the data by communicating the data destruction actions to their security mechanisms such as an IDS.

4.5 Summary of Integration Location

Inserting the monitoring, preservation, and the deceptive systems within the hardware may reduce the risks of performance degradation. Software solutions are also possible and are relatively easier to implement compared to hardware solutions. How-

ever, the performance of a software solution may not work as well as a hardware solution, especially if resource sharing is necessary between the system under protection and deceptive system. A virtual machine monitor or networking solutions may also be viable options. VMI may monitor for adversaries and insert deception during appropriate opportunities. Similarly, network interposition may also monitor for destructive changes to network storage mediums and insert deception when necessary.

Some of the solutions outlined above allow for the deception service and components to be isolated from the adversary. As the assumption is that the attacker does not have physical access to the compromised system, she may not be able to identify hardware or networking solutions that support the deceptive components. Denying the adversary from the truth is critical for some deceptive strategies and isolating the deceptive service and components is an advantage over some architectures. The deceptive service and components may also exist within the system itself, which provides several benefits, such as having access to the operating system internals, but it may be difficult to remain hidden from the adversary.

Each of the designs has strategic advantages and disadvantages. As shown in the top of the Figure 4.1, placing deceptive service and components near the application level help preserve files under unauthorized destruction but at the cost to stealthiness (if the attacker can examine the layer with the deception). Placing the deception in lower layers of abstraction, such as within the hardware, may increase stealthiness, but the deceptive system complexity increases as it is necessary to translate raw disk writes to files as the policies may define specific write patterns as destructive for certain types of files. Further, selecting the layers to insert deception may also depend on the deceptive technique, which is guided by the type of adversary that the defender faces.

5 PROOF OF CONCEPT DESIGN

5.1 Desired Traits

In the previous chapter, several design choices and architectures to support deceptive goals are detailed. For the proof of concept, two prototypes are selected based on the goals of this dissertation. The DecMS-Kernel proof of concept uses a kernel module to hide the presence of DecMS and reactively preserves the files from destruction before the writes occur. The DecMS-VMI proof of concept builds on the experimental results of DecMS-Kernel by placing the deception and preservation outside of the system under protection. Below, a discussion for each of the goals that lead to the design of the prototypes is provided.

5.1.1 Identification

Trait The identification method should accurately identify files that are under threat of unauthorized data destruction and provide information of forensic value, such as timestamp information, active running processes, and the method that triggers the data destruction.

The Monitoring Policies and services should provide a list of files that can easily be understood by a system administrator. Some of the low-level preservation mechanism may preserve files by preserving raw storage blocks that may need additional processing to be easily understood by a system administrator. The engineering challenges in developing a hardware-based DecMS motivated the decision to use DecMS-Kernel and DecMS-VMI as the proof of concept systems in the experimental evaluation. DecMS-Kernel has a full view of the system under protection, so it is possible to log the desired information. Likewise, for DecMS-VMI, the proof of concept relies

on existing software to bridge the semantic gap. Capturing file-level and forensics information is possible through VMI.

5.1.2 Impedance

Trait The impedance should provide sufficient time to adjust the deceptive service or preserve a file under destruction and have an acceptable impact on the user. Further, the impedance should be designed with the DecMS Preservation Goal to preserve the data under destruction without compromising data integrity and authenticity.

The wrapper method from Section 4.2 is capable of halting the flow of an I/O request to allow the Analysis or Preservation services to complete tasks. The modification method is practically challenging if the source code for applications, or the operating system, is not available. The introspection method requires additional mechanisms to halt the I/O request. The interposition circumvents the valid layer(s) to facilitate the I/O request and may require an implementation to service the I/O when benign writes occur.

5.1.3 Preservation

Trait The preserved data should maintain its integrity and should not be modified by the adversary after preservation.

The Protection Policies and service should hide the presence of the file or place the file in a location that is inaccessible. Two approaches are selected: (i) threat collocation with protection and (ii) external layer to the threat.

For DecMS-Kernel, files and the preserved data relevant to DecMS are hidden from the process list along with the kernel module. For DecMS-VMI, DecMS is not accessible from within the VM. All preserved data is placed outside of the VM.

5.1.4 Reduce

Trait Demonstrate the ability to reduce the effectiveness of threats that lead to a loss of security elements caused by unauthorized data destruction.

The prototypes should be able to protect against instances of the threat space from Section 3.1. In the following sections, instances of threats are given to evaluate the effectiveness of the proof of concept instances in enhancing data preservation.

5.2 Threat Instances

In the following section, several threats instances are detailed and are later used to evaluate the DecMS prototypes.

5.2.1 Ransomware

Ransomware renders user files on a system inaccessible until a ransom payment is received. Typically, the malware encrypts user multimedia files that are NRA, but may also target backup files [97]. The malware operator will leave behind a message instructing the user to send a payment in exchange for file decryption. A well-designed piece of ransomware can encrypt user files using a public key scheme to minimize any possibility of decryption without payment [98].

Some widespread strains of ransomware in 2016, such as *TeslaCrypt* and *AlphaCrypt*, use symmetric encryption. TeslaCrypt and AlphaCrypt use AES with a local, randomly generated key, which is sent to a server while the local copy of the key is deleted. Variants of this program can be thwarted by retrieving the key locally or intercepting the key in communication [99]. More sophisticated ransomware, such as *CryptoLocker*, *CryptoWall 4.0*, *Samas*, *TorrentLocker*, *KeRanger*, and *Locky* encrypt the AES symmetric key with an RSA public key to avoid locally storing or communicating the AES key in an exploitable way [97, 100–104].

Name	MD5
Destover	`2618dd3e5c59ca851f03df12c0cab3b8`
Shamoon	`d214c717a357fe3a455610b197c390aa`
Shamoon 2	`2cd0a5f1e9bcce6807e57ec8477d222a`
Stonedril	`0ccc9ec82f1d44c243329014b82d3125`

Table 5.1: Wiper Malware samples for evaluation

To evaluate the effectiveness of DecMS-Kernel against ransomware, a ransomware-like application, based on the methodology of the `Linux.Encoder.1` Linux ransomware and its OS X variant `KeRanger` [1] was developed[2]. The ransomware statically links the ARMmbed [3] TLS libraries for RSA public key encryption as well as AES encryption in CBC mode [105] [97]. The AES key is stored encrypted via RSA public key encryption and prepended to the files targeted by the ransomware along with the Initialization Vector (IV) for AES. The code to encrypt the test files is based on the `crypt_and_hash` application found in the ARMmbed GitHub [106].

5.2.2 Wiper Malware

Destover, the Wiper Malware that infamously infected Sony computers in 2014, overwrote master boot records with 64 KiB [4] of `0xAA` [107]. Shamoon destroyed data on a storage medium by overwriting with a JPEG file fragment [108]. An attacker may destroy a file by overwriting the entire file or portions of the files that would render the file unusable for some applications [109].

The MBR is an obvious target for Wiper malware [108]. Destroying the MBR or partition table will render the storage medium unusable. The blacklist for our evaluation includes 'e:," "PhysicalDisk1," "Harddisk1" [110], which point to NTFS data structures protected by DecMS-VMI.

[1] The source code or sample of the actual ransomware was not acquired.

[2] Thomas Yurek, an undergraduate student at Purdue University developed and evaluated the experimental crypto ransomware on DecMS-Kernel.

[3] `https://tls.mbed.org/`

[4] KiB is defined as 2^10 bytes.

5.2.3 Anti-forensics

The anti-forensic threats considered include falsified timestamp evaluation and secure delete methods.

DecMS-Kernel Secure Delete Anti-Forensics

Secure delete methods are used to evaluate DecMS-Kernel effectiveness against attacks on availability. Two configurations of `srm` are evaluated. The default setting is based on the secure delete algorithm described by Peter Gutmann [111], which consists of 38 passes:

1. A single pass of 1-bits
2. Five passes of random bits
3. 27 special values defined in [111]
4. Five passes of random bits

Optionally, `srm` has a "less secure" mode that consists of a pass of 1-bits followed by a pass of random bits.

DecMS-VMI Anti-forensic Evaluation

Two anti-forensic techniques are considered on a Windows 7 VM: secure delete and timestamp fabrication. While the design of DecMS may be able to counter other anti-forensic techniques, such as data concealment or obfuscation, we focus on secure delete and timestamp fabrication to cover the entire threat space outline in Chapter 3.1.

The secure delete tools evaluated under DecMS-VMI are listed in Figure 5.2.

Table 5.2: A list of secure delete methods considered in the evaluation of DecMS-VMI.

Algorithm	**Description**
AFSSI-5020	Three passes: random data, complement w/ 8-bit shift, complement w/ 16-bits shift [112]
AR 380-19	Three passes: random byte, random byte, complement of the second random byte [112]
British HMG IS5 (Baseline)	Single pass of zeros [112]
British HMG IS5 (Enhanced)	Three passes: zeros, ones, random data [112]
Canadian RCMP TSSIT OPS-II	Seven passes: Three alternating passes of zeros and ones, then a random byte [112]
DoD 5220.22-M(ECE)	Seven passes: A combination of random bytes, complement of random bytes, and zeros [112]
DoD 5220.22-M (e)	Three passes: zeros, then ones, then random [112] [113]
German VSITR	Same as Canadian RCMP TSSIT OPS-II [112]
Gutmann's 35-pass method	35 passes: data (1-4), fixed patterns (5-31), random data (32-35) [111]
Overwrite with zeros	Single pass of all zeros [114]
Pseudorandom data	Overwrite with random bits [112]
Russian GOST P50739-95	Three passes: Single pass of zeros, then random data [112]
Schneier's Algorithm [20]	Seven passes: zeros, ones, remaining passes consist of random data [112]

5.3 Randomness Classifier

For both DecMS-Kernel and DecMS-VMI, a classification tree is used. This section provides the details for training, testing, and validating the classification tree.

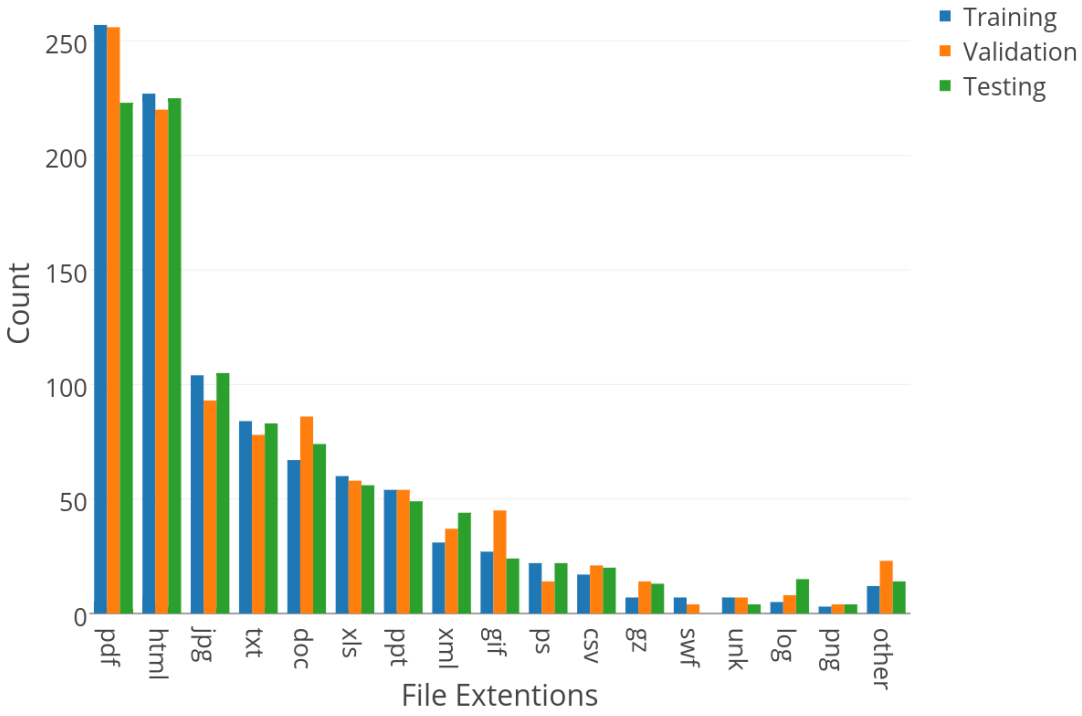


Figure 5.1.: The distribution of files in training, validation, and testing sets.

The classification tree training uses the `scikit-learn` Python library [115], which uses an optimized version of the Classification and Regression Tree (CART) algorithm. We explore the maximum depth parameter, up to a maximum depth of four, which produces classification trees that consist of, at most, four randomness features and four Boolean statements.

The Observation Policy provides the Analysis Policy with: a write buffer, containing the data to be written to a file; a File Handle, metadata information about the file; an offset, the location of where the write buffer should write to the file. For each interposed write buffer, we must consider the *sample size, sample offset, and sample frequency* for system write calls issued to a file.

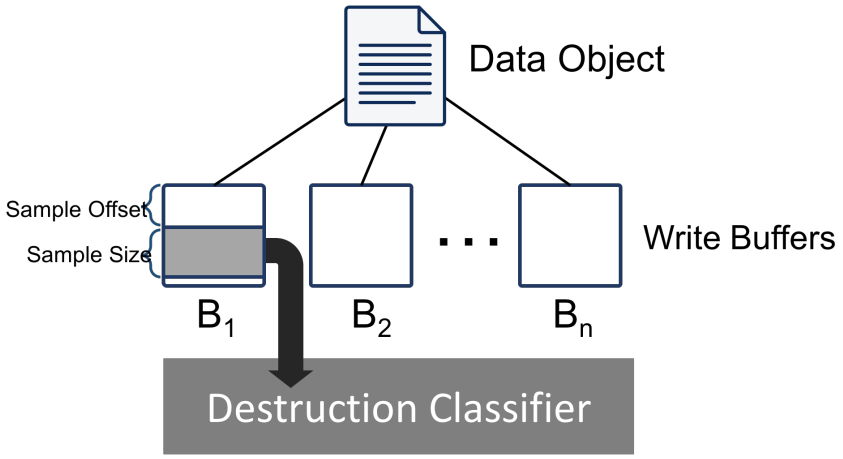


Figure 5.2.: An illustration of write buffer sampling for DecMS-VMI

Let B_i represent the write buffer provided by the Observation Policy. A *sample offset* S_o defines the starting location in B_i for analysis. A *sample size* S_k which defines the amount of data that is inspected per B_i. A *sample frequency*, S_f, defines the frequency to inspect the write buffer B_i. An illustration of the above is shown in Figure 5.2.

The sample offset, in conjunction with the offset provided by the Observation Policy, defines locations of interests within the file. Several files use file signatures [109] and metadata located at the beginning of a file[5].

There are several design choices one must consider for the Analysis Policy. The sample size and sample frequency is a trade-off between accuracy and time: larger samples provide more information to identify if the write is destructive but the analysis latency grows as the sample size increases.

The sample size should be at least the recommended minimum input size for each NIST randomness feature used in our classifier. The maximum sample size must also consider testing the entire write buffer, sub-segments of the write buffer, or combine several write buffers. We constrain the maximum buffer size as the smallest file size

[5]In DecMS-VMI, the file signatures are used to help identify data destruction

in Windows, 4,096 bytes. We validate our classifier by exploring several buffer sizes between 16 bytes to 4,096 bytes and identify a buffer size that provides high accuracy.

The features of our classification tree consist of p-values returned by NIST randomness tests [67]. Our evaluation shows that even with four randomness tests (or fewer), False Postive (FP) and False Negative (FN) rates[6] can fall below 1% and would be acceptable in many scenarios.

We also observed through experimentation that a sample offset of zero provides viable information, such as file signatures. Further, we also set the sampling frequency to only analyze the first write to a file. We show that only sampling the first write is sufficient to detect the data destruction in our evaluation.

We use disjoint training and validation sets for our experimental evaluation. The training set is used to train our classification tree, the validation set explores the optimal parameters for our classification tree, and the testing set evaluates previously unseen data with the optimal classification parameters, which we present in conjunction with other detection mechanisms in Section 6.2.4.

There are two class labels for classification. The *benign class* are writes that are not destructive and the *destructive class* are destructive writes. For the benign class, we use the forensics files corpora detailed in [116]. Specifically, DecMS is evaluated against the Govdocs1[7] data set, which consists of files gathered from .gov domains. The full distribution of training, validation, and testing sets are shown in Figure 5.1.

The *Govdocs1* data set provides ten "threads," each consisting of about 1,000 randomly selected files from the entire corpora. The intention for the "threads" is for researchers to select distinct threads for training, validation, and testing.

For the training and validation destructive class, we use the same files overwritten with pseudorandom bits using the `shred` [72] utility.

The benign class consists of all 991 files in thread0 for training and 993 files in thread1 for validation. For the destructive class, we use the same files overwritten

[6]FP and FN rates often referred to as Type-I and Type-II error rates, respectively.
[7]`http://digitalcorpora.org/corpora/govdocs`

with pseudorandom bits using the `shred` [72] utility. The training and validation set each consist of an equal number of benign and destructive class samples.

5.3.1 Parameter Settings for Randomness Classifier

For our evaluation, we measure the accuracy with increasing write buffer sizes for DecMS. We find the optimal buffer size by evaluating the precision and recall for buffers ranging from 16, 32, ..., 4,096 bytes. We vary the depth of the classification tree from 1 to 4. As the depth of the tree increases, the classification latency increases, but there is a better fit to the data. Our policy defines a sample offset of zero, and the sampling frequency is defined to sample only the first system write call to a file. The rationale is derived from the observation that the secure delete tools we examine, shown in Table 5.2, destroy files sequentially from the beginning of the file to the end of the file. Therefore sampling from every system write call on a file is redundant. The parameter values can be changed either deterministically or can be sampled from distribution to add to the entropy of the protected system.

5.3.2 Randomness Tests

Through our evaluation, we decided on a tree that uses four different randomness tests [67] that provide high accuracy and low computational cost: *Frequency (Monobit) Test, Frequency Test Within a Block, Runs Test,* and *Longest-Run-of-Ones in a Block Test.*

Frequency Monobit Test

The Frequency (Monobit) Test calculates the proportions of zero and one bits in a binary sequence. Each proportion is expected to be about 1/2 if the sequence is to be considered random. A sample size of at least 100 bits is recommended [67].

Frequency Test Within a Block

Rather than comparing the ratio of one-bits to zero-bits over the entire sequence, the Frequency Test Within a Block breaks the sequence up into blocks. For each block, the number of ones observed should be approximately half the length of the block size if the sequence is random. NIST recommends a sample that is at least 100 bits long, a block size of at least 20 bits, a block size greater than 1% of the input sequences, and less than 100 block segments [67].

Runs Test

The Runs Test examines the "runs" within a given sequence. A run is an uninterrupted sequence of identical bits. The test examines if the runs within a given sequence "vary in length as expected for a random sequence" [67]. The minimum sample size is 100 bits.

Longest-Run-of-Ones in a Block Test

The Longest-Run-of-Ones Test is measured within a block of the given sequence and is compared to what is expected for a random sequence of the same block size. NIST provides recommendations for the block size relative to the size of the input sequence. For instance, for a sample length of at least 128-bits, NIST recommends block sizes of eight bits. For larger sample lengths, the block size is increased: block size of 128 for input length of 6272 and 10,000 for input length of 750,000 [67].

5.3.3 Classification Training and Validation

Classification accuracy is measured by *recall* and *precision* in our evaluation. Recall is measured by $\frac{\text{TP}}{\text{TP+FN}}$. A high recall rate is of importance because DecMS should capture all data destruction activities (i.e., a low FN). Precision is measured by $\frac{\text{TP}}{\text{TP+FP}}$. A high precision value indicates a low FP rate, meaning that few benign cases are

classified as destructive. It is difficult to achieve a high precision without reducing the recall or vice versa [117]. While both metrics are of importance, the impact of a false positive classification is preserving a file when a benign write occurs, potentially slowing down a valid write to a file. The impact of a false negative classification is missing a destructive action and not preserving a file under destruction.

Figures 5.3 and 5.4 illustrate the recall and precision of various buffer sizes and max tree depth configurations. The x-axis is the *maximum depth* training parameter and the y-axis is the *write buffer size*. Each cell in the figures represent the precision/recall rate of training and validating the classifier with the given parameters. The general trend is that as the size of the buffer increases, the recall and precision increases. A buffer size of 512 bytes or above is capable of over 0.95 precision and over 0.98 recall. For the tree depth parameter, our results show a trend of increasing recall and precision rate as the depth of the tree increases. Note that even while inspecting small write buffers between 16 to 256 bytes, DecMS is capable of recall rate above 0.9 and precision rate above 0.84.

The box highlighted in red in Figures 5.3 and 5.4 indicates the classification parameters selected for our testing of DecMS. We select a buffer size of 4,096 and tree depth of two because the recall and precision rates on our validation sets are both above 99% and increasing the depth of the tree does not significantly improve the accuracy. Figure 5.5 details the classification tree created by our training. Three features are used in this classification tree: Frequency Test within a Block, Block Frequency (monobit) Test, and longest runs [67]. The false positives we observed in our validation did not follow any discernible pattern and were approximately uniformly spread among HTML, GIF, and SWF files.

5.4 Other Destructive Patterns

The classification process also considers common write patterns found in data destruction. For example, `bleachbit` writes all zeros over a file for destruction,

Figure 5.3.: Recall score on the validation set for increasing buffer sizes and a maximum depth of tree parameters.

whereas `srm` uses a combination of random bits and fixed patterns. The common destruction patterns are an additional check to the randomness test.

DecMS-Kernel first examines the write buffer to determine randomness. If the buffer is classified as benign, DecMS-Kernel then checks the write buffer for fixed patterns. For our experimental evaluation, DecMS-Kernel also checks if the write buffer contains all zeros or all ones.

In addition to the randomness test and common destructive patterns, DecMS-VMI uses file signatures to detect destruction. Overwriting a file signature is a promising indicator for data destruction, as indicated in [23] for crypto ransomware. While some files, such as ASCII text files, do not use file signatures, multimedia files often do.

Figure 5.4.: Precision score on the validation set for increasing buffer sizes and a maximum depth of tree parameters.

When overwriting the beginning of a file, we check to see if the write buffer contains a file signature that matches the file extension [8].

5.5 Design Overview for DecMS-Kernel

Figure 5.7 illustrates the integration method for the DecMS-Kernel proof of concept. A wrapper for several system calls inspects for destructive write requests. Figure 5.6 illustrates, at a high level, the DecMS-Kernel proof of concept. The Observation Policy defines a set of system calls that write to a file. When a process invokes a system call that modifies files, DecMS-Kernel intercepts and inspects which files are accessed. Step (1) shows that a process is calling the `sys_open` system call. Step

[8] An adversary may circumvent this analysis by writing a file signature that matches the file extension then overwriting the rest of the file.

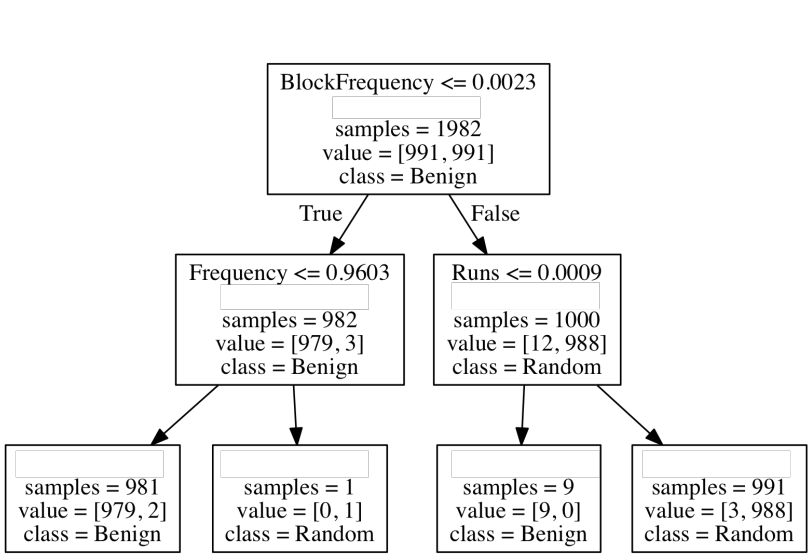


Figure 5.5.: Classification tree for randomness testing with input buffer size of 4,096 bytes and tree depth of two.

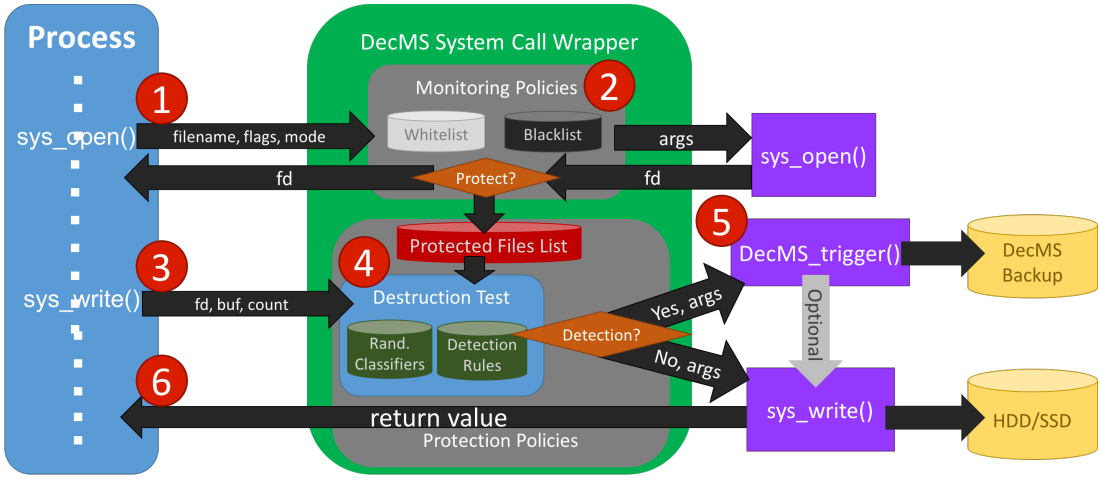


Figure 5.6.: General flow of DecMS-Kernel.

(2) intercepts the `sys_open` system call and identifies if the file should be protected, based on the Observation Policy. DecMS-Kernel places the file descriptor in the "protected files list." At some later point, if the process calls `sys_write` (Step (3)), on a protected file, DecMS-Kernel determines (Step (4)), as defined by the Analysis

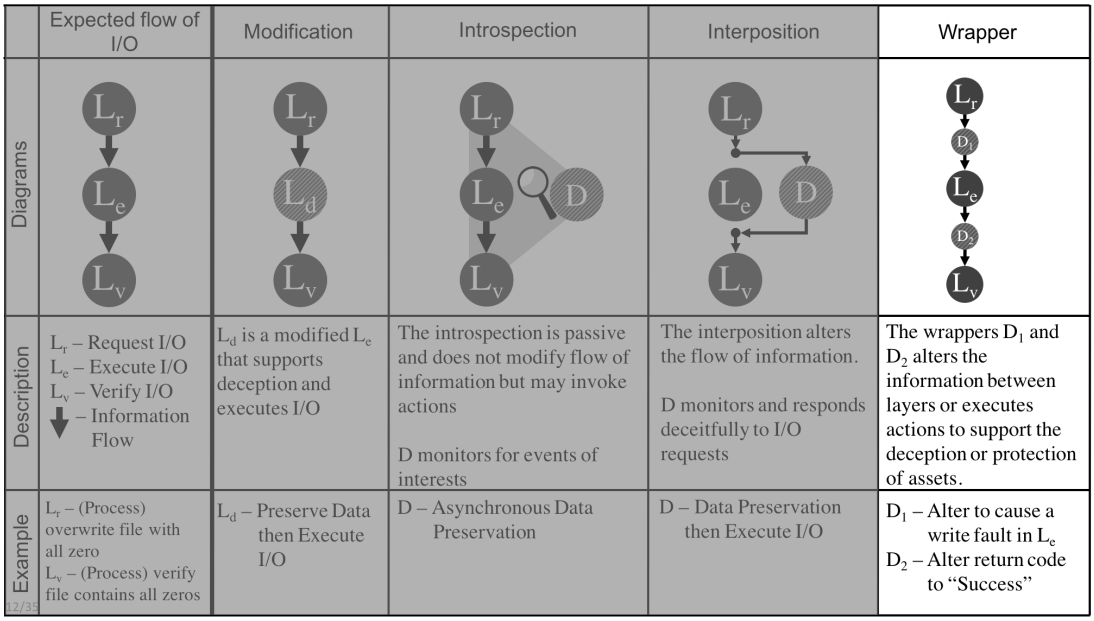


Figure 5.7.: The integration method (wrapper) for DecMS-Kernel, highlighted in white.

Policy, if the write buffer contains random binary data or write templates commonly used for data destruction. If classified as "destructive," DecMS-Kernel triggers the Deception or Preservation, Step (5). The Preservation Service can back-up the data before destruction can occur, and the Deception service can report that the file was successfully written. Step (6) sends a return value to the caller of the `sys_write` call. The return value can appear to look as if the write was successful without additional execution of code.

The design space consists of the placement of DecMS-Kernel, the policies to protect data items of interest, the policies to drive the randomness testing, and the policy on reacting to detection of data destruction. The system call wrapper may also hide the presence of the services and files through the use of several existing rootkit hiding techniques.

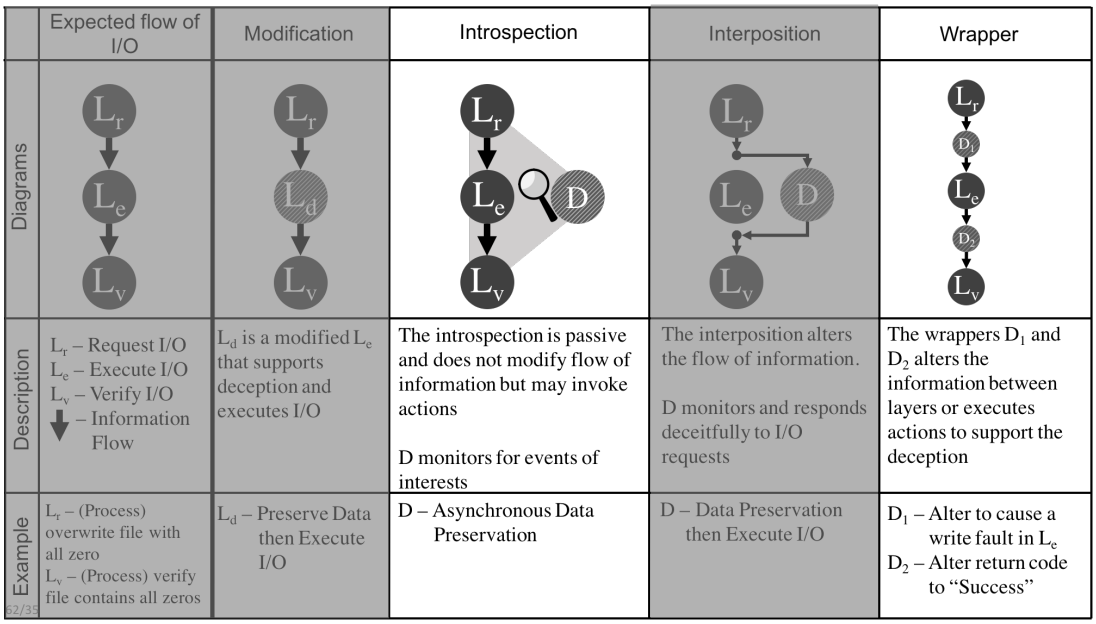


Figure 5.8.: The integration methods (wrapper and introspection) for DecMS-VMI, highlighted in white.

5.6 Design Overview for DecMS-VMI

DecMS-VMI is designed to protect against data destruction attacks and is a supplement to existing security monitoring tools, such as anti-malware or intrusion detection systems, and data back-up/redundancy technology. DecMS-VMI observes persistent storage I/O, through active VMI that wraps several system calls (Figure 5.8), and provides resiliency against destructive I/O. The introspection observes for system state and provides forensically valuable information when data destruction is observed. The high level design of DecMS-VMI is shown in Figure 5.9. The trusted monitoring system, shown on the right, monitors for data destruction on the Untrusted System, shown on the left. The monitoring system is isolated from the potentially compromised "Untrusted System," where a destructive attacker may reside. We assume that the monitoring and the analysis code are trusted and inaccessible from the compromised machine.

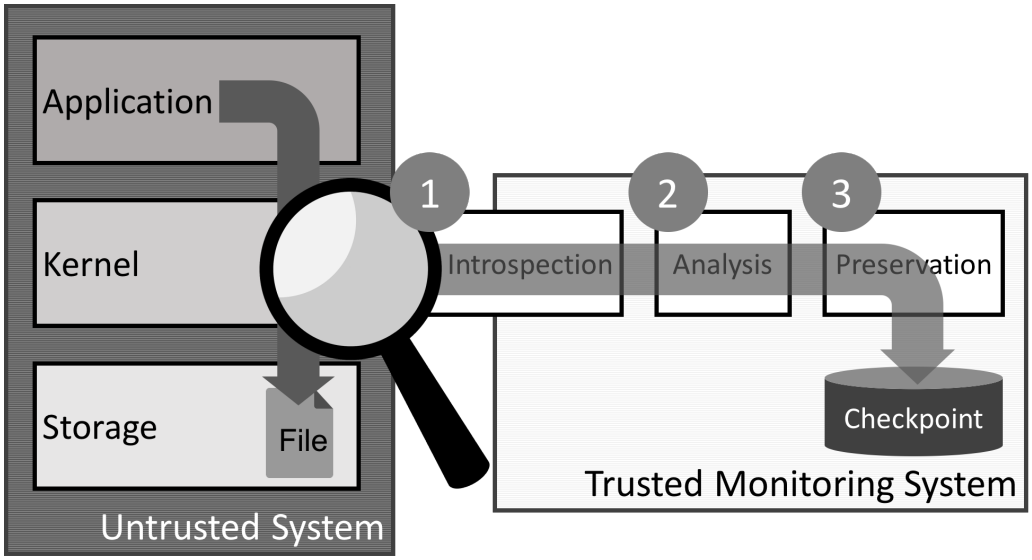


Figure 5.9.: DecMS-VMI interposes storage medium I/O in isolation. If the I/O appears to be destructive, DecMS-VMI preserves the data.

The top of Figure 5.9 shows an application in the Untrusted System that writes data to a file located on some storage medium such as a Solid State Drive (SSD). DecMS-VMI, located on the Trusted Monitoring System, interposes (1) when a file is opened for writing to create a temporary checkpoint. Next DecMS-VMI interposes all writes to an existing file, determines if the I/O is destructive (2), and the checkpoint is preserved (3) if the behavior is suspect. If the application is malicious, it may overwrite a critical file and compromise the file's integrity, availability, or utility [2].

Each of the numbered items in Figure 5.9 are policy driven, to provide flexibility in deployment. The Observation Policy (Section 6.2.1) defines the files/directories under protection, and the I/O related system calls to interpose. The Analysis Policy (Section 6.2.2) describes a set of metrics derived from the interposed system call parameters and defines a procedure to determine if a destructive action is taking place on the live system. The Preservation Policy (Section 6.2.3) describes how to preserve the data under destruction.

5.6.1 Assumptions

To summarize, we make the following assumptions:

1. The attacker destroys data by overwriting a data object, partially or completely, that is located on the compromised machine.

2. The attacker does not have physical access to the storage medium. Ergo, the attacker cannot physically destroy the disk.

3. The attacker may have administrator access to the compromised machine but cannot avoid monitoring of our proposed system [9].

4. The VMM analysis and storage used by DecMS-VMI is assumed to be within our Trusted Computing Base (TCB).

[9] For example, Kernel Object Hooking (KOH), Dynamic Kernel Object Manipulation (DKOM), or Direct Kernel Structure Manipulation (DKSM) [59, 74] could be used to remain hidden from security monitoring.

5. Our proposed system works in conjunction with other security monitoring tools, such as Anti-virus scanners (AVS), and Intrusion Detection Systems (IDS). However, we assume that AVS, IDS, and other security monitoring systems fail to identify the entity that is destroying data as malicious to demonstrate the effectiveness of DecMS-VMI.

6. Our prototype implementation does not consider the case when valid users wish to securely destroy or encrypt data.

5.6.2 Requirements

Preservation: Upon detecting events of interest, a log of the event and the data under destruction should be preserved. The logging and preserved data should be inaccessible to the adversary.

High Accuracy: Detection of data destruction should be accurate with few false negatives (incorrectly identifying a suspicious write as benign). False positives (incorrectly identifying a benign write as destructive) should also be kept low, to avoid encumbering the end user with degraded system performance. We focus on obtaining low false negative rates to reduce the risk of data loss when destruction occurs.

Acceptable Performance: DecMS-VMI should impact latency and throughput for legitimate users as little as possible.

6 EVALUATION

6.1 Kernel-Based Evaluation

6.1.1 Observation Policy and Service

To intercept system calls, we used the Suterusu Loadable Kernel Module Rootkit [60]. Suterusu is an open source modern rootkit designed for x86(_64) Linux 3.X kernels. The rootkit demonstrates several common features, such as hiding the presence of processes, files, and network sockets. The process and file hiding features are used to mask the presence of DecMS-Kernel. Further, Suterusu hooks system read/write calls for logging keystrokes. Care was taken to remove malicious functionality such as the keylogger and remote execution of arbitrary binaries. The design choice of using Suterusu is driven by its ability to hide and ease of extensibility.

Suterusu relies on inline function hooking that modifies a targeted function to transfer execution to another routine [60]. Suterusu provides `hijack_start, hijack_stop, hijack_pause, hijack_resume` helper functions to easily hijack functions.

The following functions are hijacked for DecMS-Kernel. A short description is provided for each hijacked function.

sys_open - DecMS-Kernel keeps track of each file a user has open. The opened file is inspected according to a predefined Monitoring Policies, as shown in Figure 5.6 (2). For our evaluation, DecMS-Kernel places files into the "protected file list" if sys_open is invoked by a user process with a write request. Further, only regular files are placed in the "protected file list." Directory files, block devices, character devices, pipes, sockets, and symbolic links are ignored in our experimental evaluation.

sys_write - When a write system call is invoked, DecMS-Kernel checks if the file is in the "protected file list." If so, it passes the write buffer through the randomness classifier. If the buffer is classified as destructive, then execute Preservation Policy (Figure 5.6 (5)). The policy defined in our experimental evaluation is to save the file elsewhere, with the assumption of sufficient storage space, and allow the `sys_write` to continue after the file is copied.

sys_close - If the file is in the "protected file list," then remove it.

6.1.2 Analysis Policy and Service

Figure 6.1 illiterates the policies to determine if a file is undergoing data destruction and whether the file should be copied. The policy we use to evaluate DecMS-Kernel is to test if the first write to a file is destructive. Two checks are then conducted. The randomness test uses a classification tree and the "common secure delete patterns" to determine if the write buffer contains all ones or all zeros.

Checking subsequent writes to track how a file is changed over time is certainly possible with DecMS-Kernel. However, analyzing the buffer for every system write call on a file may not provide enough forensic value to justify the increased latency, and in the case of a secure delete with multiple rounds, could result in excessive storage space being used for pseudorandom data. However, checking the first write call to a file can be circumvented by an attacker who first writes benign information and then later switches to data destruction. The defined policy should take into consideration whether the potential savings in computational overhead and storage space is worth the vulnerability to a targeted attack against DecMS-Kernel.

During initialization, DecMS-Kernel hijacks all three system calls listed above. One potential flaw is that hijacking is paused to allow the system call to execute. There is a window of opportunity for other threads to execute the system call without the hijacking enabled [60]; thus Suterusu hijacking is not thread-safe. After the

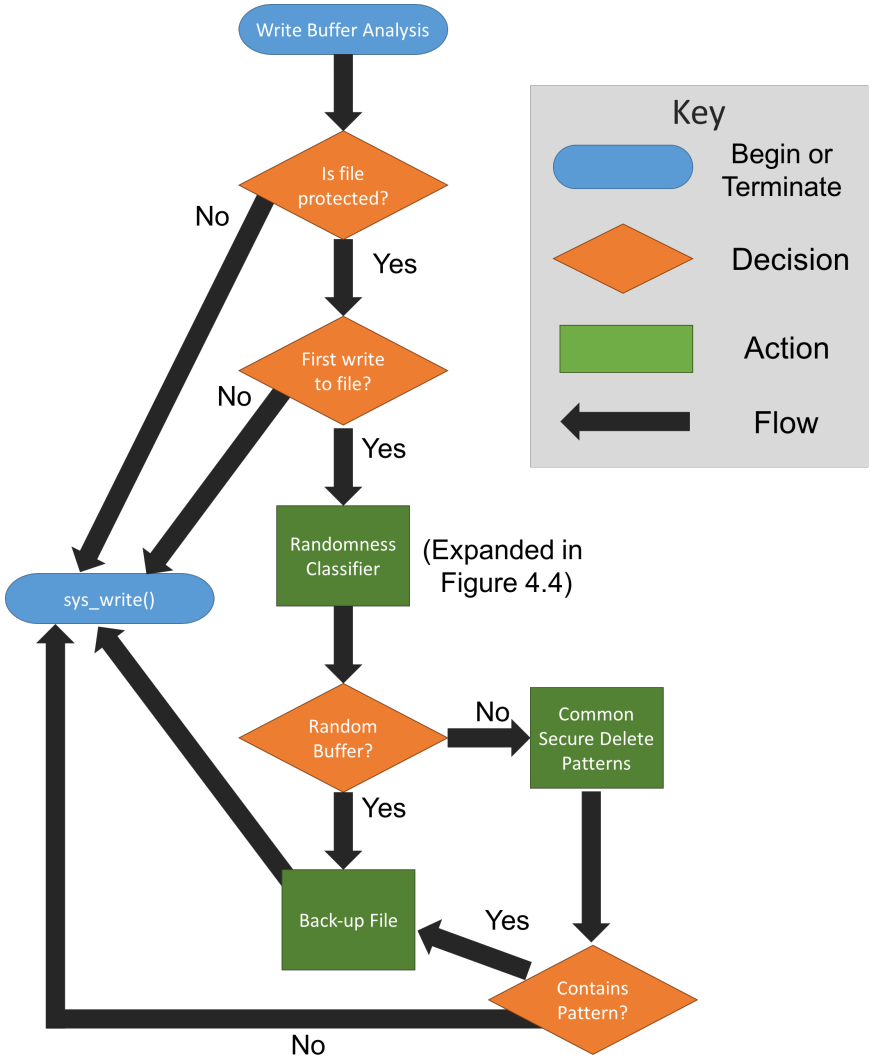


Figure 6.1.: DecMS-Kernel policies to determine if a write buffer is used to destroy data

system call completes, the hijacking resumes. Coppola [60] explains that locks and permanently hijacking the system calls would solve the issue.

6.1.3 Preservation Policy and Service

If a process attempts to destroy a protected file, the file is saved using Redhat Linux System Auditing (`audit`) [118]. Using Linux System Auditing allows for exist-

ing parsing tools to work to extract information generated by DecMS-Kernel. Further, `audit` is easily configurable with trigger actions if the log file is full, the disk is full, as well as sending log information over a network[1]. Another advantage is that we can leverage Suterusu hiding file and network connections to covertly backup data.

Protected files that are under destruction are encoded and sent to the audit daemon. For our evaluation, the data is stored locally and only accessible under a privileged account.

6.1.4 Deception Policy and Service

A denial of the truth is the deceptive strategy explored in this prototype implementation. A masking strategy hides the true nature of the system through a rootkit mechanism. Suterusu hides the existence of the kernel module that handles both the presence of DecMS-Kernel and the preservation. To complement the dissimulation (hiding the real), the simulation (showing the false) is implicit by returning that all writes to a file are successful. The state of the system is also hidden. Files that are associated with the data preservation are not viewable through typical file listings (e.g., ls).

6.1.5 Service Location

All of the services above are co-located with the threats but with additional protection. The assumption is that the attacker is running in user mode and all the protection mechanisms execute with elevated privileges.

6.1.6 Metrics of Interest

For the experimental evaluation of DecMS-Kernel, two metrics of interest are calculated: the ability to accurately classify write buffers as benign or destructive, the

[1] `http://linux.die.net/man/5/auditd.conf`

latency associated with conducting such an analysis on a live system, and the ability to accurately preserve the data under destruction. For the Monitoring Policies, the accuracy is necessary to determine if the system is capable of detecting data destruction. The latency also indicates the effectiveness of the Protection Policies. High latency indicates that some unknown service is present whenever a data destruction action takes place. The preservation service must be able to preserve the entirety of a file before the destruction occurs.

Detection Latency

The detection latency is a measurement of several components of DecMS-Kernel. However, the execution of each component is driven by the system's defined policies.

The components that contribute to the detection latency include:

1. Interception of system calls
2. Determining if a buffer is random or contains common data destruction patterns
3. The latency to preserve the data under destruction

Write buffer interception occurs each time the `sys_write` system call is invoked. However, the randomness analysis only occurs if the write call is applied to a file marked for protection, shown in Figure 5.6 (2), the Observation Policy. Furthermore, the *Protection Policies*, shown in Figure 5.6 (4), could be configured to *always* check for randomness for each write call applied to the protected file. However, the policy we define for the experimental evaluation simply checks for randomness and common data destruction patterns *on the first write to the file* and subsequent writes to the file are not analyzed for randomness. This was done to minimize computational overhead and prevent excessive storage from being allocated to random data. If the first write was classified as destructive, the entire file was moved elsewhere.

Relationship of Accuracy and Latency

In general, the accuracy of the randomness test increases as the sample size increases. However, the latency to conduct the randomness test increases as the sample size increases. While this dissertation does not optimize for performance, the analysis could be further improved by using specialized hardware such as a graphics processing unit (GPU) [119] that is capable of calculating randomness features more efficiently than a software solution. For our evaluation, we measure the accuracy and latency with increasing input sizes for DecMS-Kernel. The minimum buffer size we measure reflects the recommended minimum size for each randomness test feature[2] as defined by NIST in [67]. The maximum buffer size we measure reflects the default block size for Ubuntu 14.04.4 LTS, 4,096 bytes.

6.1.7 Experimental Results

Our experimental evaluation was conducted within a Linux 3.18.27 Ubuntu 14.04.4 LTS VMWare Fusion Virtual Machine with 2GB of RAM and one processor core. The host system is a MacBook Pro with Mac OS X 10.10.5, 2.7 GHz Intel Core i7, 8 GB 1067 MHz DDR3 RAM and a 256GB Solid State SATA Hard Drive.

Latency Analysis

There are two sources of latency associated with DecMS-Kernel: inline function hooking and randomness classification. Inline function hooking intercepts system write calls and latency is observed for each write to the disk. As defined by our Observation Policy in Figure 6.1, the write buffer passes through our randomness classifier and common secure delete pattern checker only on the first buffer written to storage.

[2]NIST recommends 120 bits for Frequency (monobit), Frequency Block, and Runs Tests, and 128 bits for longest run-of-ones Test

Figure 6.2 shows the latency test of DecMS-Kernel compared against the system baseline as well as DecMS-Kernel with only system call interception enabled (without inspecting the write buffer for randomness or destruction templates). The latency samples 500 separate file writes of 4 KiB. The median latency for the baseline is 0.0075 ms, and the median latency of DecMS-Kernel without conducting any classification is 0.015 ms. Thus the cost of each write call interception is about 0.0075 ms. The median latency for DecMS-Kernel of buffer size 16 bytes and a max depth of one is 0.024 ms, which adds about 0.017 ms of overhead. Likewise, the median overhead caused by a classification tree with a maximum depth of two and buffer size of 4,096 bytes (D2B4096) is 0.046 ms, and for the D4B4096 [3] classification tree, the median latency is 0.117 ms.

Note that the costs associated with the randomness classification become increasingly large as the file size increases. The cost to intercept the write buffer eventually becomes much larger than the cost to run the randomness test once. In Figure 6.3, the latency for a 32 MiB [4] file is shown. The cost associated with writes interception with 32 MiB. files is estimated to be about 93.05 ms. As shown in Figure 6.3, the increased latency from the baseline for 32 MiB files is about 60.00 ms.

Figures 6.4 and 6.5 compare the latency of DecMS-Kernel with data backup enabled for 4 KiB files and 32 MiB files respectively. When DecMS-Kernel detects data destruction, the file is hex encoded [120] and sent to the `auditd` for storage. The hex encoding increases the overhead by a factor of 2.95x. The overhead for hex encoding was calculated by measuring the average latency of 1000 10 Mib files for both the baseline and hex encoding only. The usage of hex encoding is for convenience and can be avoided altogether by writing the raw binary values out to the log. The total median latency increase for 4 KiB files when a write buffer is classified as destructive is a factor 19.5. For 32 MiB files, the median latency increases by a factor 20.6.

To summarize, the latency introduced by DecMS-Kernel is about three times the latency of the baseline. Because our policy only checks for randomness on the first

[3] Maximum depth of four and buffer size of 4,096

[4] MiB is 2^20 bytes.

write to a protected file, the latency associated with intercepting the write buffer outweighs the cost to run the tree classifier and common secure delete template matching regardless of the configuration explored in Section 5.3.3. The latency to move a file if the write buffer is classified as destructive increases the baseline latency by about a factor of 20.

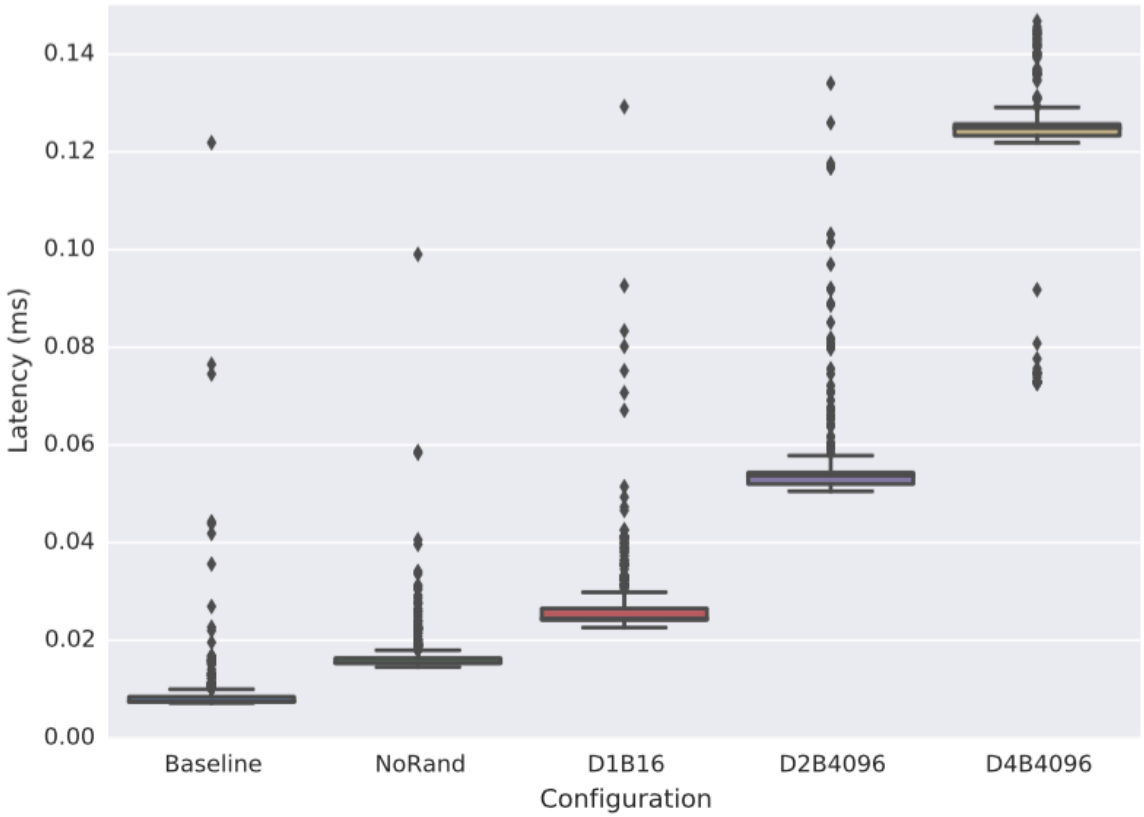


Figure 6.2.: A comparison of real time file latency for 4 KiB files for DecMS-Kernel under various sample sizes.

Evaluation Against Applications

The evaluation is conducted on the `shred` [72] and `srm` [73] command line applications. Of the 993 test files, all were successfully detected by DecMS-Kernel, indicating that the Protection Policies and service are working under the experimental config-

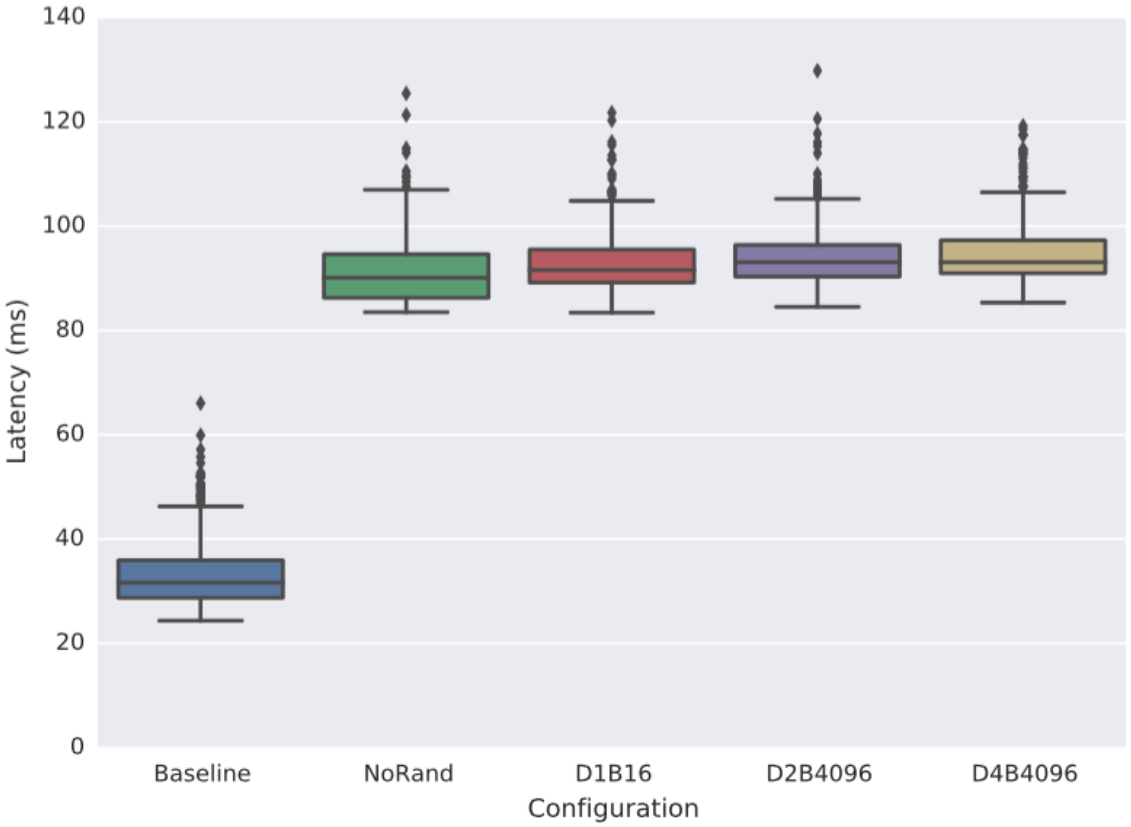


Figure 6.3.: A comparison of real time file latency for 32 MiB files for DecMS-Kernel under various sample sizes.

uration and threat assumptions. DecMS-Kernel is configured to detect if the file is undergoing destruction and a log is generated.

Ransomware Evaluation

We tested DecMS-Kernel by measuring the ability to detect encryption of all the files greater than 4,096 KiB in the Diverse testing set. The results show that DecMS-Kernel is capable of detecting 916/917 of those files. The misclassification of one out of the 917 was caused by the incorrect classification of the random buffer written by the experimental ransomware.

One false negative classification out of 917 provides a significant improvement compared to the system without DecMS-Kernel. However, the loss of even a single

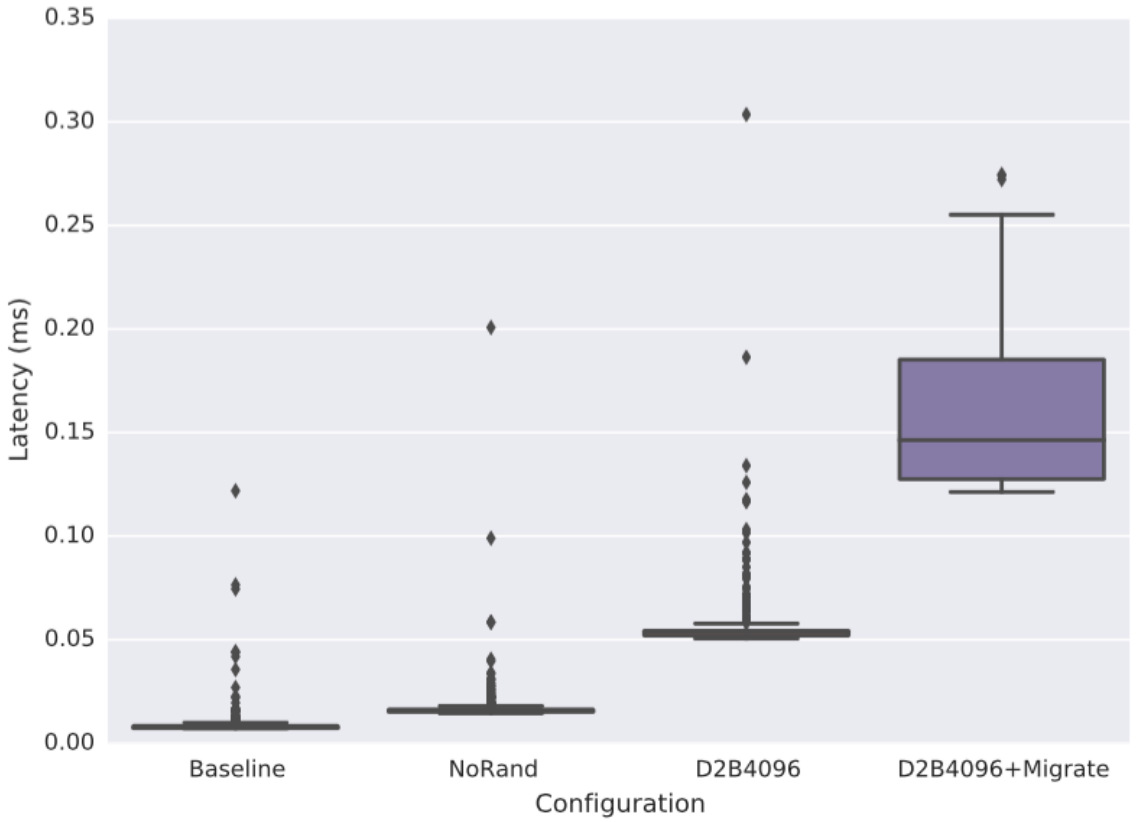


Figure 6.4.: A comparison of real time file latency for 4 KiB files for DecMS-Kernel, with and without the Analysis Policy and Preservation Policy.

file is devastating if the file is not replaceable. The classifier can be adjusted to improve accuracy, but perfect accuracy is not possible. For the DecMS-VMI proof of concept, a file is always preserved (temporarily based on some retention policy) upon opening a file. Even if a destructive write is misclassified, the file may still be preserved.

6.1.8 Discussion

DecMS-Kernel is capable of detecting data destruction with high rates of recall and precision.

We measured the latency of DecMS-Kernel, and our experimental results show that it is suitable for users with low I/O demands. Furthermore, we demonstrate

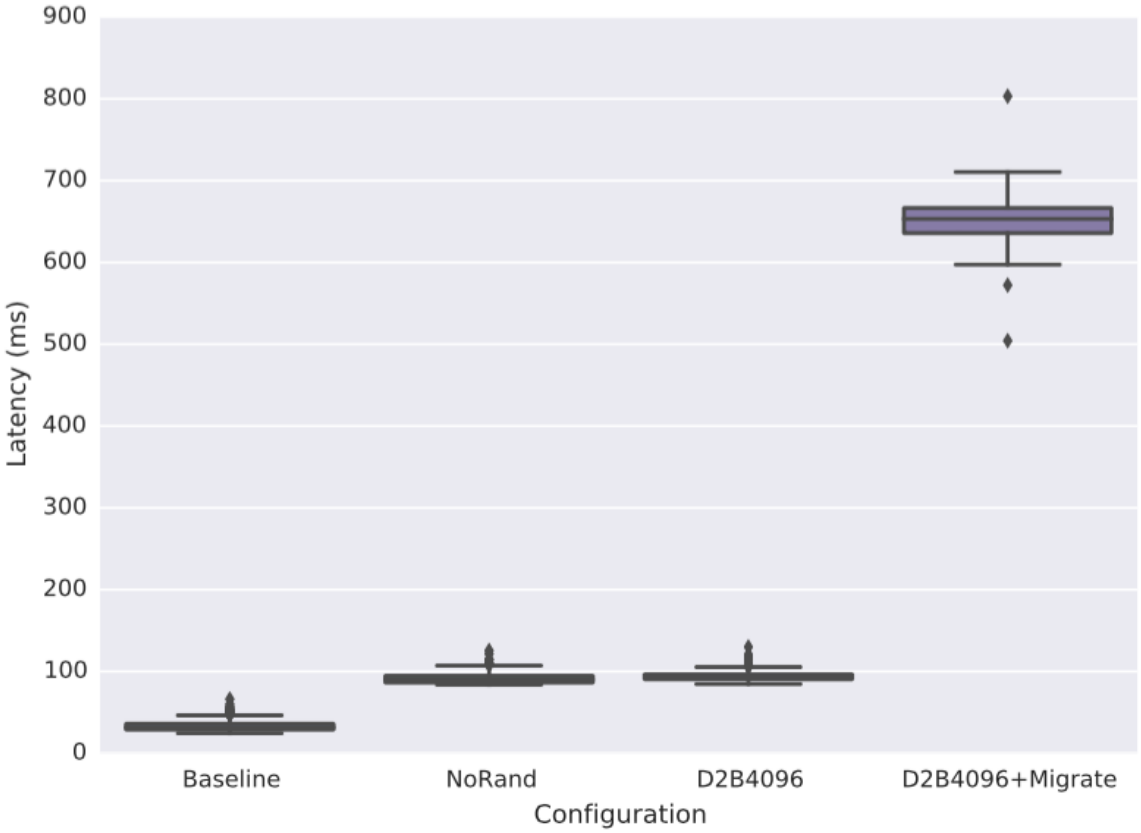


Figure 6.5.: A comparison of real time file latency for 32 MiB files for DecMS-Kernel, with and without the Analysis and Preservation Policies.

that DecMS-Kernel is capable of defending against commonly available secure delete tools such as `shred` and `srm` as well as the ransomware which may use the commonly observed combination of AES for file encryption and RSA public key encryption to encrypt the AES key.

The above results show that the DecMS-Kernel proof of concept is capable of preserving files under destruction and providing a log of targets that a threat may target, satisfying the DecMS Identification Goal, DecMS Impedance Goal, and DecMS Preservation Goal. Two types of threats are shown to be unsuccessful in destroying data, which indicates that it may be possible to reduce the effectiveness of some data destruction methods (DecMS Reduction Goal).

There are several improvements guiding the design choices in the subsequent evaluations of DecMS. For large files (32 MiB in the evaluation), it appears that the latency may not be suitable. Specifically, copying the files to another location is only suitable for small file sizes because of the large latency. Likewise, the large observable latency may indicate to the attacker that the I/O system is doing additional work during destructive actions. The masking/mimicking strategy at the kernel level may, therefore, be unsuitable under the current configuration.

Given the above results, the following design choices were made. First, the process preserving data under destruction should be optimized. An obvious choice is to use a logging or snapshot file system that only saves changes to a file rather than overwriting a file. A logging or snapshot file system will not require copying the entire contents of a file when a destructive write is observed. However, the challenge is hiding the fact that a logging or snapshot file system is in place. The control mechanisms of the file system, to roll back to an older version of a file, should be outside of both the user and threats control. Based on the results above, the following proof of concept places the Monitoring Policies and Protection Policies, and associated services, outside of the system under protection.

6.2 VMI-based Evaluation

Our prototype implementation DecMS-VMI interposes system calls through Virtual Machine Introspection (VMI) and places the analysis and preservation within the Virtual Machine Monitor (VMM), which is part of the Trusted Monitoring System (Figure 5.9). The challenge with VMI is bridging the semantic gap, i.e., determining file activity from outside the operating system. Prior work [94] and an associated open source project [95] address semantic gap challenges, which security monitoring tools [69] use reliably.

Our evaluation of DecMS-VMI protects a Windows [5] VM against data destruction and uses VMI to intercept system calls associated with storage medium I/O. The cost of DecMS-VMI is a decrease in I/O performance. However, we show that the additional overhead may be acceptable in some settings.

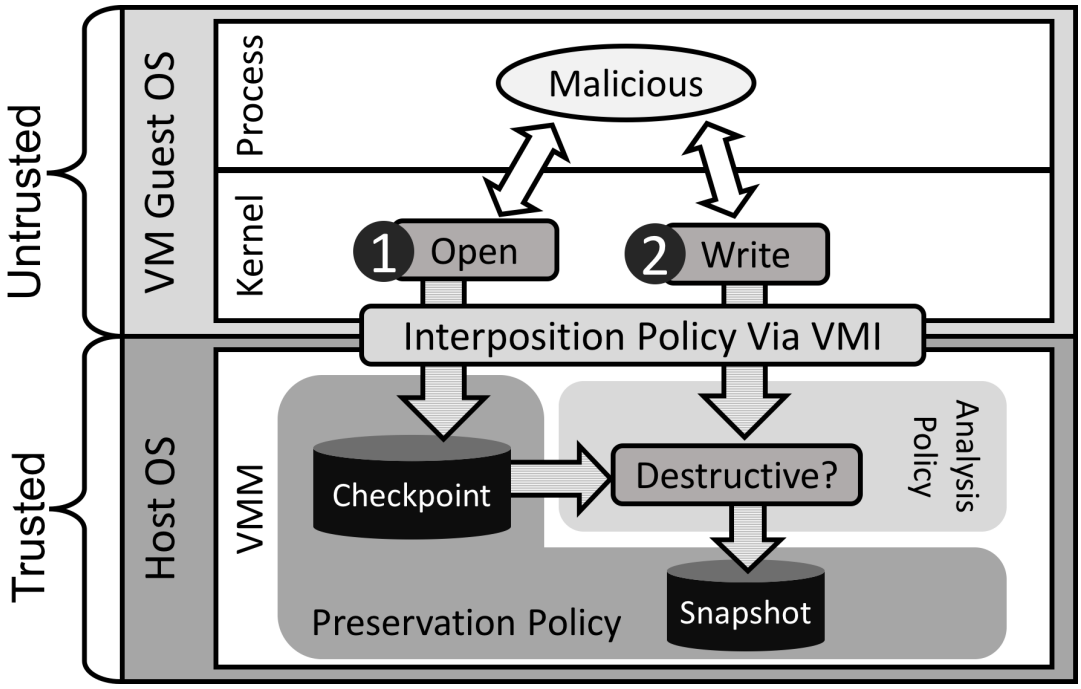


Figure 6.6.: DecMS-VMI examines file modifications and preserves the file if data destruction is suspected.

Figure 6.6 illustrates the high-level design of DecMS-VMI when placed within the VMM. When a process invokes a system call that writes to a file, DecMS-VMI intercepts the system call and inspects the parameters (1). The Observation Policy defines the set of system calls to monitor and a set of files that are under protection. A temporary checkpoint of the file system occurs each time a file is open for writing. Later, when the process writes to the file (2), the Analysis Policy examines the system call parameters and decides if the write is suspect or benign. If the write is suspicious, the *temporary checkpoint* converts to a *permanent snapshot*. If an attacker destroys

[5]Our prototype and associated third-party software protect Windows 7, but we do not anticipate significant obstacles to using DecMS-VMI with more recent versions of Windows. Conceptually, it should also port to Linux, Mac OS, and other systems.

the files, a system administrator can use the snapshot of the file system to recover the files. Temporary checkpoints are later consumed by a garbage collector to free up storage space if they are no longer needed, according to the Preservation Policy.

6.2.1 Observation Policy and Service

Method

To intercept I/O system calls and inspect their parameters, we used the Drakvuf Dynamic Malware Analysis System [69]. Drakvuf runs inside the VMM and inserts breakpoints in key memory locations inside the VM's memory to transfer control to the VMM. When an application within the VM issues a system call, control is transferred to Drakvuf. The design of Drakvuf is stealthy, such that it is difficult to detect the presence of Drakvuf within the monitored environment. We implement DecMS-VMI as a plugin for Drakvuf.

Policy

Our DecMS-VMI prototype intercepts two system calls on the guest VM: open file and write a file. To open a file for writing, applications within the guest VM must issue an open system call and request access permission for a given file. The Observation Policy examines all open system calls and determines if the file is open for writing by examining the call parameters. If the system call requests a write permission, the Observation Policy determines if the file should be protected based on a blacklist or whitelist. Algorithm 1 contains pseudocode summarizing the Observation Policy decision flow for open file system calls 6.2.1.

If the file is on the blacklist, we take a snapshot of the file system because the file is considered critical to system stability. Whitelisted files are considered unimportant and do not require preservation. If the file is on neither list, DecMS-VMI takes

Algorithm 1 Open File VMI Pseudocode

```
if File is opened for writing then
  if File is in Blacklist then
    Create a Snapshot (Permanent)
    Done
  else if File is in Whitelist then
    Done
  else
    Create Checkpoint (Temporary)
  end if
end if
```

a temporary checkpoint of the file system, and subsequent write system calls are analyzed, according to the Analysis Policy, to determine if the write is suspect.

Implementation

Several files may always need protection from destruction, regardless of the write pattern. Data objects that are critical for system stability, such as the Master Boot Record (MBR), are present in the blacklist. If a file on a blacklist is opened for writing, the Preservation Policy is automatically triggered, which handles the preservation of the file that is open for writing. The blacklist for our evaluation consists of references any devices in the Win32 Device Namespaces and the NT Namespaces [110].

Ignoring certain data objects, defined on the whitelist, provides an increase in performance by avoiding analysis of files that are not important. Data objects on the whitelist, such as caches and temporary files, are ignored. For our experimental evaluation, the whitelist includes all files that exist outside a specified "evaluation" directory. We do this for two reasons. First, it allows us to have better control over our experimental analysis because the system may contain file caches that are periodically written to, potentially interfering with our results. Secondly, in practice, the system files are much easier to recover from if destroyed. Users are more concerned with personal files, typically stored in their home directory.

DecMS-VMI examines the following Windows system calls according to the Observation Policy:

`NtOpenFile` is the Windows system call used to open existing files, while `NTCreatFile` can both open existing files and create new ones. The prototype Observation Policy identifies that the file exists and opened with the write permission by examining the `DesiredAccess` and `CreateDisposition` call parameters.

The `NtWriteFile` Windows system call writes data to an open file [121]. The `FileHandle, Buffer, ByteOffset`, and `Length` parameters are given to the Anal-

ysis Policy, if the file is not in the blacklist or whitelist, to determine if write is suspected of being destructive.

The NtSetInformationFile modify file metadata [122]. DecMS-VMI uses the `FileInformationClass, FileInformation,` and `FileHandle` parameters. When a FILE_BASIC_INFORMATION is indicated by `FileInformationClass`, a timestamp change is being made. For each timestamp type, a new time is specified in `FileInformation`. The Observation Policy in our evaluation examines all timestamp modifications for malicious timestamp changes. The Analysis Policy determines if one of the timestamp arguments as given to one of these system calls will result in a significant timestamp shift from the current time. We assume that the timestamp on the VM and the VMM are synchronized, and system time changes within the VM are tracked. The Preservation Policy logs all incidents within the VMM.

6.2.2 Analysis Policy and Service

The interposed information is analyzed to identify destructive actions. The Analysis Policy describes the set of algorithms to determine if a write call is suspect. The Analysis Policy details feature extraction that is then used to decide if a write is suspect. Feature extraction should be quick to minimize analysis latency.

Policy

Our experimental policy examines the first write to the beginning of a file. We later demonstrate in our evaluation that our conservative sampling provides high accuracy and acceptable performance. The analysis extracts the following features.

File Signatures We use the file signature corpus found in [123]. If the system call overwrites the file signature with data that does not match the file extension, we flag the write as suspicious.

Data Destruction Patterns Several secure delete algorithms, as shown in Table 5.2, overwrite files with repetitive patterns. To identify common secure delete

patterns, the Analysis Policy examines if the entire write buffer consists of repetitive bytes, such as 0xFF or 0x00. If the first 4 KiB of the write buffer contains a fixed pattern, we flag the write a suspicious.

Randomness Testing Several data destruction tools use random binary data to overwrite files. Prior work in [23,24,53] use entropy as a metric to identify encryption in ransomware. Instead, we use a classification tree with features used in evaluating cryptographic pseudorandom number generators [67]. The experimental evaluation demonstrates effectiveness for detecting data destruction attacks. The benefit of using a classification tree is the ease of implementation in C to provide a minimum impact on latency. A detailed description of the randomness test training and validation is given in Section 5.3. If the first 4 KiB of the write buffer appears to contain random binary data, we flag the write as suspicious.

Algorithm 2 shows the flow of our prototype implementation handling system write calls. The ordering of the detection methods is from most efficient to least efficient to compute. First, we check to see if the file extension in the `FileHandle` does not match the file signature found in the write buffer. Next, we check if the buffer contains known destructive patterns. Finally, a randomness test examines the write buffer. If any of the tests return true, then the checkpoint, created when the file was open, converts to a persistent snapshot.

Algorithm 2 Write File VMI Pseudocode

```
if First write and beginning of file then
  if File signature does not match file extension then
    Change checkpoint to snapshot
  else if Sampled buffer is a common pattern then
    Change checkpoint to snapshot
  else if Sampled buffer is random then
    Change checkpoint to snapshot
  end if
else if MAC DTS Update then
  if DTS does not match the current time ± δ then
    Log incident, including the original and attempted change MAC DTS.
  end if
end if
```

Implementation

Drakvuf interposes system calls within the guest VM transfers control between the guest VM and the VMM. When the guest VM issues an open or write system call, control transfers to the VMM. The previously mentioned features are extracted then pass through a decision process, shown in Algorithm 2. After the analysis completes, control transfers back to the guest VM, and the operating system continues.

A tree classifier identifies if a buffer contains random data. In Section 5.3, the details regarding parameter constraints and features to produce our classification tree.

6.2.3 Preservation Policy and Service

If a protected file is undergoing data destruction, the Preservation Policy defines the actions necessary to preserve the data, the information to log regarding the state of the VM, and the policies relating to the retention of temporary checkpoints when a file is open for writing.

Method

Several design choices are possible regarding the preservation of data under destruction. Upon detecting a destructive write and before the write is committed to storage, the files under destruction can move into a container that is isolated from the attacker. We define the above to be a *reactive strategy*. While the design is simple, the latency may be high if the file size is large. The reactive strategy only creates back-ups of files as destruction occurs, thus only occupying space when necessary.

In contrast to the reactive strategy, the *proactive strategy* protects data in anticipation of a destructive action. The advantage is that there are several existing schemes to quickly create a checkpoint of the storage medium or file to reduce latency relative to the reactive strategy. For example, a copy-on-write (CoW) scheme may outperform a reactive strategy. A checkpoint under a CoW scheme preserves the stor-

age state, and any changes to the storage are tracked. Analysis of destructive actions can occur after the data is written, reducing the latency compared to the reactive strategy. If destruction is determined, the system can rollback to the checkpoint. One disadvantage is that a checkpoint must be taken in a consistent state and be updated periodically to save disk space.

A version control or log-structured file system provides advantages over a CoW scheme. All writes are appended to the disk with a checkpoint number. Checkpoints are taken continuously, allowing for a system administrator to roll back to a consistent state if data destruction occurs. A disadvantage is a need for garbage collection. As writes occur on the disk, checkpoints are created and consume disk space. A garbage collector can run periodically, have a minimum time to retain checkpoints, and run when the disk is nearly full — all based on policy.

We find that for data preservation, a log-structured file system works well because writes are in append mode only, writes are efficient, and automatic garbage collection is possible (when checkpoints are no longer needed). We choose to use NilFS [124] for this purpose because of its superior performance for file writes. NilFS uses checkpoints, which can be garbage collected, and snapshots, which are permanent. A checkpoint converts to a snapshot upon command. We use the snapshot feature to make a permanent copy of the file system when we suspect a file is the target of data destruction. From the perspective of the Windows VM, the user is writing files to an NTFS disk while the hypervisor and Host OS translate the writes to NilFS. The creation of NilFS snapshots is done outside the Windows VM thus protecting it from a compromised guest OS.

Implementation

For our experimental evaluation, we take checkpoints synchronously when a file is opened. When a checkpoint is required, the task is given to a thread pool. Although NiLFS takes checkpoints quickly, we do not want the VM to hang during the

checkpoint creation. Subsequent writes block until the checkpoint completes. From our experimental evaluation, we found that this strategy does not impact the write performance significantly while providing certainty that the checkpoint matches the storage state.

6.2.4 Experimental Results

Our experimental evaluation is conducted on a machine running Ubuntu 16.04.2 LTS (GNU/Linux 4.4.0-75-generic x86_64) configured with Xen 4.8.0 with 4,096 MiB Domain0 memory and four dedicated vCPUs. The host machine is a Dell PowerEdge R410 with two Intel Xeon X5570s clocked at 2.93 GHz with 16 GB DDR3 Synchronized 1333Mhz RAM and a Samsung 850 250 GB SSD. We use the Drakvuf v0.4-7a79990 VMI tool to intercept Windows system calls associated with file I/O. The Windows Guest VM uses 2 GB of RAM, two vCPUs, and two virtual disks. The first disk contains the operating system and uses a Logical Volume Manager (LVM) partition on the host machine. The second virtual machine disk exists within a NilFS partition. All the experimental evaluation is conducted on the second disk, as it is easier to control the read/write accesses on a disk without interference from the OS related temporary files.

We compare the experimental results to our baseline system, which is a Windows VM that does not have DecMS-VMI enabled but uses NILFS to create a periodic checkpoint. The baseline system is a fair comparison to DecMS-VMI as it has the benefits of using NILFS to periodically take checkpoints but without the benefit of monitoring for potential data destruction. The garbage collector and continuous checkpointing are disabled to provide optimal disk performance for the baseline. The hardware and VM configuration for the baseline and DecMS-VMI are identical.

We evaluate the three requirements identified in Section 5.6.2: accuracy, preservation, and performance. For data preservation, we check the difference between the files under protection before and after running wiper malware samples, listed in Ta-

ble 5.1, and the secure delete tools, listed in Table 5.2. Some of the secure delete tools use deterministic patterns for every secure delete, such as overwriting with all zeros (e.g., British HMG IS5). Other methods generate random data at runtime (e.g., AFSSI-5020), so the data used to overwrite the file is different each run of the algorithm. While the sample size of wiper malware seems small, consider that modern wiper malware is not designed to be as widespread as other malware such as botnets or ransomware. The samples in our evaluation represent a sample of all the latest wiper malware found in the wild.

We refer to ordinary files that should not trigger the system as benign. The benign files used in our evaluation are from GovDocs [116].

For accuracy, we measure the recall and precision rates of our Analysis Policy to identify benign writes and destructive writes. Finally, to measure the performance impact of our prototype implementation, we measure and compare the latency and throughput on the same hardware with and without DecMS-VMI enabled. We also run PCMark 8 [125] to measure the performance impact of completing office tasks.

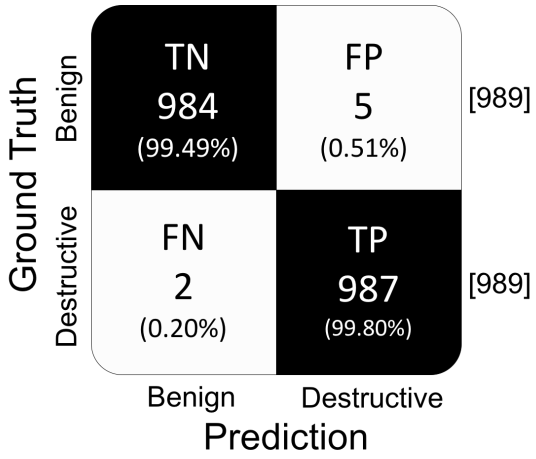
Metrics of Interest

Just as in the DecMS-Kernel prototype evaluation, the metrics of interest for DecMS-VMI are: the accuracy of benign or destructive classes, the impact on the latency associated with conducting such an analysis on a live system, and the ability to preserve the files under destruction.

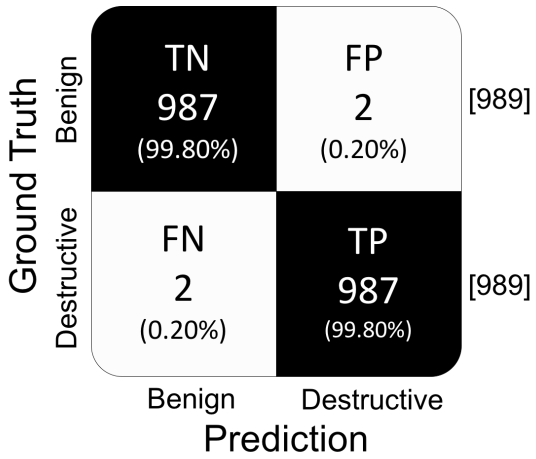
While both metrics are of importance, the cost of false positive classification is producing a NILFS snapshot on benign writes, which produces write latency for valid users. Therefore, we favor a higher recall value than precision when selecting parameters for our classifier. A high recall implies that there is are few false negative classifications. If speed is a priority, then a high precision rate may be of interest, implying that the classifier has few false positive classifications.

The components that contribute to latency include the latency to intercept system calls via VMI, the time needed to calculate features, the latency to run the Analysis Policy (e.g., classification time), and the latency to trigger and create a checkpoint or snapshot.

Accuracy Analysis



(a) Initial Results



(b) Corrected Results

Figure 6.7.: Confusion matrix for PRNG data destruction, the worst performing test.

In this section, we evaluate the accuracy of our Analysis Policy to distinguish between benign writes and destructive writes. The test set consists of two classes of files: Benign and Destructive. The Benign Class consists of Govdocs1 files from Thread2, which represents common files real users typically encounter and DecMS-VMI should not mistake writing these files as suspicious. The Benign files consist mostly of multimedia files such as PDF, HTML, and JPEG. Figure 5.1 lists the file-type distribution for testing accuracy.

The evaluation consists of overwriting files and observing the outcome of the Analysis Policy. For the Benign class, we overwrite the files' contents, without changing the file type, as would be the common case for benign use. For the Destructive class, we overwrite the files with all of the secure delete tools found in Table 5.2.

DecMS-VMI with our experimental Analysis Policy can correctly preserve files for *all* data destruction tools at 100% True Positive Rate, *except for the Pseudorandom Data destruction method.* Figure 6.7a shows the confusion matrix for DecMS-VMI under the Pseudorandom Data destruction method, which consists of overwriting a file with a single pass of pseudorandom bits. The false negative rate is 0.2%, in other words, out of 989 files, DecMS-VMI falsely identified two destructive overwrites as benign. A CSV file and an XML file are not triggered by file signature/extension mismatch because of the lack of fixed file signature for flat text files. In practice, false negative files are not necessarily lost. The checkpoints are available until the NILFS garbage collector runs, which can be set to only remove checkpoints if older than some date/time or if the filesystem is nearly full.

The false positive rate is 0.51%: five files out of 989 benign writes were incorrectly classified as destructive. Four of the five files triggered a snapshot because of a mismatched file signature. Upon further inspection, we verified that the four files (three Excel spreadsheets, and one PDF) in our testing set simply have mismatched file extensions and signatures. There are several reasons why the mismatch can happen in practice. As mentioned in [23], file signatures may change between different software versions. The three Excel Spreadsheets use a file signature for which was not in our

file signature corpus. It appears that these three files use an older file signature that we did not account. After adjusting the file signature corpus, we have a false positive rate of 0.2%, as shown in Figure 6.7b. The PDF file (206709.pdf in the GovDocs dataset) has 112 bytes of data before the correct file signature, which may indicate file corruption. The other false positive file, 186957.pdf in GovDocs, was incorrectly classified as destructive by our randomness classification tree.

DecMS-VMI also successfully detects all of the Wiper Malware in Table 5.1 *upon the first suspicious write*. Not only did we detect the first suspicious write for all of the samples, but we show in the following section that all of the files were successfully preserved without any unauthorized modification.

Preservation Under Wipers

To test if DecMS-VMI can accurately preserve data under destruction, we test against four wiper malware samples that have caused substantial damage to real systems. Table 5.1 list of all the malware we evaluate.

Our test environment consists of files found in GovDocs Thread9. We place all of the files under a protected directory in a separate disk image in our Windows VM. We also populate the VM with synthetic data to make the VM appear to be a real system rather than a malware analysis system. Internet connection is disabled in our test environment, as none of the samples require a network connection to destroy files on the system, according to the tech reports for each malware sample [4,107,108,126].

The malware samples use two approaches to destroy files on the system: overwriting each user file or by-passing the file system and overwriting the raw disk. The Shamoon, Shamoon2, and Stonedrill samples overwrite data using raw disk access, which triggered the Blacklist in our Analysis Policy. The Destover sample overwrites the protected files using a JPEG image fragment. The Analysis Policy triggers a snapshot, because of mismatched file extension and file signature, to preserve the files

under destruction. Therefore, the detection happens either before or upon the first destructive write to a protected file.

We ran each sample under administrative privileges with DecMS-VMI enabled and verified that the malware samples destroy data. After the destruction completes, we mount the disk image of the first snapshot taken because of the malware's data destruction actions. We then compare all of the files from the snapshot to the original files using the Unix `diff` tool. *All files were successfully preserved against all of the Wiper malware samples considered.*

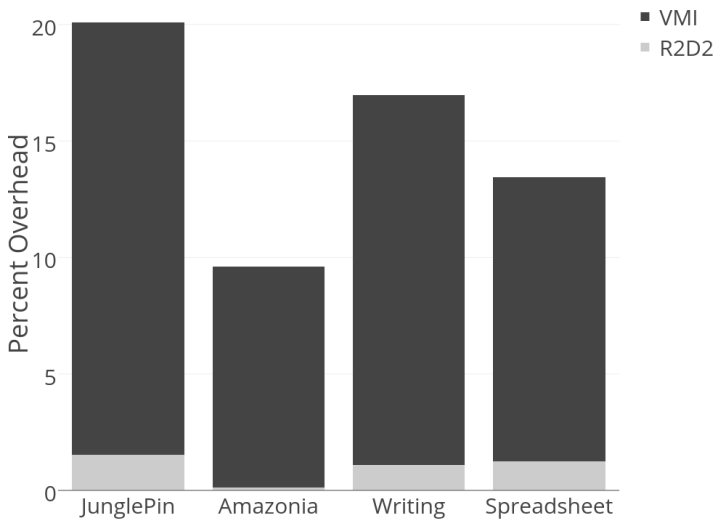
Performance Analysis

The performance evaluation consists of two categories: the latency for benign user tasks and suspect file writes.

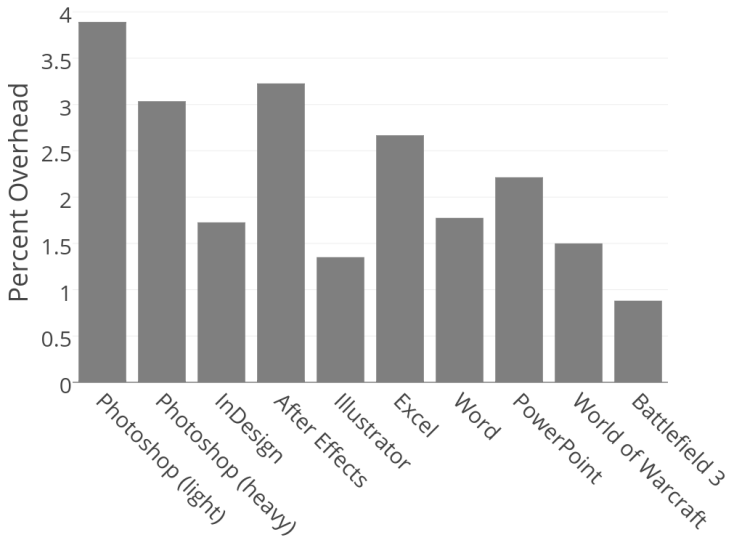
Benign Activity For benign user activity, we evaluate the performance by using PCMark 8 [125]. PCMark 8 is a benchmark suite to measure system performance under common user tasks. We evaluate DecMS-VMI using the "Work Benchmark" test suite [125] which consists of web browsing, document processing, and spreadsheet editing[6]. These tests mimic how real users interact with the software and measures user interface and runtime latencies. Below, a summary of each of the PCMark 8 Work Benchmark test for the evaluation follows. For details, refer to the PCMark 8 Technical Guide [125].

The Work Benchmark contains two separate web browser performance tests. The *JunglePin Test* simulates a user browsing on a social networking website. The test measures the latency to render the page and the rendering speed for several animations. The other web browser test, *Amazonia Test*, is an online commerce website, which consists of the latency to update a shopping cart and several animations. Both tests use Internet Explorer 9.

[6]We do not consider the video chat test because our experimental machine does not properly pass through GPU requests from the VM to the host machine.



(a) Work Tests



(b) Storage Tests

Figure 6.8.: PCMark 8 office benchmark results showing the overhead incurred by DecMS-VMI for a variety of common office tasks.

The Work Benchmark also includes a word processing test called the *Writing Test*. The test measures the time to load/save a document, resize the window, copy/paste text, and add a picture to a document. The Writing test uses a document editor develop by PCMark.

The *Spreadsheet Test* measures the time to open and close a spreadsheet, copy and paste data between spreadsheets, process data, and edit cells. The Spreadsheet Test uses LibreOffice Calc, an open source office document editor, for data processing.

Figure 6.8a illustrates the overhead of DecMS-VMI under several benign user tasks, from the Work Benchmark, which is typical in a work environment. The ratio shown in the bar chart is the Median overhead introduced by VMI and the DecMS-VMI Analysis Policy. The plot shows that the majority of the overhead is from VMI for all tests, with DecMS-VMI analysis contributing to less than 1.6% overhead, compared to the baseline, for all cases. The JunglePin Test performs the worst, with 20.09% total overhead, with 1.54% overhead caused by DecMS-VMI analysis. It appears that the overhead is caused by the rich image content of the JunglePin website. The majority of the overhead is from intercepting the file creation and write system calls for each image on the JunglePin web page. Additional overhead may also be caused by a system-wide impact associated with the use of VMI

The Work Benchmark measurements include the latency that effects user interface elements, such as the scroll speed for both JunglePin and Amazonia, resizing the window when writing a document, and the latency to edit a cell in a spreadsheet citepcmark. It appears that the UI elements are affected by using VMI (discussed with the throughput analysis), even without running any analysis or preservation. Since DecMS-VMI interposes storage I/O, we want to examine the impact of I/O performance degradation without considering the degraded UI performance. We run PCMark 8's "Storage Benchmark," [125] which measures storage latency performance, to compare the latency of the baseline and DecMS-VMI. The Storage Benchmark simulates disk usage and does not simulate the UI, which is the main distinction compared to the Work Benchmark.

The storage performance consists of I/O traces of several popular applications: four from Adobe Systems, three from Microsoft, and two computer games. The Adobe Systems applications include Photoshop, a photo editing application; InDesign, a web-publishing application; AfterEffects, a video editing and effects application; and

Illustrator, a vector image editing application. The Microsoft tests include Microsoft Word, Excel, and Powerpoint. The two computer games are Battlefield 3, a first-person shooter, and World of Warcraft, an online role-playing game. All tests have a single configuration except Photoshop, which runs under a "light" or "heavy" usage.

Figure 6.8b shows the median overhead for storage I/O for DecMS-VMI. The Adobe Photoshop and AfterEffects Test performed the worst with a median overhead of 3-4%. Based on the description of the tests [125], it appears that the Photoshop and AfterEffects Tests are heavy in random access I/O and writing. The best performing tests are World of Warcraft and Battlefield 3 with a median overhead of less than 1.5%. Both games are heavy in reading content on the disk and DecMS-VMI does not interpose file read system calls. Based on the results, it appears that DecMS-VMI introduces a latency increase of between 1% to 4% on disk storage.

To confirm that VMI interposition adds overhead to the entire system (including the UI), we ran DecMS-VMI using the CrystalDiskMark 5.2.1 [127] throughput, a Windows disk benchmarking application, to measure reading performance under DecMS-VMI. Recall that our experimental policies do not interpose read system calls. VMI reduces throughput for sequential reads by an average of 5.6% and 27% for random reads, compared to our baseline. The above provides evidence that the VMI methods we rely on account for overhead outside of system calls we interpose for DecMS-VMI.

We also examined the write throughput under DecMS-VMI, which exposes a limitation to using VMI in this implementation. Under the CrystalDiskMark random write test, we observed a decrease in performance of 57.5% in our prototype implementation of DecMS-VMI. However, 95.4% of the performance reduction was caused by VMI. Nonetheless, these results show that DecMS-VMI may not be suitable for applications that demand high throughput of random writes. Sequential writes, however, perform relatively well with 4.12% decrease in throughput, with VMI accounting for about 72.1%.

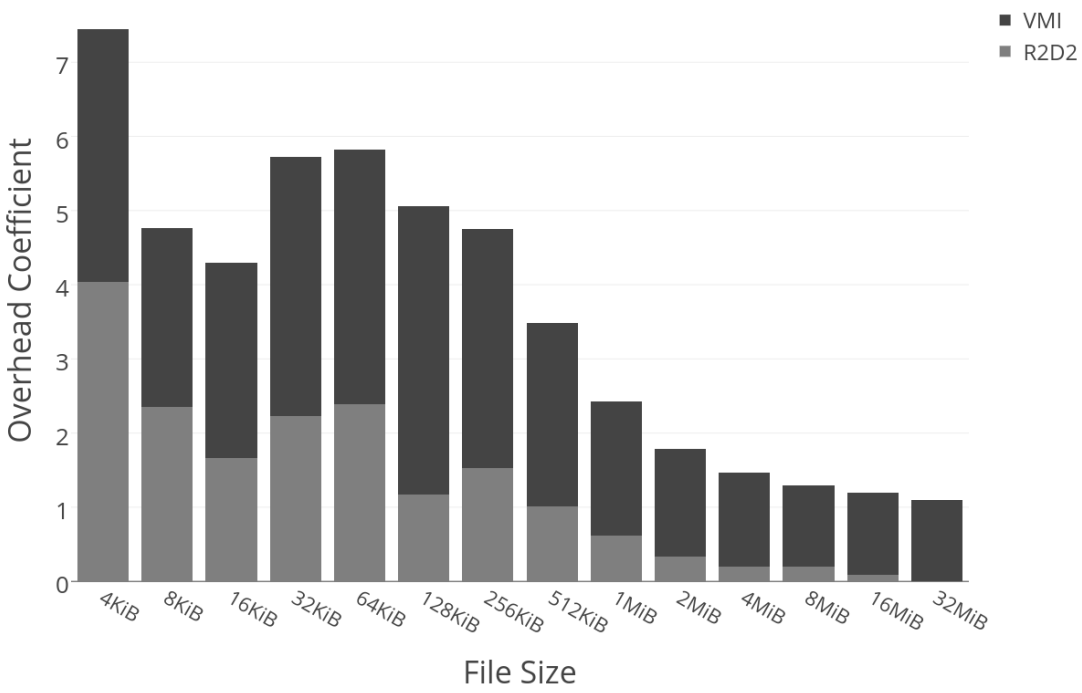


Figure 6.9.: Median Latency introduced by DecMS-VMI and VMI when data destruction is suspect.

Suspicious Activity To measure performance when a write is suspect, we measure the latency for destructive actions under various file sizes. Our test consists of overwriting different files of different sizes, from 4 KiB to 32 MiB. We measure the median latency ($n = 100$) to destroy an individual file of a specific size by overwriting the file with pseudorandom data. As defined by our Analysis Policy, the randomness test executes last, so all features of the analysis and classification execute during the test. Thus, the test represents the longest execution path for DecMS-VMI analysis.

Figure 6.9 contains the latency results of writing a fixed file size that triggers DecMS-VMI to convert a checkpoint into a snapshot. Recall that DecMS-VMI only examines the first 4 KiB to identify if a write is suspected of data destruction. As the size of the file increases, as shown on the X-axis, the analysis time becomes a smaller ratio to the time it takes to write the file. Thus, for small destructive writes, we see a large multiplicative factor in overhead. For 4 KiB files, the overhead is increased by

a factor 7.44X compared to the baseline. While the overhead seems to be quite large, this may not be relevant to a legitimate user. Foremost, if the latency is increased for a malicious action (data destruction), then that is a *desirable* outcome. So, we only worry about this increased latency for a false positive, i.e., a legitimate write by a legitimate user is mistaken to be a destructive write. We have seen from our accuracy analysis (Figure 6.7b) that this happens very rarely (0.2% of the cases). Finally, the worst case increase in latency is for the smallest file size of 4 KiB, and that increases latency from 0.16 ms to 1.20 ms, which is not perceptible for a user-interactive workload. Figures A.1, A.2, and A.3 in the appendix contains the write latencies for various file sizes, measured in milliseconds, for DecMS-VMI.

Summary of Results

Our results show that DecMS-VMI protects against data destruction attacks at the cost of tolerable overhead under several common workloads. While the overhead is non-negligible, our results indicate that there are several use cases where the overhead may be acceptable. The majority of the overhead is caused by VMI and not the DecMS-VMI analysis. For benign user tasks, the DecMS-VMI analysis only accounts for 1.4% to 9.29% (Figure 6.8a) of the total latency introduced, which is between 0.13% to 1.54% additional overhead compared to the baseline system. There has already been more than a 5x reduction in VMI overhead from 2004 to 2011 [62] [128] and as VMI mechanisms continue to improve in performance, DecMS-VMI will also inherit the benefits.

Latency analysis: Timestamp modification [7]

Figure 6.10 illustrates the latency of determining if a timestamp is fraudulent. The latency is measured as the cumulative latency of changing 100 timestamps and averaged. Note that in our parlance, one timestamp consists of four different 64-bit

[7] Assistance from Thomas Yurek for prototype implementation and evaluation.

values [129]: creation, last access, last written, last update, all of which are updated together. The baseline median latency for changing a timestamp is 7.46 ms, while the latency associated with VMI is 29.30 ms, a 3.83X increase. We suspect that the large factor increase in latency is because writing timestamp metadata to a file is quick and only requires writing 256-bits worth of data. The median latency associated with comparing the timestamp passed in as a parameter and the true system time is 36 ms, an increase of 22.9% over simple VMI. The increase in latency for timestamp update still keeps the total time to less than 30 ms and so should be tolerable for most user interactive scenarios. However, an informed attacker could examine the timestamp modification latency within the guest VM to determine if DecMS-VMI is running.

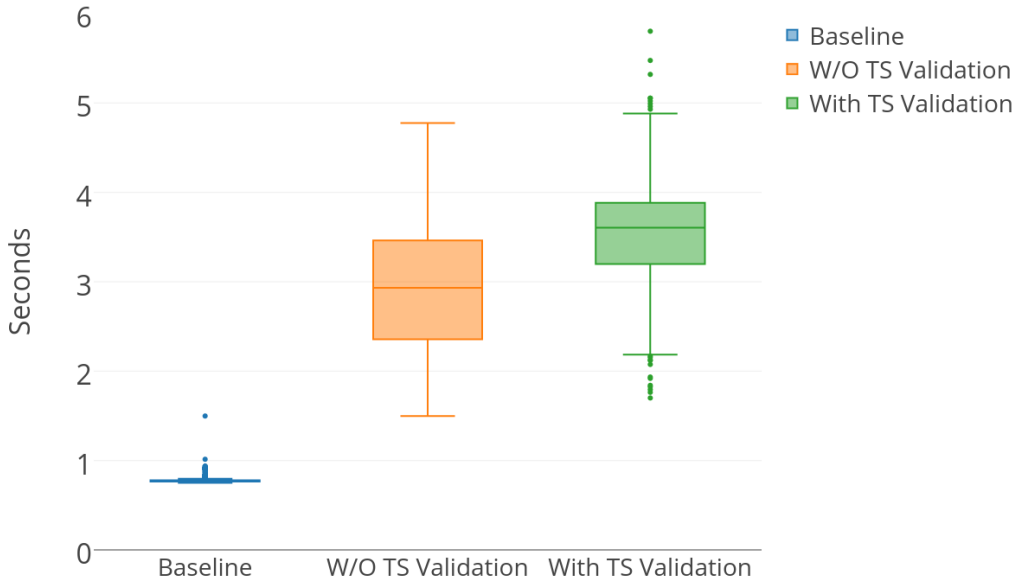


Figure 6.10.: Cumulative latency (100 samples) to check for fraudulent timestamps.

Protecting Against Storage Saturation

One potential issue in the DecMS-VMI configuration is protecting against a threat that saturates the system with multiple overwrites to the same file. As the DecMS-

VMI system uses NilFS, overwriting the same storage location multiple times with destructive writes causes the storage medium to fill at a linear rate. That is, for each destructive write of size n at the same location, the NilFS system must consume $O(kn)$, where k is the number of times a file is overwritten.

If the NilFS storage medium is full, a snapshot or checkpoint cannot be taken. The Preservation Policy should indicate what should happen. One strategy is to ignore checkpoint/snapshot creation each time such a request is observed from the VMM. Another policy is to impede the request and free old checkpoints. However, if the attacker is aware of policies of DecMS, she may attempt to remove old checkpoints that contain valuable information. Alternatively, the Deception Policy and service can inject a deceptive response.

Rather than denying the write request, an inconsistency response is given. The DecMS-VMI policy reports to the guest OS that the overwrite actions are successful, but the files still exist within the protected VM. If the adversary observes the failure, the attacker may attempt an alternative strategy to destroy the data, allowing for additional information to be gathered regarding attacker strategy. Further, the NilFS does not increase its space usage when the deception is injected into the system.

The silent error approach is successful for all of the secure delete methods in Figure 5.2, except Gutmann's 35-pass method. On Gutmann's 35-pass method, the Analysis Policy failed to detect data destruction for nine of the 35-passes. Nonetheless, the other 26 passes did not occupy space on the NilFS storage.

Note that some of the specifications for the secure delete algorithms require a verification between each write cycle. The experimental evaluation shows that the write action attempts three times and continues regardless of the fact that the data is not observable from the guest VM.

The results from the silent error experiment show that the data destruction tools do not follow the usual programming standard. One issue is that the secure delete specifications do not detail what to do if overwriting and verification failed. Another insight from this experiment is the fact that an inconsistent view of deception can be

effective if the threat does not validate its actions properly. Finally, the silent error injection is effective for several secure delete algorithms; thus DecMS-VMI is capable of adjusting strategies based on the state of the system.

7 CONCLUSION

7.1 Summary

This dissertation explores the use of deception to defend digital assets from remote threats that destroy information by overwriting data. Rather than halting a detected data destruction, the deception is designed to gather information about the adversary while preserving the data under destruction for later recovery.

Several design choices to achieve the above goals are presented in Chapter 4. A summary of the contributions include details in planning the deception for a DecMS system. The deceptive goals are defined, and several viable deceptive strategies were given. The deceptive planning also describes several services to facilitate the deception and preserve data from destruction. The design decisions for DecMS are driven by the goals and risks of using deception. Several computing systems are evaluated based on the cost metrics. DecMS could viably protect batch and interactive systems because of the flexibility in some cost factors. Further, the hardware and software design space are explored, and several locations are identified to monitor and defend against destructive threats.

Two software solutions, Kernel-based and VMI-based DecMS, are selected based on their ease of deployment while providing isolation techniques to mask the presence of the deception. In Chapter 6 the Kernel-based (DecMS-Kernel) and VMI-based (DecMS-VMI) solutions are evaluated. Both solutions attempt to mask the presence of DecMS and mimic the baseline system.

The evaluation of DecMS-Kernel demonstrates the ability to protect against secure delete and ransomware-like threats by preserving the data under destruction. One potential issue of DecMS-Kernel is the relatively high latency for large files; large files migrate to a log file, and the latency may be noticeable for an active adversary. The

impedance should be long enough to preserve the data under destruction without increasing suspicion. While an optimized solution is not a goal of the work presented here, improvements in performance were integrated into DecMS-VMI.

For DecMS-VMI, several improvements are incorporated based on the evaluation of DecMS-Kernel. First, a continuous logging file system preserves the files under destruction, significantly reducing the latency and throughput of the data destruction. Next, the control mechanisms for the preservation are placed outside of the system under protection, increasing the difficulty for an attacker to access DecMS from within the guest OS. In addition to the masking and mimicking strategy, a masking/inventing strategy is explored. The results indicate that the DecMS-VMI system is capable of defending against several wiper malware, secure delete, and timestamp fabrication techniques. The data is successfully preserved for nearly all threat instances and file preservation, except one single file where the classifier incorrectly identified a secure delete as a benign write.

By hiding the presence of data preservation and back-up system, it may be possible to use deception to protect a system against adversarial data destruction attacks. A careful understanding of threat bias and destruction methods is necessary to provide protection. The current proof of concept systems assumes that the threat will not attempt to overcome the defense mechanisms of DecMS. Future work should evaluate threats that attempt to overcome the deception and preservation methods found in DecMS. Further, the solutions here are not optimized for performance, so reducing the latency or increasing the throughput may be valuable. Improvements in speed is a desired for benign users on the system and may also increase the challenge for the adversary who conducts statistical analysis to uncover the presence of DecMS.

7.2 Shortcomings, Enhancements, and Future Work

The discussion below is focused on the DecMS-VMI proof of concept system as several of the shortcomings of DecMS-Kernel were addressed in DecMS-VMI.

7.2.1 Counterdeception

If the attacker becomes aware of DecMS-VMI, she may decide to go elsewhere and target a system without DecMS-VMI. However, if the attacker is persistent, she will attempt to circumvent DecMS-VMI. *Counterdeception* is an analysis of a denial and deception operation [130, Chapter 7]. A destructive adversary's counterdeception analysis may yield methods to circumvent the data preservation enhancements of DecMS. Three viable counterdeception strategies that an adversary may employ are (i) to remain hidden from DecMS monitoring from within the OS, (ii) run the data destruction outside of DecMS, or (iii) avoid the behaviors and heuristics that cause the snapshot to trigger.

Hide from Monitoring

Attackers have an array of techniques to hide from security monitoring tools and system administrators. As mentioned in Section 6.1.1, a variety of techniques exists, predominantly in use by rootkits, to hide from security monitoring (e.g., KOH or DKOM). If the attacker is successful in avoiding the VMM monitoring, then snapshot or checkpoint creation never triggers, which makes the recovery of the destroyed data difficult. Specifically, the adversary may install her own privileged I/O software to avoid the DecMS-VMI system call monitoring altogether. In addition to writing the I/O software, the attacker must circumvent software integrity and authenticity validation[1], and have the privilege to install custom drivers or to make changes to the kernel.

Another approach for the attacker is to use rootkit hiding techniques to avoid the VMM monitoring. In the current prototype configuration of DecMS-VMI, an attacker who uses a KOH should be able to circumvent the system call monitoring. However, a KOH detection technique already exists in Drakvuf, the VMI tool on which DecMS-VMI builds. The System Service Descriptor Table (SSDT) Monitoring plug-in allows

[1]Such as Windows 10 driver verification [131].

Drakvuf to detect if the system call table within Windows is modified. The SSDT Monitoring plugin can be extended to work with DecMS-VMI and trigger a snapshot before any modification to the SSDT is observed, in addition to other actions at the security administrator's discretion. The attacker must use a hiding technique that DecMS-VMI, or VM security monitoring software, fail to detect. While there are several methods, such as KOH or DKOM, the attacker must evaluate the stealthiness against VM security monitoring tools. Rather than avoiding *all* monitoring, the attacker may choose a strategy that requires less effort to avoid *some* aspects of DecMS-VMI. The attacker may alter her destructive behavior to avoid the snapshot trigger or destroy the data from a location that the current configuration of DecMS-VMI cannot observe.

Out-of-band Destruction

Another counterdeception method to circumvent DecMS is to modify the bootloader and destroy data from outside the operating system. An attacker who writes over the bootloader without detection can destroy the files or the entire file system without the risk of the VMM interposing OS system calls. In our evaluation, any changes to the NTFS data structures automatically trigger a snapshot of the file system. Our experimental evaluation does not include malware that attempts to replace the bootloader, but some samples did modify the partition table, which is detected and in principle should work the same if the malware overwrites the bootloader. However, a limitation of DecMS-VMI, assuming the KOH/DKOM detection methods fail, is if the attacker replaces the bootloader by circumventing the VMI interposition, then she may be able to destroy files without triggering checkpoints/snapshots. Under the experimental configuration of DecMS-VMI, an attacker may be able to replace the bootloader without detection if she can execute before DecMS-VMI is enabled. The experimental evaluation assumes that DecMS-VMI is enabled before any malware or data destruction tools execute. In practice, if a race condition exists, the attacker

is likely to use it for out-of-band destruction. An attacker may modify an existing wiper malware to execute before DecMS-VMI is enabled.

Bacs et al. in [68] successfully demonstrate the ability to modify changes to virtual disks at specified locations such as the bootloader. The work identifies specific regions of the virtual disk for any modification without relying on VMI. One solution is to combine the detection mechanism in [68] with the preservation methods in the work proposed here. Before modifying the bootloader, create a snapshot so that the filesystem is recoverable if the modification is malicious. The advantage is that monitoring for changes on specific regions of a virtual disk can be done without DecMS-VMI or the guest OS running, mitigating the race condition attack mentioned previously.

Another possibility is to escape the guest OS and compromise the host machine [132]. Once the attacker compromises the host machine, which is assumed to be trusted in this dissertation, the attacker may destroy the preserved files. In addition to common security practices to audit and verify the correctness of VM software, DecMS can be layered and protect the files stored on the host OS. Protecting the host machine from an attacker who escapes the VM is outside of the scope of DecMS.

Modify Behavior to Misclassify Destruction

An attacker may avoid the behavior and heuristic indicators that trigger a snapshot of the file system. To circumvent our experimental Analysis Policy, the attacker may destroy a file by overwriting it with data that is non-random, that does not follow common data destruction patterns, and avoids overwriting file signatures. However, note that our experimental policy creates a checkpoint each time a protected file is opened for writing. For an attacker to overcome the checkpoint and destroy a file permanently, she must circumvent the Analysis Policy and force the NILFS garbage collector to deallocate the relevant checkpoint. The effort to force the NILFS garbage collection to run is high (compared to the baseline system), requiring the attacker to

either fill the storage space and wait until the checkpoint is destroyed or wait until the minimum retention time for checkpoints is met. For both cases, the speed of the attacker is reduced compared to the state of practice.

Alternatively, an attacker may also avoid the Analysis Policy by overwriting small segments of the file. The experimental Analysis Policy only analyzes write buffers of at least 4,096 KiB and when writes occur at the beginning of a file, as a trade-off of robustness for speed. Currently, the experimental Analysis Policy is adjustable to sample all writes or to sample the write buffers randomly. Additional experimentation is necessary to determine the sampling rate to provide sufficient protection without compromising performance.

The attacker may move a file from a blacklisted directory into one that is whitelisted, such as a web-browser cache, and destroy the file there. The Observation Policy can monitor the NtSetInformationFile system call, which renames and moves files, to mitigate the attack. DecMS-VMI takes a snapshot if a file is moved from a protected directory to one with less strict monitoring rules. Similarly, NtDeleteFile should also be monitored to prevent an attacker from deleting protected files and then filling the space on the disk.

The weaknesses above are not exclusive to DecMS-VMI but all security monitoring tools. Attackers are persistent and eventually discover new methods to avoid monitoring. However, the design of DecMS-VMI increases the challenge of unauthorized data destruction by isolating the snapshot or checkpoint mechanism and preserving the data under destruction that is out of reach for the attacker. Some policy changes can help mitigate destructive attacks by periodically taking snapshots regardless of the changes observed, a standard configuration of NILFS. Further, DecMS-VMI can introduce inconsistencies to produce doubt for the attacker's data destruction methods. As mentioned in prior work related to VMI, the latency associated with VMI can be hidden from the attacker by adjusting for the timing delay within the guest VM [62], making the detection of DecMS-VMI from the attackers' perspective more difficult. Further, injecting deceptive faults into the VM to disrupt the data destruc-

tion may also slow down or disrupt the attacker. The uncertainty and confusion may cause the attacker to stall and waste time overcoming faults that may not exist, as shown by the work of Sun et al. [58].

For the prototype evaluation of DecMS-VMI, if the attacker is aware of all of the methods to trigger a snapshot, then, with minimal effort, the attacker may modify her destruction to avoid triggering a snapshot and force (or wait) for the garbage collector to remove temporary checkpoints. Even if the attacker does not trigger the garbage collector, it may be difficult for the defender to identify and recover from some destructive incidents. Specifically, attacks on integrity or authenticity may be troublesome if there is no clear indication of when a file suffered a loss of integrity or authenticity. The defender is forced to trace through the checkpoints and identify when the incident occurred. Slowing down the speed at which the defender can recover files may be sufficient to meet the goals and motivation of adversary.

Counter-Counterdeception

Identifying the strategies to circumvent DecMS can help produce features to identify destructive adversaries. Almeshekah and Spafford identify that a successful deception for defense should monitor the perception and actions of an adversary to adjust the defense accordingly [7]. Features such as (i) filling the disk with checkpoints/snapshots at a high rate (to force garbage collection) or (ii) out-of-band writing to the guest virtual disk are both viable counter-counterdeception features. Future work for DecMS should investigate the use of counter-counterdeception features to identify adversaries that attempt to circumvent the protection mechanisms of DecMS.

7.2.2 Alternative Analysis Policy

An efficient method to identify Crypto Ransomware is analyzing read/write patterns, as shown in [23, 24, 53]. Ransomware follows several predictable I/O patterns,

such as overwriting the file with the encrypted version or copying the encrypted file elsewhere and destroying the original file [24].

Read/Write patterns may apply to detecting Wiper Malware. However, it appears that Wipers overwrite files and do not necessarily read the file contents before destruction. Future work should investigate other I/O patterns that are write-centric. Other possible detection metrics include the process-centric approach in [53], which measures the rate at which a process writes to files.

File type funneling quantifies the number of file type read and written to a storage medium per process [23]. Other patterns, such as directory listing, file type coverage, and file renaming, are features for the classifier in [53]. File-centric features may not be applicable for Wiper Malware. Wipers may write directly to a storage medium, bypassing file system conventions [4,107,108]. All of the above metrics are file-centric and are not readily applicable to low-level disk writing. Applying the above metrics for low-level storage access requires the defender to bridge the semantic gap to a raw storage medium. Prior work [91] provides some solution to the raw storage medium semantic gap problem.

Some of the metrics listed above are not applicable to secure delete tools. For example, secure delete tools do not need to read a file before destroying it. Listing the files within a directory is not required for secure deletion. Some secure delete tools do not follow conventional patterns and thus make detection difficult without prior knowledge of how the tool works. For instance, the `sdelete` secure delete tool renames a file 26 times before destruction [113]. However, it appears that renaming files before destruction is uncommon.

Anomaly detection is another viable detection method, proposed in Garfinkel and Rosenblum's VMI work [62], that we do not consider in our experimental evaluation. The experimental evaluation for DecMS-VMI demonstrates that the performance impact is small for false positive detection. The disadvantage is the need to train for benign interactions of files.

A substantial difference between Wiper Malware and Ransomware is the stability of the compromised system. The goal of Ransomware is to collect an extortion payment. Ransomware must present the ransom note to the end user, which means there must be a system that is at least partly functional to provide such output to the user. Presumably, if the user pays the extortion, the files are unlocked.

The goal of some wiper malware is to make a system or data unavailable to users. The attacker is not required to present a message to the end user upon wiping a machine [2]. Several examples [4,108] restart the compromised machine, which displays a Master Boot Record error.

7.2.3 Other Limitations

A benign user may have a legitimate reason to destroy or encrypt data, and that should not trigger DecMS-VMI. The grounds to destroy or encrypt data may be completely valid for privacy reasons. Our current implementation does not support valid data destruction or encryption from within the VM. A simple solution to support valid data destruction or encryption is through a manual out-of-band mechanism whereby the user approaches the system administrator to disable DecMS-VMI for the specific user's VM temporarily. The simple solution, however, is a path for an attacker to subject the users and system administrators to social engineering whereby the protection is temporarily disabled for unauthorized data destruction. A better solution for benign data encryption and destruction is left for future work.

The extra storage requirement of DecMS-VMI is directly related to the frequency and size of edits made to protected files. As NilFS behaves similarly to a versioning file system, the physical size of a snapshot is only the size of changes made to files. With a false positive rate of roughly one in one thousand, an administrator can expect that a snapshot, containing all changes to all protected files since the last snapshot, will be generated for roughly every one thousand saved changes. In practice, the size of these

[2]There are exceptions [107] if the attacker wishes to let the defenders know of the destruction and convey a message.

snapshots should be much smaller than the total size of all protected directories, and so the extra storage needed to use DecMS-VMI is comparatively small, depending heavily on entropy of the writes.

Additional enhancements for storage space that is not explored in this dissertation is combing NILFS with a full backup, to optimize storage space for the guest VMs under protection. If storage space on the guest OS is full, then the storage is unavailable until the garbage collector can free storage space by purging old checkpoints. One possible solution is to periodically create a full backup of the entire NILFS storage from the host OS. All of the snapshots and checkpoints are then cleared, to free up space on NILFS. If at some point a destructive adversary is discovered, then the full backup can be referenced to recover files. Backing up and cleaning the NILFS snapshots frees up storage for the guest OS and is a trade-off of convenience. If the destruction occurred before the full backup but discovered after clearing the checkpoints/snapshots, then the system administrator must reference the full backup rather than rollback to an old checkpoint/snapshot. Future work should explore the benefits of using a full backup (and clearing snapshots/checkpoints) with DecMS configuration that uses a log-structured filesystem.

Our prototype implementation of DecMS-VMI only supports the snapshot or checkpointing at the file system level and does not allow for recovery of individual files without reverting the entire file system. While it is suitable for recovering from devastating wiper malware that attempts to bring a system and data offline quickly, the current solution does not directly provide methods to repair individual files. The system administrator would then need to manually identify what files should be saved or reverted before rolling back to an earlier snapshot. One solution is to replace the log-structured file system with one that tracks changes at the file level. The system administrator may then revert specific files to previous versions without rolling back the entire file system.

Another limitation of the current prototype of DecMS-VMI is if destruction actions are intermingled with benign writes to the storage medium. In the current

implementation of DecMS-VMI, the snapshots roll back to the state before data destruction takes place, undoing benign writes along with destructive writes. It may be difficult to revert the effects of destructive actions when benign applications are affected by the destruction. Fortunately, there are existing systems, such as the Taser Intrusion Recovery System [55] mentioned in Section 2.5.5. Taser resolves the issue through taint analysis. Taser requires snapshots that are already in place in DecMS-VMI. However, incorporating that into DecMS-VMI will require additional engineering effort; to work with a taint analysis system is left for future work.

While the dissertation does not focus on performance, there are several enhancements to explore in future work. Improving throughput should be a priority to help expand the applicability to other computer systems, such as batch and transaction systems. The DecMS-VMI prototype also effects the user interface, which may be an issue if used in practice. It appears that transferring control to the VMM whenever there is an open or write system call causes the UI to stall. Another future enhancement is to avoid checkpoint/snapshot for destructive writes to small files. It may be quicker to copy the file than to issue a checkpoint for the entire file system.

Another area of future work, which may improve performance and further isolate DecMS from potential adversaries, is to place DecMS within hardware components such as a hardware wrapper around storage devices or placing DecMS within the hardware controller itself. Future work should examine if it is feasible to improve performance so that an attacker cannot determine if DecMS is in place without insider knowledge or access to the physical system.

7.3 Conclusion

The evidence provided by the experimental evaluation confirms the thesis: It is feasible to use deception to enhance the preservation of digital assets against unauthorized data destruction.

LIST OF REFERENCES

LIST OF REFERENCES

[1] A. Solomon, "A Brief History of PC Viruses," *Computer Fraud & Security Bulletin*, vol. 1993, no. 12, pp. 9–19, 1993.

[2] D. B. Parker, "Toward a New Framework for Information Security?," in *Computer Security Handbook*, pp. 3.1–3.23, John Wiley & Sons, Inc., 2012.

[3] N. Perloroth, "In Cyberattack on Saudi Firm, U.S. Sees Iran Firing Back," *New York Times*, October 2012. `http://www.nytimes.com/2012/10/24/business/global/cyberattack-on-saudi-oil-firm-disquiets-us.html` News report. Accessed: 2017-06-02.

[4] C. Raiu, M. A. Hasbini, S. Belov, and S. Mineev, "From Shamoon to Stonedrill – Wipers Attacking Saudi Organizations and Beyond," *Kaspersky Lab*, March 2017. Version 1.05. Report.

[5] L. E. Panetta, "Remarks by Secretary Panetta on Cybersecurity to the Business Executives for National Security," October 2012. Transcript from `http://archive.defense.gov/transcripts/transcript.aspx?transcriptid=5136` Accessed: 2017-06-02.

[6] "Internet Security Tech Report," *Symantec*, vol. 22, April 2017. `https://www.symantec.com/content/dam/symantec/docs/reports/istr-22-2017-en.pdf` Report. Accessed: 2017-10-30.

[7] M. H. Almeshekah and E. H. Spafford, "Planning and Integrating Deception into Computer Security Defenses," in *Proceedings of the 2014 New Security Paradigms Workshop*, NSPW '14, (New York, NY, USA), pp. 127–138, ACM, 2014.

[8] M. Perklin, "Anti-Forensics and Anti-Anti-Forensics," *DEFCON 20*, July 2014. `https://www.defcon.org/images/defcon-20/dc-20-presentations/Perklin/DEFCON-20-Perklin-AntiForensics.pdf` Presentation. Accessed: 2017-11-17.

[9] "2015 Data Breach Investigations Report," *Verizon Wireless*, 2015. `http://www.verizonenterprise.com/resources/reports/rp_data-breach-investigation-report_2015_en_xg.pdf` Report. Accessed: 2017-11-17.

[10] R. C. Daley and P. G. Neumann, "A General-purpose File System for Secondary Storage," in *Proceedings of the November 30–December 1, 1965, Fall Joint Computer Conference, Part I*, AFIPS '65 (Fall, part I), (New York, NY, USA), pp. 213–229, ACM, 1965.

[11] R. Kissel, A. Regenscheid, M. Scholl, and K. Stine, "NIST Special Publication 800-88 Guidelines for Media Sanitization." National Institute of Standards and Technology, September 2006. Revision 1. `http://ws680.nist.gov/publication/get_pdf.cfm?pub_id=50819` Accessed: 2017-11-17.

[12] J. Reardon, *Secure Data Deletion*. Information Security and Cryptography, Springer International Publishing, 2016.

[13] B. D. Carrier, *A Hypothesis-based Approach to Digital Forensic Investigations*. PhD thesis, Purdue University, West Lafayette, IN, USA, 2006. AAI3232156.

[14] G. H. Kim and E. H. Spafford, "The Design and Implementation of Tripwire: A File System Integrity Checker," in *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, CCS '94, (New York, NY, USA), pp. 18–29, ACM, 1994.

[15] "Delete," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[16] D. J. Santry, M. J. Feeley, N. C. Hutchinson, and A. C. Veitch, "Elephant: The File System that Never Forgets," *Hot Topics in Operating Systems, 1999. Proceedings of the 7th Workshop on*, pp. 2–7, 1999.

[17] M. Weik, "Undelete," in *Computer Science and Communications Dictionary*, pp. 1857–1857, Boston, MA: Springer US, 2001.

[18] J. Reardon, D. Basin, and S. Capkun, "SoK: Secure Data Deletion," in *Proceedings of the 2013 IEEE Symposium on Security and Privacy*, SP '13, (Washington, DC, USA), pp. 301–315, IEEE Computer Society, 2013.

[19] "Office of the Designated Approving Authority (ODAA) Process Manual," *Defense Security Service, United States of America*, November 2013. Version 3.2.

[20] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. New York, NY, USA: John Wiley & Sons, Inc., 2nd ed., 1995.

[21] A. Savoldi, M. Piccinelli, and P. Gubian, "A Statistical Method for Detecting On-disk Wiped Areas," *Digital Investigation*, vol. 8, no. 3, pp. 194 – 214, 2012.

[22] G. C. Kessler, "Anti-Forensics and the Digital Investigator," in *Proceedings of the 5th Australian Digital Forensics Conference*, December 2007.

[23] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and Drop It): Stopping Ransomware Attacks on User Data," *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, vol. 00, pp. 303–312, 2016.

[24] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," in *25th USENIX Security Symposium (USENIX Security 16)*, (Austin, TX), pp. 757–772, USENIX Association, 2016.

[25] B. Blunden, *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Jones & Bartlett Learning, 2011.

[26] J. Williams and A. Torres, "ADD – Complicating Memory Forensics Through Memory Disarray," *ShmooCon*, January 2014.

[27] R. Harris, "Arriving at an Anti-forensics Consensus: Examining How to Define and Control the Anti-forensics Problem," *Digital Investigation*, vol. 3, pp. 44–49, Sept. 2006.

[28] G. Brose, "Access Control," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 2–7, Boston, MA: Springer US, 2011.

[29] A. Estes, "Biba Integrity Model," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 81–81, Boston, MA: Springer US, 2011.

[30] S. De Capitani di Vimercati and P. Samarati, "Clark and Wilson Model," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 208–209, Boston, MA: Springer US, 2011.

[31] M. Bishop, "Biba Integrity Model," in *Computer Security: Art and Science*, Addison-Wesley, 2003.

[32] K. Thompson, "Reflections on Trusting Trust," *Communications of the ACM*, vol. 27, pp. 761–763, Aug. 1984.

[33] S. Garfinkel, G. Spafford, and A. Schwartz, *Practical Unix & Internet Security, 3rd Edition*. O'Reilly Media, Inc., 2003.

[34] A. Chervenak, V. Vellanki, and Z. Kurmas, "Protecting File Systems: A Survey of Backup Techniques," in *Joint NASA and IEEE Mass Storage Conference*, 1998.

[35] H. Berghel, "Hiding Data, Forensics, and Anti-forensics," *Communications of the ACM*, vol. 50, pp. 15–20, April 2007.

[36] P. Gupta, H. Krishnan, C. P. Wright, M. Zubair, J. Dave, and E. Zadok, "Versatility and Unix Semantics in a Fan-Out Unification File System," tech. rep., Stony Brook University, 2004. FSL-04-01.

[37] R. Pike, D. Presotto, S. Dorward, B. Flandrena, K. Thompson, H. Trickey, and P. Winterbottom, "Plan 9 from Bell Labs," *Computing Systems*, vol. 8, no. 3, pp. 221–254, 1995.

[38] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai, "The Linux Implementation of a Log-structured File System," *SIGOPS Operating Systems Review*, vol. 40, pp. 102–107, July 2006.

[39] G. Palmer, "A Road Map for Digital Forensic Research," *Proceedings of the 2001 Digital Forensics Research Workshop (DFRWS 2004)*, pp. 1–42, 2001.

[40] M. Kuhn, "Data Remanence," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 306–306, Boston, MA: Springer US, 2011.

[41] B. Carrier, "NTFS Orphan Files," *The Sleuth Kit Informer*, September 2004. Issue 16. `https://www.sleuthkit.org/informer/sleuthkit-informer-16.html` Accessed: 2017-8-28.

[42] S. L. Garfinkel, "Carving Contiguous and Fragmented Files with Fast Object Validation," *Digital Investigation*, vol. 4, pp. 2–12, Sept. 2007.

[43] K. Bazzani, "Hybrid-device Storage Based on Environmental State," September 2014. US Patent 8,850,151.

[44] M. H. Ligh, A. Case, J. Levy, and A. Walters, *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. Wiley Publishing, 1st ed., 2014.

[45] F. Buchholz and E. Spafford, "On the Role of File System Metadata in Digital Forensics," *Digital Investigation*, vol. 1, pp. 298–309, December 2004.

[46] E. H. Spafford, "Some Challenges in Digital Forensics," Research Advances in Digital Forensics. Proceedings of the IFIP Conference on Distributed Computing Systems (ICDCS 2006), (Lisbon, Portugal), Springer, August 2006.

[47] L. Milkovic, "Defeating Windows Memory Forensics," *29th Chaos Communication Congress*, December 2012. Presentation.

[48] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen, "ReVirt: Enabling Intrusion Analysis Through Virtual-machine Logging and Replay," *SIGOPS Operating Systems Review*, vol. 36, pp. 211–224, Dec. 2002.

[49] B. Schneier and J. Kelsey, "Secure Audit Logs to Support Computer Forensics," in *ACM Transactions on Information and System Security*, vol. 2, (New York, NY, USA), pp. 159–176, ACM, May 1999.

[50] N. Idika and A. P. Mathur, "A Survey of Malware Detection Techniques," tech. rep., Purdue University, February 2007. #4328.

[51] P. Okane, S. Sezer, and K. McLaughlin, "Obfuscation: The Hidden Malware," *IEEE Security and Privacy*, vol. 9, no. 5, pp. 41–47, 2011.

[52] A. G. Pennington, J. L. Griffin, J. S. Bucy, J. D. Strunk, and G. R. Ganger, "Storage-Based Intrusion Detection," *ACM Transactions on Information and System Security*, vol. 13, pp. 30:1–30:27, Dec. 2010.

[53] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "ShieldFS: A Self-healing, Ransomware-aware Filesystem," in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ACSAC '16, (New York, NY, USA), pp. 336–347, ACM, 2016.

[54] J. D. Strunk, G. R. Goodson, M. L. Scheinholtz, C. A. N. Soules, and G. R. Ganger, "Self-securing Storage: Protecting Data in Compromised System," in *Proceedings of the 4th Conference on Symposium on Operating System Design & Implementation – Volume 4*, OSDI'00, (Berkeley, CA, USA), USENIX Association, 2000.

[55] A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara, "The Taser Intrusion Recovery System," in *Proceedings of the 20th ACM Symposium on Operating Systems Principles*, SOSP '05, (New York, NY, USA), pp. 163–176, ACM, 2005.

[56] M. H. Almeshekah and E. H. Spafford, "Cyber Security Deception," in *Cyber Deception* (S. Jajodia, V. S. Subrahmanian, V. Swarup, and C. Wang, eds.), pp. 25–52, Springer, 2016.

[57] V. Neagoe and M. Bishop, "Inconsistency in Deception for Defense," in *Proceedings of the 2006 Workshop on New Security Paradigms*, NSPW '06, (New York, NY, USA), pp. 31–38, ACM, 2007.

[58] R. Sun, D. E. Porter, D. Oliveira, and M. Bishop, "The Case for Less Predictable Operating System Behavior," in *15th Workshop on Hot Topics in Operating Systems (HotOS XV)*, (Kartause Ittingen, Switzerland), USENIX Association, 2015.

[59] J. Kong, *Designing BSD Rootkits*. San Francisco, CA, USA: No Starch Press, 2007.

[60] M. Coppola, "Suterusu Rootkit: Inline Kernel Function Hooking on x86 and ARM," *Michael Coppola's Blog*, June 2013. `https://poppopret.org/2013/01/07/suterusu-rootkit-inline-kernel-function-hooking-on-x86-and-arm/` Accessed: 2016-06-06.

[61] J. F. Levine, J. B. Grizzard, and H. L. Owen, "Detecting and Categorizing Kernel-level Rootkits to Aid Future Detection," *IEEE Security Privacy*, vol. 4, pp. 24–32, Jan 2006.

[62] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *In Proceedings of Network and Distributed Systems Security Symposium*, pp. 191–206, The Internet Society, 2003.

[63] X. Jiang, X. Wang, and D. Xu, "Stealthy Malware Detection and Monitoring Through VMM-based "Out-of-the-Box" Semantic View Reconstruction," *ACM Transactions on Information and System Security*, vol. 13, no. 2, pp. 1–28, 2010.

[64] M. S. Barik, G. Gupta, S. Sinha, A. Mishra, and C. Mazumdar, "An Efficient Technique for Enhancing Forensic Capabilities of Ext2 File System," *Digital Investigation*, vol. 4, pp. 55–61, 2007.

[65] B. A. Kuperman and E. Spafford, "Audlib: A Configurable, High-fidelity Application Audit Mechanism," *Software – Practice and Experience*, vol. 39, no. 7, pp. 701–736, 2009.

[66] K. Nance, B. Hay, and M. Bishop, "Investigating the Implications of Virtual Machine introspection for Digital Forensics," *International Conference on Availability, Reliability and Security, ARES 2009*, pp. 1024–1029, 2009.

[67] A. Rukhin, J. Soto, J. Nechvatal, S. Miles, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, and S. Vo, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications," *National Institute of Standards and Technology*, vol. 800, no. April, p. 131, 2010.

[68] A. Bacs, C. Giuffrida, B. Grill, and H. Bos, "Slick: An Intrusion Detection System for Virtualized Storage Devices," in *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, SAC '16, (New York, NY, USA), pp. 2033–2040, ACM, 2016.

[69] T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias, "Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014.

[70] "Use Bootrec.exe in the Windows RE to Troubleshoot Startup Issues," *Microsoft Corporation*, March 2017. `https://support.microsoft.com/en-us/help/927392/use-bootrec-exe-in-the-windows-re-to-troubleshoot-startup-issues` Technical support website. Accessed: 2017-10-26.

[71] C. Grenier, "Testdisk," *CGSecurity*, June 2016. `http://www.cgsecurity.org/` Open source software website. Accessed: 2017-10-26.

[72] "Shred – Linux Man Page," *Linux Documentation*, 2014. `http://linux.die.net/man/1/shred` Software documentation. Accessed: 2017-12-01.

[73] "Srm - Secure File Deletion for POSIX Systems," *SourceForge*, 2014. `http://srm.sourceforge.net/` Software website. Accessed: 2017-12-01.

[74] B. Jain, M. B. Baig, D. Zhang, D. E. Porter, and R. Sion, "SoK: Introspections on Trust and the Semantic Gap," in *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, SP '14, (Washington, DC, USA), pp. 605–620, IEEE Computer Society, 2014.

[75] T. Haq, "Meet GreenDispenser: A New Breed of ATM Malware," *Proofpoint*, September 2015. `https://www.proofpoint.com/us/threat-insight/post/Meet-GreenDispenser` Technical blog. Accessed: 2017-10-19.

[76] "Latency," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[77] "Throughput," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[78] "Measures of Variation," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[79] "Error Rate," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[80] "Fault-tolerant System," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[81] A. S. Tanenbaum and H. Bos, *Modern Operating Systems*. Upper Saddle River, NJ, USA: Prentice Hall Press, 4th ed., 2014.

[82] "Interactive," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[83] S. K. Card, G. G. Robertson, and J. D. Mackinlay, "The Information Visualizer, An Information Workspace," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, (New York, NY, USA), pp. 181–186, ACM, 1991.

[84] P. Bernstein and E. Newcomer, *Principles of Transaction Processing*. The Morgan Kaufmann Series in Data Management Systems, Elsevier Science, 2009.

[85] "Real-time systems," in *Encyclopedia of Computer Science* (E. D. Reilly, A. Ralston, and D. Hemmendinger, eds.), John Wiley & Sons, 4th ed., 2003.

[86] A. S. Tanenbaum, *Structured Computer Organization (5th Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2005.

[87] D. Zamboni, *Using Internal Sensors for Computer Intrusion Detection*. PhD thesis, Purdue University, West Lafayette, IN, 2001. AAI3055548.

[88] A. D. Keromytis, "Buffer Overflow Attacks," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 174–177, Boston, MA: Springer US, 2011.

[89] "Know Your Enemy: Sebek," *The Honeynet Project*, pp. 1–21, November 2003. `http://old.honeynet.org/papers/sebek.pdf` Report. Accessed: 2017-12-02.

[90] T. W. Curry, "Profiling and Tracing Dynamic Library Usage via Interposition," in *Proceedings of the USENIX Summer 1994 Technical Conference on USENIX Summer 1994 Technical Conference*, USTC'94, (Berkeley, CA, USA), pp. 18–18, USENIX Association, 1994.

[91] J. Mankin and D. Kaeli, "DIONE: A Flexible Disk Monitoring and Analysis Framework," in *Proceedings of the 15th International Conference on Research in Attacks, Intrusions, and Defenses*, RAID'12, pp. 127–146, Springer-Verlag (Berlin, Heidelberg), 2012.

[92] C. Spensky, H. Hu, and K. Leach, "LO-PHI: Low-Observable Physical Host Instrumentation for Malware Analysis," in *Proceedings of the Network and Distributed System Security Symposium*, The Internet Society, 2016.

[93] Z. Deng, X. Zhang, and D. Xu, "SPIDER: Stealthy Binary Program Instrumentation and Debugging via Hardware Virtualization," in *Proceedings of the 29th Annual Computer Security Applications Conference*, ACSAC '13, (New York, NY, USA), pp. 289–298, ACM, 2013.

[94] W. Lee, B. D. Payne, and M. Carbone, "Secure and Flexible Monitoring of Virtual Machines," in *23rd Annual Computer Security Applications Conference (ACSAC 2007)*, (Los Alamitos, CA, USA), pp. 385–397, IEEE Computer Society, 2007.

[95] B. Payne, S. Maresca, T. K. Lengye, and A. Saba, "LibVMI." GitHub Repository, 2016. `github.com/libvmi/libvmi` Software. Accessed: 2016-11-16.

[96] B. D. Payne, "Virtual Machine Introspection," in *Encyclopedia of Cryptography and Security* (H. C. A. van Tilborg and S. Jajodia, eds.), pp. 1360–1362, Boston, MA: Springer US, 2011.

[97] C. Xiao and J. Chen, "New OS X Ransomware KeRanger Infected Transmission BitTorrent Client Installer," *Palo Alto Networks Blog*, March 2016. `http://researchcenter.paloaltonetworks.com/2016/03/new-os-x-ransomware-keranger-infected-transmission-bittorrent-client-installer/` Blog. Accessed: 2016-06-06.

[98] A. Young and M. Yung, "Cryptovirology: Extortion-based security threats and countermeasures," in *Symposium on Security and Privacy*, pp. 129–140, IEEE, 1996.

[99] F. Sinitsyn, "TeslaCrypt 2.0 Disguised as CryptoWall," *Kaspersky Lab*, July 2015. Blog. https://securelist.com/blog/research/71371/teslacrypt-2-0-disguised-as-cryptowall/ Accessed: 2016-06-06.

[100] E. v. Dorp, "CryptoLocker – A New Ransomware Variant," *Emsisoft*, September 2013. Blog. https://blog.emsisoft.com/2013/09/10/cryptolocker-a-new-ransomware-variant/ Accessed: 2016-06-06.

[101] D. Bisson, "Under the Hood of Cryptowall 4.0," *Tripwire, Inc.*, February 2016. News article. http://www.tripwire.com/state-of-security/security-awareness/under-the-hood-of-cryptowall-4-0/ Accessed: 2016-06-06.

[102] M. Mallen, "No Mas, Samas: What's in this Ransomware's Modus Operandi?," *Microsoft Malware Protection Center*, March 2016. Blog. https://blogs.technet.microsoft.com/mmpc/2016/03/17/no-mas-samas-whats-in-this-ransomwares-modus-operandi/ Accessed: 2016-06-06.

[103] "The Current State of Ransomware: TorrentLocker," *Sophos*, December 2015. Blog. https://blogs.sophos.com/2015/12/23/the-current-state-of-ransomware-torrentlocker/ Accessed: 2016-06-06.

[104] Hasherezade, "Look Into Locky Ransomware," *Malwarebytes Labs*, March 2016. Blog. https://blog.malwarebytes.org/threat-analysis/2016/03/look-into-locky/ Accessed: 2016-06-06.

[105] B. Botezatu, "Linux Ransomware Debut Fails on Predictable Encryption Key," *Bitdefender Labs*, November 2015. Blog. https://labs.bitdefender.com/2015/11/linux-ransomware-debut-fails-on-predictable-encryption-key/, Accessed: 2017-12-01.

[106] "mbedtls – An Open Source, Portable, Easy to Use, Readable and Flexible SSL Library." GitHub Repository, https://github.com/ARMmbed/mbedtls, Commit 9fa2e86d93b9b6e04c0a797b34aaf7b6066fbb25, 2016. C source code.

[107] K. Baumgartner, "Sony/Destover: Mystery North Korean Actor's Destructive and Past Network Activity," *Kaspersky Lab*, December 2014. Blog. https://securelist.com/blog/research/67985/destover/ Accessed: 2017-05-06.

[108] D. Tarakanov, "Shamoon The Wiper: Further Details (Part II)," *Kaspersky Lab*, September 2012. Blog. https://securelist.com/blog/incidents/57784/shamoon-the-wiper-further-details-part-ii/ Accessed: 2017-05-06.

[109] T. Sammes and B. Jenkinson, *Forensic Computing: A Practitioner's Guide*. London, UK, UK: Springer-Verlag, 2000.

[110] "Naming Files, Paths, and Namespaces," *Microsoft Corporation*, 2016. Developer documentation. https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx Accessed: 2017-06-02.

[111] P. Gutmann, "Secure Deletion of Data from Magnetic and Solid-state Memory," in *Proceedings of the 6th Conference on USENIX Security Symposium, Focusing on Applications of Cryptography – Volume 6*, SSYM'96, (Berkeley, CA, USA), pp. 8–8, USENIX Association, 1996.

[112] G. Trant, J. Low, and D. v. Lith, "Eraser Appendix A: Erasure Methods." http://eraser.heidi.ie/appendix-a-erasure-methods/, 2016. Developer documentation. Accessed: 2016-11-27.

[113] M. Russinovich, "SDelete v2.0," *Microsoft Corporation*, July 2016. https://technet.microsoft.com/en-us/sysinternals/sdelete.aspx Software website. Accessed: 2017-12-01.

[114] A. Ziem, "BleachBit – Clean Your System and Free Disk Space." https://www.bleachbit.org/, 2016. Software website. Accessed: 2016-06-06.

[115] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, Nov. 2011.

[116] S. Garfinkel, P. Farrell, V. Roussev, and G. Dinolt, "Bringing Science to Digital Forensics with Standardized Forensic Corpora," *Digital Investigation*, vol. 6, no. Supplement, pp. S2 – S11, 2009. The Proceedings of the 9th Annual DFRWS Conference.

[117] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*. Springer Publishing Company, Incorporated, 2nd ed., 2010.

[118] "Redhat Enterprise Linux Security Guide – System Auditing," *Red Hat, Inc.* https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/chap-system_auditing.html Software documentation. Accessed: 2016-06-06.

[119] "GPU (Graphics Processing Unit)," in *A Dictionary of Computer Science* (A. Butterfield and G. E. Ngondi, eds.), Oxford University Press, 7th ed., 2016.

[120] J. Pepas, "Hexify: A Tiny Function Which Converts Binary Data Into Hex." GitHubGist Repository, https://gist.github.com/cellularmitosis/0d8c0abf7f8aa6a2dff3, 2016. C source code.

[121] "ZwWriteFile Routine," *Microsoft Corporation*, 2016. msdn.microsoft.com/en-us/library/windows/hardware/ff567121(v=vs.85).aspx Developer documentation. Accessed: 2016-11-27.

[122] "ZwSetInformationFile Routine," *Microsoft Corporation*. Developer documentation. msdn.microsoft.com/en-us/library/windows/hardware/ff567096(v=vs.85).aspx Accessed: 2016-11-16.

[123] G. C. Kessler, "File Signatures Table." http://www.garykessler.net/library/file_sigs.html, 2017. Technical Website. Accessed: 2017-06-02.

[124] R. Konishi, Y. Amagai, K. Sato, H. Hifumi, S. Kihara, and S. Moriai, "The Linux Implementation of a Log-structured File System," *SIGOPS Operating System Review*, vol. 40, pp. 102–107, July 2006.

[125] "PCMark 8 Technical Guide," *Futuremark Corporation*, April 2016.

[126] R. Falcone, "Second Wave of Shamoon 2 Attacks Identified," *Palo Alto Networks Blog*, January 2017. Blog. https://researchcenter.paloaltonetworks.com/2017/01/unit42-second-wave-shamoon-2-attacks-identified/ Accessed: 2017-11-29.

[127] K. Kasumu, "CrystalDiskMark." http://crystalmark.info/?lang=en, 2017. Software website. Accessed: 2017-06-02.

[128] B. Dolan-Gavitt, T. Leek, M. Zhivich, J. Giffin, and W. Lee, "Virtuoso: Narrowing the semantic gap in virtual machine introspection," in *2011 IEEE Symposium on Security and Privacy (SP)*, pp. 297–312, IEEE, 2011.

[129] "File Times," *Microsoft Corporation*, 2017. Software documentation. https://msdn.microsoft.com/en-us/library/windows/desktop/ms724290(v=vs.85).aspx Accessed: 2017-11-27.

[130] K. E. Heckman, F. J. Stech, R. K. Thomas, B. Schmoker, and A. W. Tsow, *Cyber Denial, Deception and Counter Deception: A Framework for Supporting Active Cyber Defense*. Springer Publishing Company, Incorporated, 1st ed., 2015.

[131] "Driver Signing," *Microsoft Corporation*. https://docs.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing Software documentation. Accessed: 2017-12-4.

[132] C. Brook, "VMWare Patchs Pwn2Own VM Escape Vulnerabilities," *Threatpost*, March 2017. Blog. https://threatpost.com/vmware-patches-pwn2own-vm-escape-vulnerabilities/124629/ Accessed: 2017-11-30.

APPENDIX

A ADDITIONAL PERFORMANCE RESULTS

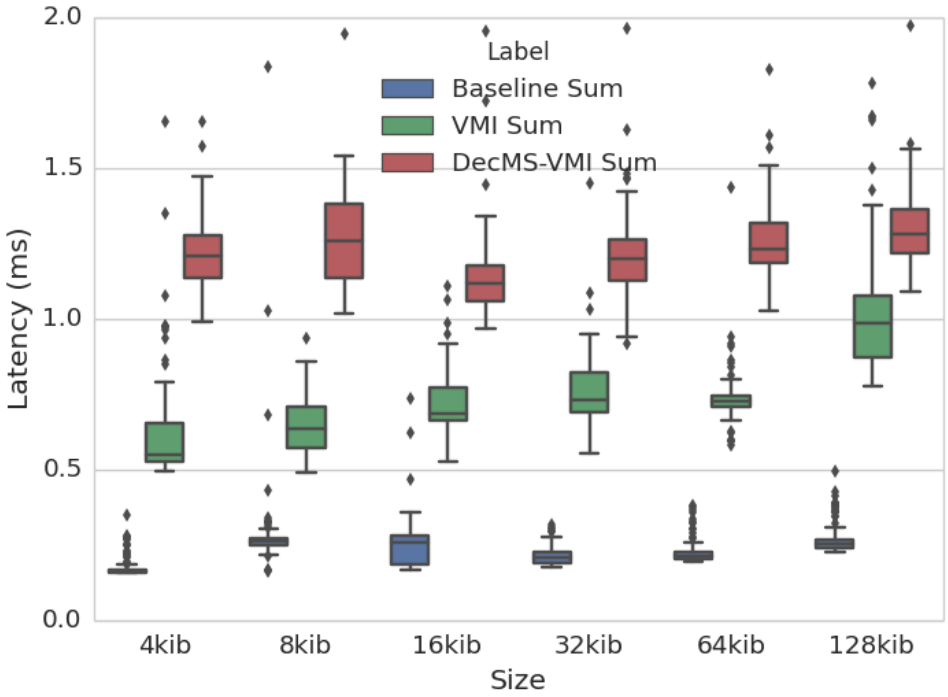


Figure A.1.: Write latency for file sizes between 4 KiB to 128 KiB for DecMS-VMI.

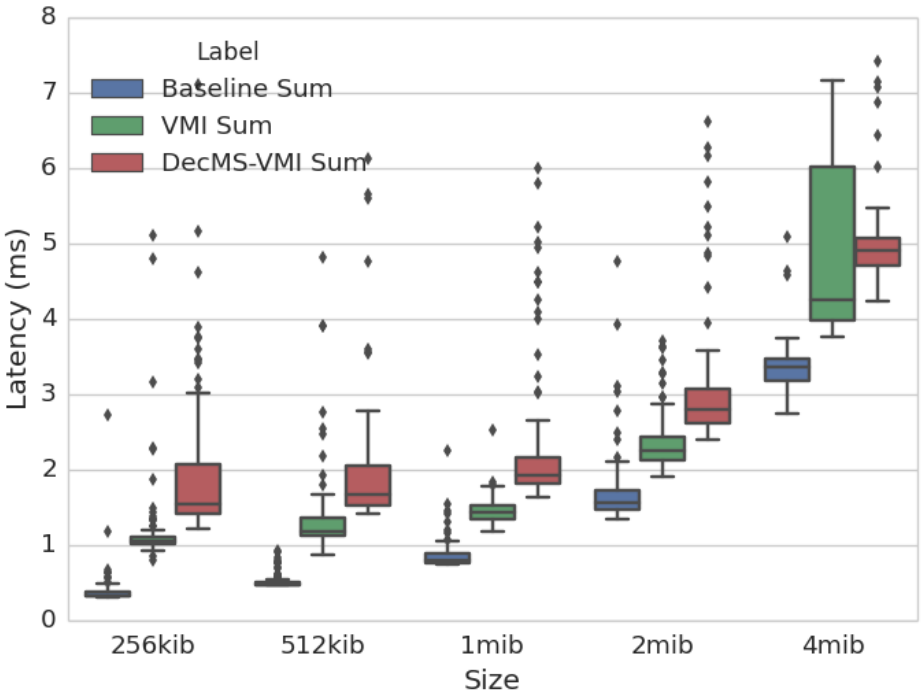


Figure A.2.: Write latency for file sizes between 256 KiB to 4 MiB for DecMS-VMI.

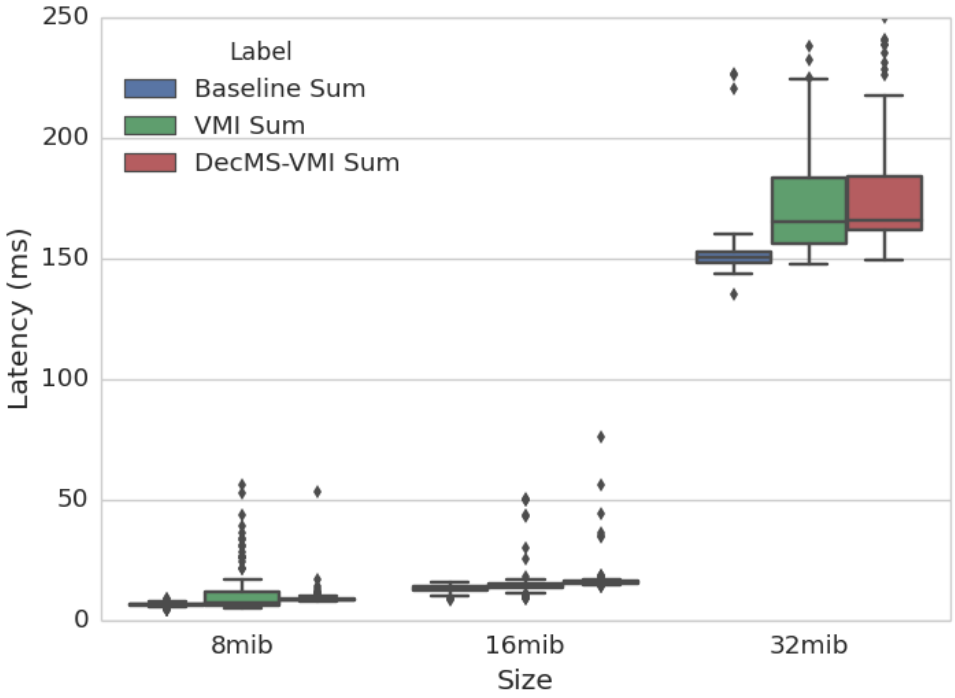


Figure A.3.: Write latency for file sizes between 8 MiB to 32 MiB for DecMS-VMI.

Table A.1: DecMS-VMI latency for 32 MiB files.

	BL O	BL W	BL C	BL S	DMS O	DMS W	DMS C	DMS S	VMI O	VMI W	VMI C	VMI S
count	100	100	100	100	100	100	100	100	100	100	100	100
mean	5.232	147.440	0.075	152.748	8.960	170.420	0.368	179.748	7.161	166.784	0.247	174.193
std	0.507	13.185	0.020	13.204	12.504	26.278	0.050	29.990	7.464	26.013	0.061	26.761
min	2.394	132.655	0.064	135.114	3.759	143.841	0.268	149.618	3.784	141.972	0.189	147.617
25%	5.064	142.710	0.070	148.056	5.698	155.732	0.338	161.909	5.141	150.799	0.228	156.298
50%	5.152	145.596	0.071	150.820	5.939	159.301	0.362	165.869	5.275	157.900	0.242	165.479
75%	5.241	147.701	0.072	153.033	6.317	175.143	0.389	184.170	5.469	170.403	0.249	183.494
max	7.138	221.694	0.243	226.769	105.405	291.480	0.573	295.797	52.550	306.449	0.757	310.991

Table A.2: DecMS-VMI latency for 4 KiB files.

	BL O	BL W	BL C	BL S	DMS O	DMS W	DMS C	DMS S	VMI O	VMI W	VMI C	VMI S
count	100	100	100	100	100	100	100	100	100	100	100	100
mean	0.138	0.049	0.042	0.230	0.360	0.715	0.330	1.406	0.286	0.192	0.193	0.672
std	0.525	0.013	0.008	0.533	0.142	1.116	0.054	1.150	0.420	0.043	0.059	0.455
min	0.075	0.042	0.038	0.157	0.265	0.387	0.262	0.989	0.184	0.147	0.144	0.497
25%	0.076	0.043	0.038	0.158	0.317	0.455	0.302	1.135	0.199	0.162	0.156	0.527
50%	0.078	0.043	0.038	0.162	0.334	0.539	0.317	1.207	0.219	0.176	0.170	0.552
75%	0.082	0.046	0.041	0.169	0.342	0.606	0.343	1.278	0.249	0.201	0.207	0.655
max	5.336	0.101	0.085	5.501	1.625	9.376	0.738	10.402	4.311	0.379	0.576	4.824

VITA

VITA

Christopher Noe Gutierrez is a first-generation college graduate and the first in his family to earn an advanced degree. He graduated from California State University, Bakersfield (CSUB), his hometown, in 2008 with a degree in computer science. While attending CSUB, Christopher tutored computer science and mathematics and was award several scholarships and research stipends, which piqued his interest in computer security. He was recognized by the CSUB Computer Science Department for Outstanding Academic Achievement and continued his studies at California State University Northridge (CSUN) with a scholarship from the LSAMP - Bridge to the Doctorate Program. Christopher earned a master's degree from CSUN and was awarded the Outstanding Academic Achievement from the Department of Computer Science.

Christopher earned a Ph.D. from Purdue University with funding through the Frederick N. Andrews Fellowship and research assistantships from the United States Missile Defense Agency, Northrop Grumman, and the National Science Foundation. In 2017, Christopher received the Diamond Award from Center for Education and Research in Information Assurance and Security (CERIAS). His general research interests include computing systems security and deceptive systems. Christopher accepted a Research Scientist position with the Security and Privacy Research Lab at Intel Corporation.