Available online at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Reactive redundancy for data destruction protection (R2D2)

Christopher N. Gutierrez *, *Eugene H. Spafford*, *Saurabh Bagchi*, *Thomas Yurek* [1]

Purdue University, CERIAS, 656 Oval Dr., West Lafayette, IN 47907, USA

ARTICLE INFO

Article history:
Received 30 September 2017
Received in revised form 12 December 2017
Accepted 25 December 2017
Available online 12 January 2018

Keywords:
Computer security
Virtual Machine Introspection
Operating systems security
Data integrity
Data availability
Data resiliency
Malware

ABSTRACT

Data destruction programs, such as Wiper Malware, cause substantial damage by overwriting critical digital assets on compromised machines, denying users access to computing resources. Our system, called R2D2, analyzes write buffers before they can reach a storage medium, determines if the write is destructive, and preserves the data under destruction. We interpose the inspection in the Virtual Machine Monitor (VMM) through a technique known as Virtual Machine Introspection (VMI). This has the benefit that it does not rely on the entire OS as a root of trust. We demonstrate the effectiveness of our prototype implementation by preserving data targeted for destruction by Wiper Malware such as Shamoon and Stonedrill, and a host of secure delete tools. We discover that R2D2 detects data destruction with high accuracy (99.8% true negative and true positive rates) and preserves critical data for all the Wiper Malware samples in the wild that we experimented with. While our prototype is not optimized for performance, we show that it is applicable for common user tasks in an office or home setting, with a latency increase of 1%–4% and 9%–20% to complete batch tasks and interactive tasks respectively. VMI accounts for 90.7%–98.5% of the latency overhead and thus R2D2 incurs a small cost for environments already using VMI.

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. Early examples date back to 1988 with programs like the Jerusalem Virus, which destroyed files on Fridays that fall on the 13th of each month (Solomon, 1993). Such malware is referred to as Wiper Malware because it aims to destroy files and make them unrecoverable. In the present day, loss of data, through an adversarial attack on data availability, utility, and integrity (Parker, 2012), is more than just a nuisance and can cause substantial damage. Recent 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 (Perloroth, 2012; Raiu et al., 2017).

Similar to Wiper Malware, Crypto Ransomware causes severe damage and has increased in usage among cyber criminals (Verizon Wireless, 2017). Rather than destroying critical data, Crypto Ransomware renders a target's computer or data unusable unless the user pays a fee. In May 2017, the WannaCry Crypto Ransomware reportedly spread onto 200,000 computers across 150 countries and is regarded as the

* *Corresponding author.*
E-mail addresses: gutier20@purdue.edu (C.N. Gutierrez), spaf@purdue.edu (E.H. Spafford), sbagchi@purdue.edu (S. Bagchi), tyurek@purdue.edu (T. Yurek).
[1] Present address: University of Illinois at Urbana-Champaign Dept. of Computer Science.
https://doi.org/10.1016/j.cose.2017.12.012
0167-4048/

most widespread instance of Crypto Ransomware to date (Piper, 2017).

Although the two types of malware differ in attack patterns, motivation, and end goals, there are similarities. Both Wiper Malware and Crypto Ransomware make data or computing services unavailable or unusable to users. Both spread quickly within a computing network and evade analysis to prolong their effectiveness. Both deny access to user data by overwriting critical files.

However, Wiper Malware differs from Crypto Ransomware in subtle ways that warrant investigation. Crypto Ransomware overwrites files with high entropy, because of encryption, while Wiper Malware may overwrite files with arbitrary data. Crypto Ransomware targets specific user files and typically leaves the compromised system in a running state to present the ransom note to the user. Wiper Malware typically is more indiscriminate in the files it destroys and does not care if the system is operational afterward. Our proposed system, Reactive Redundancy for Data Destruction (R2D2), reactively preserves data that is under active destruction. We evaluate our prototype and demonstrate that it is effective in preserving data against infamous Wiper Malware samples. We also show that our system is effective against other data destruction attacks, such as anti-forensic secure deletion algorithms. Our results demonstrate that we can accurately detect data destruction and preserve the critical data under destruction. We also demonstrate that the performance of R2D2 may be acceptable to users who are in need of data destruction protection.

Through our experimental evaluation, we discovered that some of the existing detection features for Crypto Ransomware apply to Wiper Malware and secure deletion tools. However, prior work (Continella et al., 2016; Kharaz et al., 2016; Scaife et al., 2016) protects against Crypto Ransomware by placing the monitoring and analysis within the OS where the attacker resides. Attackers may disable defenses such as anti-virus software, shadow copies, or checkpoints, as observed in Crypto Ransomware (e.g. TeslaCrypt (Sinitsyn, 2015)) and Wiper Malware (e.g. Jokra (McMillen, 2014)).

Our solution places R2D2 outside the operating system that is under protection. Our prototype implementation of R2D2 exists within a Virtual Machine Monitor (VMM), which monitors for data destruction in a guest virtual machine through Virtual Machine Introspection (VMI). VMI allows the VMM to access the contents of memory within a guest virtual machine, interposes events, and uses this information to "extrapolate the software state" of the virtual machine (Garfinkel and Rosenblum, 2003). Placing R2D2 outside the operating system under protection isolates the attacker from R2D2, significantly reducing the risk that an attacker can disable R2D2 from within the guest virtual machine. R2D2 is effective in an enterprise environment where users work within a Virtual Machine Guest Operating System hosted on a Virtual Machine Server to complement other VMM security monitoring tools.

Our novel contributions are the following:

1. We demonstrate effectiveness against recent Wiper Malware threats (Destover, Shamoon, Shamoon 2, and Stonedrill).
2. A system that successfully preserves data under destruction with high accuracy (99.8% true negative and true positive rates in our worst performing experiment) and acceptable performance for the home user or office related tasks (1%–4% additional latency for storage and 9%–20% additional latency for user tasks, which reduces to 0.1%–1.6% if the system is already using VMI).
3. Design and implementation separate the monitoring and the analysis components from the system under protection, an improvement over prior related work.
4. R2D2 does not rely on signature-based detection, but it is designed to operate in conjunction with other VMM security monitoring tools, such as signature-based anti-malware software or intrusion detection systems.
5. Our solution is policy driven, consisting of interposition, analysis, and preservation policies, allowing flexibility for practitioners.

2. Overview

R2D2 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 backup/redundancy technology. R2D2 observes persistent storage I/O and provides resiliency against destructive I/O. The high-level design of R2D2 is shown in Fig. 1. 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.

The top of Fig. 1 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). R2D2, located on the Trusted Monitoring System, (1) interposes when a file is open for writing to create a temporary checkpoint. (2) Next R2D2 interposes all writes to an existing file, determines if the I/O is destructive, and (3) the checkpoint is preserved 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 (Parker, 2012).

Each of the numbered items in Fig. 1 are policy driven, to provide flexibility in deployment. The Interposition Policy (Section 3.1) defines the files/directories under protection, and

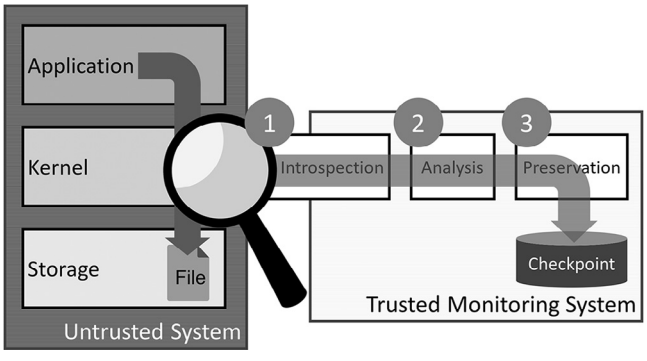

Fig. 1 – R2D2 interposes storage medium I/O in isolation. If the I/O appears to be destructive, R2D2 preserves the data.

the I/O related system calls to interpose. The Analysis Policy (Section 3.2) 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 3.3) describes how to preserve the data under destruction.

2.1. *Threat model*

For data destruction, an adversary does not simply delete data objects, but rather, *securely deletes* the data, rendering the data object unrecoverable. A standard delete command unlinks the data object from the file system, freeing the area where the data resided. The data may still reside on the storage medium if other files have not occupied the space. A secure delete overwrites the location of a data object, usually with a fixed sequence/pattern of bits, an arbitrary file, or random bits. We assume that the attacker overwrites files for data destruction and not through physical destruction.

Wiper Malware and secure delete tools use the above techniques for data destruction. For instance, Destover, the Wiper Malware that infamously infected Sony computers in 2014, overwrote master boot records with 64 KB of `0xAA` (Baumgartner, 2014). Shamoon destroyed data on a storage medium by overwriting with a JPEG file fragment (Tarakanov, 2012). Eraser, a freely available secure delete application, can overwrite with pseudorandom bits (Trant et al., 2016). 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 (Sammes and Jenkinson, 2000).

2.2. *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[1].
4. The VMM, analysis, and storage use by R2D2 are assumed to be within our Trusted Computing Base (TCB).
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 R2D2.
6. Our prototype implementation does not consider the case when valid users wish to securely destroy data. A discussion on the topic is presented in Section 6.

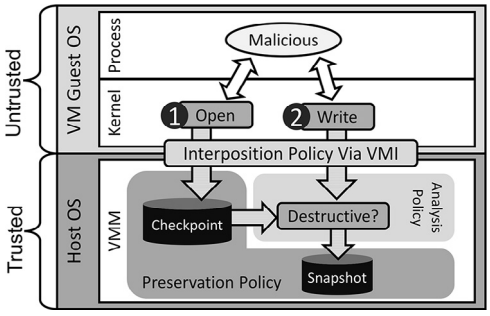

Fig. 2 – R2D2 examines file modifications and preserves the file if data destruction is suspected.

2.3. *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: R2D2 should impact latency and throughput for legitimate users as little as possible.

3. Prototype design and implementation

Our prototype implementation R2D2 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 (Fig. 1). The challenge with VMI is bridging the semantic gap, i.e., determining file activity from outside the operating system. Prior work (Lee et al., 2007) and an associated open source project (Payne et al., 2016) address semantic gap challenges, which security monitoring tools (Lengyel, 2016) use reliably.

Our evaluation of R2D2 protects a Windows[2] VM against data destruction and uses VMI to intercept system calls associated with storage medium I/O. The cost of R2D2 is a decrease in I/O performance. However, we show that the additional overhead may be applicable for some settings.

Fig. 2 illustrates the high-level design of R2D2 when placed within the VMM. When a process invokes a system call that writes to a file, R2D2 intercepts the system call and inspects the parameters (1). The Interposition 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

[1] For example, Kernel Object Hooking (KOH), Dynamic Kernel Object Manipulation (DKOM), or Direct Kernel Structure Manipulation (DKSM) (Jain et al., 2014; Kong, 2007) could be used to remain hidden from security monitoring.

[2] Our prototype and associated third party software protect Windows 7, but we do not anticipate significant obstacles to using R2D2 with more recent versions of Windows. Conceptually, it should also port to Linux, Mac OS, and other systems.

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 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.

3.1. *Prototype interposition*

3.1.1. *Method*

To intercept I/O system calls and inspect their parameters, we used the Drakvuf Dynamic Malware Analysis System (Lengyel et al., 2014). 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 calls, 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 R2D2 as a plug-in for Drakvuf.

3.1.2. *Policy*

Our R2D2 prototype intercepts two system calls on the guest VM: open file and write 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. Our 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, a policy determines if the file should be protected based on a blacklist or whitelist. Algorithm 1 contains pseudocode summarizing the Interposition Policy decision flow for open file system calls 3.1.

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, R2D2 takes a temporary checkpoint of the file system, and subsequent write system calls are analyzed, according to Analysis Policy, to determine if the write is suspect.

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
```

3.1.3. *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 Protection Policy is automatically triggered, which handles the preservation of the file that is open for writing. The blacklist for our evaluation consists of any references to the MBR. The MBR is an obvious target for Wiper Malware (Tarakanov, 2012). Destroying the MBR or partition table will render the storage medium unusable. The blacklist for our evaluation includes "e:," "PhysicalDisk1," and "Harddisk1" (Microsoft Corporation, 2016b), which point to NTFS data structures protected by R2D2.

Ignoring certain data objects, defined as the whitelist, could provide an increase in performance by avoiding analysis on 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 since 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.

R2D2 examines the following Windows system calls according to our Interposition Policy:

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

The `NtWriteFile` Windows system call writes data to an open file (Microsoft Corporation, 2016a). The `FileHandle`, `Buffer`, `ByteOffset`, and `Length` parameters are given to the Analysis Policy, if the file is not in the blacklist or whitelist, to determine if write is suspect of being destructive.

3.2. *Prototype analysis*

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 which is then used to decide if a write is suspect. Feature extraction should be quick to minimize analysis latency.

3.2.1. *Method*

The Interposition Policy provides the analysis 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.

Let B_i represent the write buffer provided by the Interposition 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 to 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 Fig. 3.

The sample offset, in conjunction with the offset provided by the Interposition Policy, defines locations of interests within the file. Several files use magic numbers (Sammes and Jenkinson, 2000) and metadata located at the beginning of a file.

There are several design choices that one must consider for the Analysis Policy. The sample size and sample frequency are

Table 1 – A list of secure delete methods considered in the evaluation of R2D2. "Random byte" is defined as a single random byte that overwrites the file. "Random data" overwrites the entire file with pseudorandom bits.

Algorithm	Description
AFSSI-5020	Three passes: random data, complement w/ 8-bit shift, complement w/ 16-bit shift (Trant et al., 2016)
AR 380-19	Three passes: random byte, random byte, complement of the second random byte (Trant et al., 2016)
British HMG IS5 (Baseline)	Single pass of zeros (Trant et al., 2016)
British HMG IS5 (Enhanced)	Three passes: zeros, ones, random data (Trant et al., 2016)
Canadian RCMP TSSIT OPS-II	Seven passes: three alternating passes of zeros and ones, then a random byte (Trant et al., 2016)
DoD 5220.22-M(ECE)	Seven passes: a combo of random bytes, complement of random bytes, and zeros (Trant et al., 2016)
DoD 5220.22-M (e)	Three passes: zeros, then ones, then random (Russinovich, 2016; Trant et al., 2016)
German VSITR	Same as Canadian RCMP TSSIT OPS-II (Trant et al., 2016)
Gutmann's 35-pass method	35 passes: data (1–4), fixed patterns (5–31), random data (32–35) (Gutmann, 1996)
Overwrite with zeros	Single pass of all zeros (Ziem, 2016)
Pseudorandom data	Overwrite with random bits (Trant et al., 2016)
Russian GOST P50739-95	Three passes: single pass of zeros, then random data (Trant et al., 2016)
Schneier's Algorithm (Schneier, 1996)	Seven passes: zeros, ones, remaining passes consist of random data (Trant et al., 2016)

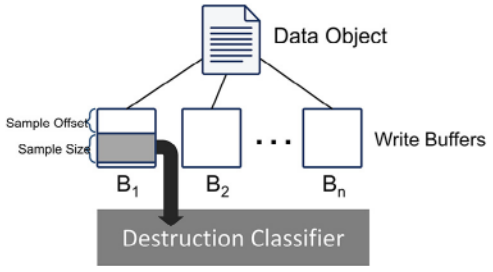

Fig. 3 – An illustration of write buffer sampling for R2D2.

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.

3.2.2. *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. Overwriting a file signature is a promising indicator for data destruction, as indicated in Scaife et al. (2016) for Ransomware. While some files, such as ASCII text files, do not use file signatures, multimedia files often do. When overwriting the beginning of a file, we check to see if the write buffer contains a file signature that matches the file extension[3]. We use the file signature corpus found in Kessler (2017). 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 1, 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 contain a fixed pattern, we flag the write a suspicious.

[3] An adversary may circumvent this analysis by writing a file signature that matches the file extension then overwriting the rest of the file. We do not observe this behavior in our evaluation and but nonetheless discuss the strategy in Section 6.

Randomness Testing Several data destruction tools use random binary data to overwrite files. Prior work in Continella et al. (2016), Kharaz et al. (2016), and Scaife et al. (2016) use entropy as a metric to identify encryption in Ransomware. We found that entropy can be problematic when trying to distinguish high entropy files, such as compressed files (e.g. zip and rar). Instead, we use a classification tree with features used in evaluating cryptographic pseudorandom number generators (Rukhin et al., 2010). 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 C. If the first 4 KiB of the write buffer appear to contain random binary data, we flag the write as suspicious.

Algorithm 2 shows the flow of our prototype implementation to handle 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 returns 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
end if
```

3.2.3. *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

Section 3.2.2. 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 Appendix C, the details regarding parameter constraints and features to produce our classification tree. We train and validate our classification tree on disjoint datasets. The classification tree can identify write buffers that contain random bits, with Precision and Recall above 99%.

3.3. *Prototype preservation*

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.

3.3.1. *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 storage 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 roll back 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, which indicates the version of the disk. 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 policy for the garbage collector can run periodically, have a minimum time to retain checkpoints, and run when the disk is nearly full.

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 (Konishi et al., 2006) 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.

3.3.2. *Implementation*

For our experimental evaluation, we take checkpoints synchronously when a file is open. When a checkpoint is required, the task is handed off 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.

4. Experimental evaluation

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 4096 MB 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 1333 MHz 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 R2D2 enabled but uses NILFS to create a periodic checkpoint. The baseline system is a fair comparison to R2D2 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 the optimal disk performance for the baseline. The hardware and VM configuration for the baseline and R2D2 are identical.

We evaluate the three requirements identified in Section 2.3: 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 Table 2, and the secure delete tools, listed in Table 1. Some of the secure delete tools use deterministic patterns for

Table 2 – Wiper Malware samples for evaluation.

Name	MD5
Destover	2618dd3e5c59ca851f03df12c0cab3b8
Shamoon	d214c717a357fe3a455610b197c390aa
Shamoon 2	2cd0a5f1e9bcce6807e57ec8477d222a
Stonedrill	0ccc9ec82f1d44c243329014b82d3125

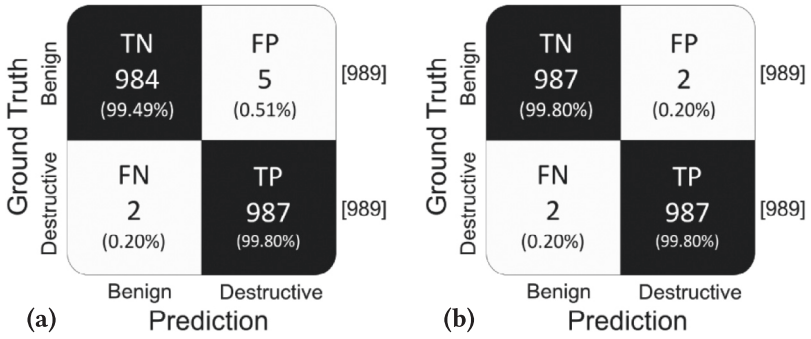

Fig. 4 – Confusion matrix for PRNG data destruction, the worst performing test. (a) Initial Results. (b) Corrected Results.

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 (Garfinkel et al., 2009).

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 R2D2 enabled. We also run PCMark 8 (Futuremark Corporation, 2016) to measure the performance impact of completing office tasks.

4.1. *Metrics of interest*

For the experimental evaluation of R2D2, two metrics of interest are calculated: the ability to accurately classify write buffers as "benign" or "destructive" and the impact on the latency associated with conducting such an analysis on a live system.

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.

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.

4.2. *Accuracy analysis*

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 R2D2 should not mistake writing these files as suspicious. The Benign files consist mostly of multimedia files such as PDF, HTML, and JPEG. Fig. A1 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 1.

R2D2 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*. Fig. 4a shows the confusion matrix for R2D2 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, R2D2 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 file system 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 Scaife et al. (2016), 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 Fig. 5. 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.

R2D2 also successfully detects all of the Wiper Malware in Table 2 *upon the first suspicious write*. Not only did we detect the first suspicious write for all of the samples, but we show in

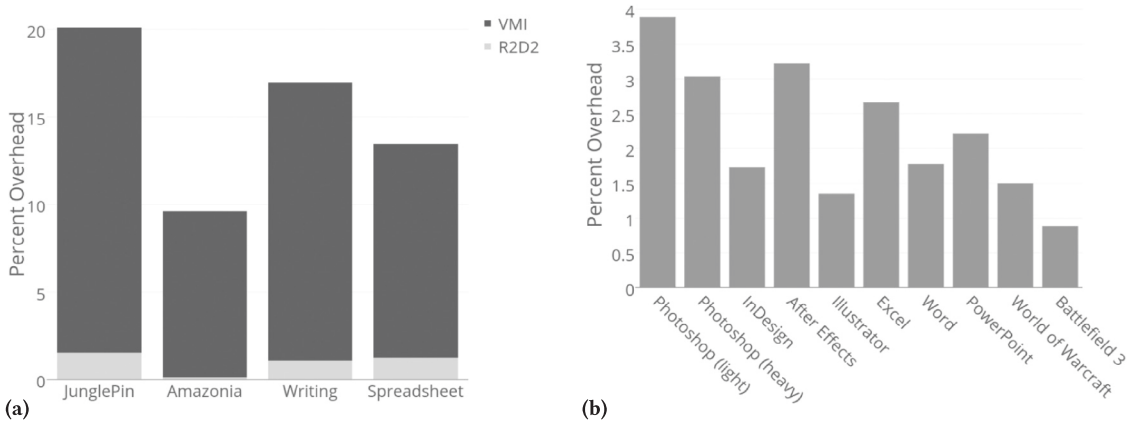

Fig. 5 – PCMark 8 office benchmark results showing the overhead incurred by R2D2 for a variety of common office tasks. The results are with respect to our baseline where no VMI but with NILFS in use. We observe that the bulk of the overhead arises from VMI provided by the Drakvuf analysis system (Lengyel, 2016). (a)Work Tests. (b) Storage Tests.

the following section that all of the files were successfully preserved without any unauthorized modification.

4.3. *Preservation under wipers*

To test if R2D2 can accurately preserve data under destruction, we test it against four Wiper Malware samples that have caused substantial damage to real systems. Table 2 lists all the malware we evaluate.

Our test environment consists of files found in GovDocs Thread9. The files in GovDocs Thread9 represent common files that a user may open or edit. The majority of the files consist of PDF documents, HTML files, and JPG images. Fig. A1 in Appendix C details the test files stored in the VM to evaluate the protection under Wiper Malware. We place all of the files under a protected directory in a separate disk image in our Windows VM. We also propagate the VM with synthetic data to make the VM appear to be a real system rather than a malware analysis system.

Software installation for the guest VM includes Google Chrome Web Browser, Sumatra PDF Reader, Notepad++ text editor, and WinRAR file archiver. File interaction with the above software is done on a subset of GoveDoc1 files to produce temporary files that are present on real computer systems. We also visit several websites with the Chrome Internet Browser. Additionally, the VM OS runs for several days before running any of the malware samples. 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 of the malware samples. The specific samples for the evaluation invoke the wiper modules, which may not include anti-analysis methods as it is the last step of the attack. Nonetheless, the setup for the guest OS successfully runs the data destruction for each malware sample considered.

The malware samples use two approaches to destroy files on the system: overwriting each user file or bypassing the file system and overwriting the raw disk. The Shamoon, Shamoon 2, 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 the mismatching 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 R2D2 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.*

4.4. *Performance analysis*

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

4.4.1. *Benign activity*

For benign user activity, we evaluate the performance by using PCMark 8 (Futuremark Corporation, 2016). PCMark 8 is a benchmark suite to measure system performance under common user tasks. We evaluate R2D2 using the "Work Benchmark" test suite which consists of web browsing, document processing, and spreadsheet editing[4]. These tests mimic how real users interact with the software and measure user interface and runtime latencies. We briefly summarize each of the PCMark 8 Work Benchmark test for the evaluation in Appendix A. For details, refer to the PCMark 8 Technical Guide (Futuremark Corporation, 2016).

Fig. 5a illustrates the overhead of R2D2 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 R2D2 Analysis Policy. The plot shows that the majority of the overhead is from VMI

[4] 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.

for all tests, with R2D2 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 R2D2 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 affects 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 R2D2 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," which measures storage latency performance, to compare the latency of the baseline and R2D2. The Storage Benchmark simulates disk usage and does not simulate the UI, which is the main distinction compared to the Work Benchmark. A summary of the Storage Benchmark test is described in Appendix B.

Fig. 5b shows the median overhead for storage I/O for R2D2. The Adobe Photoshop and AfterEffects Test performed the worst with a median overhead of 3%–4%. Based on the description of the tests (Futuremark Corporation, 2016), 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 R2D2 does not interpose file read system calls. Based on the results, it appears that R2D2 introduces a latency increase of between 1% and 4% on disk storage.

To confirm that VMI interposition adds overhead to the entire system (including the UI), we ran R2D2 using the CrystalDiskMark 5.2.1 (Kasumu, 2017) throughput, a Windows disk benchmarking application, to measure reading performance under R2D2. 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 R2D2.

We also examined the write throughput under R2D2, 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 R2D2. However, 95.4% of the performance reduction was caused by VMI. Nonetheless, these results show that R2D2 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%.

4.4.2. *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 R2D2 analysis.

Fig. 6 contains the latency results of writing a fixed file size that triggers R2D2 to convert a checkpoint into a snapshot. Recall that R2D2 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

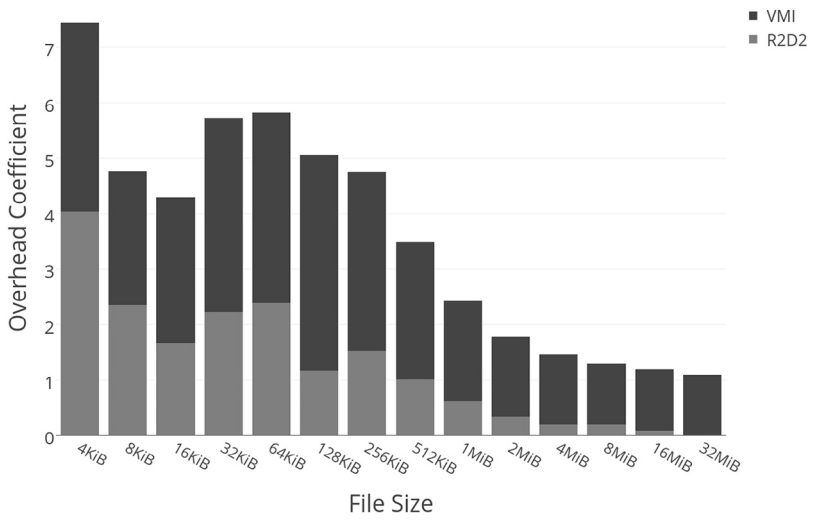

Fig. 6 – Median Latency introduced by R2D2 and VMI when data destruction is suspect.

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 (Fig. 4b) 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.

4.4.3. *Summary of results*

Our results show that R2D2 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 R2D2 analysis. For benign user tasks, the R2D2 analysis only accounts for 1.4%–9.29% (Fig. 5a) of the total latency introduced, which is between 0.13% and 1.54% additional overhead compared to the baseline system. There has already been more than a 5× reduction in VMI overhead from 2004 to 2011 (Dolan-Gavitt et al., 2011; Garfinkel and Rosenblum, 2003) and as VMI mechanisms continue to improve in performance, R2D2 will also inherit the benefits.

5. Alternative methods and policies

Our experimental evaluation is on a single design choice of R2D2, and a policy that is intended to provide reasonable performance while protecting files within a VM, based on the current threat landscape. Below, we discuss design alternatives that may provide performance improvements as well as analysis policy alternatives, some of which are from related work, which protects against Crypto Ransomware. Nonetheless, the design alternatives for the analysis policies may help protect against Wiper Malware that uses data encryption to destroy files, which reportedly (Ivanov and Mamedov, 2017; Suiche, 2017) is the case for the Petya malware.

5.1. *Placement of R2D2*

There are several alternative interposition methods for R2D2 with several advantages or disadvantages. In practice, the design choices should reflect the needs of the system. For our experimental evaluation, we felt that placing R2D2 within the VMM may provide a balance of performance and isolation from the attacker. While our results show that there is moderate latency, which may be unsuitable for some applications, R2D2 can be placed in other locations to improve performance.

For instance, R2D2 can interpose writes to the storage medium from within the operating system. Placing R2D2 within an operating system avoids the overhead associated with VMI, which we have seen is by far the most significant contributor overhead. However, the major disadvantage of this approach is that the controls to the file versioning system, and other R2D2 mechanisms, are located within the protected system and not isolated from the (potentially) untrusted operating system. If an adversary can compromise the OS, he or she can disable R2D2.

Another plausible placement for R2D2 is near the hardware. A hardware controller on a storage device could inspect the write buffers for destructive features before writing the data to the storage device. However, the challenge of placing R2D2 within a hardware controller is to log and preserve data that is meaningful for a user, system administrator, or forensic examiner. Mapping low-level disk operations to files is a well-known semantic gap problem for disk storage (Jiang et al., 2007; Mankin and Kaeli, 2012). Alternatively, a network file system can also work with R2D2, similar to the work in Strunk et al. (2000), which inspects file interactions for malicious behavior. One advantage is that R2D2 can be isolated from the attacker unless he or she can overwrite the hardware controller or gains access to the network file storage server.

5.2. *Analysis policy*

An effective tool in identifying Crypto Ransomware are read/write patterns, as shown in Continella et al. (2016), Kharaz et al. (2016), and Scaife et al. (2016). Ransomware follows several predictable read/write patterns, such as overwriting the file with the encrypted version or copying the encrypted file elsewhere and destroying the original file (Kharaz et al., 2016).

Read/write patterns may be applicable in detecting Wiper Malware. However, it appears that Wipers overwrite files and not necessarily read the file contents before destruction. Other possible detection metrics include the process-centric approach in Continella et al. (2016), which measures the rate at which a process writes to files.

File type funneling quantifies the number of file types read and written to a storage medium per process (Scaife et al., 2016). Other patterns, such as directory listing, file type coverage, and file renaming, are features for the classifier in Continella et al. (2016). File-centric features may not be applicable for Wiper Malware. Wipers may write directly to a storage medium, bypassing file system conventions (Baumgartner, 2014; Raiu et al., 2017; Tarakanov, 2012). 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 for a raw storage medium. Prior work (Mankin and Kaeli, 2012) 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 knowledge of how the tool works ahead of time. For instance, the sdelete secure delete tool renames a file 26 times before destruction (Russinovich, 2016). 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 (Garfinkel and Rosenblum, 2003), that we do not consider in our experimental evaluation. The risk is small for R2D2 as false positives only generate an additional snapshot of the file system, which we show has acceptable overhead in our assessment. 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 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[5]. Several examples (Raiu et al., 2017; Tarakanov, 2012) restart the compromised machine, which displays a Master Boot Record error.

6. Discussion and limitations

The attacker may try to execute writes to the disks without triggering the snapshot/checkpoint creation of R2D2. There are three obvious attack strategies, observed in real malware, which are (i) to remain hidden from security monitoring tools from within the OS, (ii) run the data destruction before booting the operating system, or (iii) avoid the behaviors and heuristics that cause the snapshot to trigger.

6.1. *Hide from monitoring*

Attackers have an array of techniques to hide from security monitoring tools and system administrators. 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. 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 his or her own privileged I/O software to avoid the R2D2 system call monitoring altogether. In addition to writing the I/O software, the attacker must circumvent software integrity and authenticity validation[6], 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 monitoring. In the current prototype configuration of R2D2, 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 R2D2 builds. The System Service Descriptor Table (SSDT) Monitoring plug-in allows Drakvuf to detect if the system call table within Windows is modified. The SSDT Monitoring plug-in can be extended to work with R2D2 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 R2D2, or VM security monitoring software, fails 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 R2D2. The attacker may alter his or her destructive behavior to avoid the snapshot trigger or destroy the data from a location that the current configuration of R2D2 cannot observe.

6.2. *Out-of-band destruction*

Another method to circumvent monitoring 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 master boot record 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 MBR, which is detected and in principle should work the same if the malware overwrites the bootloader. However, a limitation of R2D2, assuming the KOH/DKOM detection methods fail, is if the attacker replaces the MBR or the bootloader by circumventing the VMI interposition, then he or she may be able to destroy files without triggering checkpoints/snapshots. Under the experimental configuration of R2D2, an attacker may be able to replace the bootloader without detection if he or she can execute before R2D2 is enabled. The experimental evaluation assumes that R2D2 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 R2D2 is enabled.

Bacs et al. (2016) successfully demonstrate the ability to modify changes to virtual disks at specified locations such as the MBR or bootloader. The work observes specific regions of the virtual disk for any modification without relying on VMI. One solution is to combine the detection mechanism in Bacs et al. (2016) with the preservation methods in the work proposed here. Before modifying the bootloader, create a snapshot so that the file system is recoverable if the modification is malicious.

6.3. *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.

[5] There are exceptions (Baumgartner, 2014) if the attacker wishes to let the defenders know of the destruction and convey a message.

[6] Such as Windows 10 driver verification (Microsoft Corporation, 2017).

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 4096 KB and when writes occur at the beginning of a file, as a trade-off of robustness for speed. Currently, the experimental Analysis Policy is easily 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 Interposition Policy can monitor the NtSetInformationFile system call, which renames and moves files, to mitigate the attack. R2D2 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 evaluation of R2D2 demonstrates the feasibility of protecting files under destruction under several malware samples and data destruction tools. Our evaluation does not observe the above behavior, which may be used to circumvent R2D2. Further enhancements to R2D2 are necessary to provide sufficient protection against file movement or renaming attacks, which is left for future work.

The weaknesses above are not exclusive to R2D2 but all security monitoring tools. Attackers are persistent and eventually discover new methods to avoid monitoring. However, the design of R2D2 increases the challenge of unauthorized data destruction by isolating the snapshot/checkpoint mechanism and preserving the data under destruction that is out of reach for the attacker. Some policy changes can help mitigate destructive attackers by periodically taking snapshots regardless of the changes observed, a standard configuration of NILFS. Further, R2D2 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 (Garfinkel and Rosenblum, 2003), making the detection of R2D2 from the attacker's perspective more difficult. Further, injecting deceptive faults into the VM to disrupt the data destruction 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. (2015). For opportunistic adversaries, the confusion may cause the attacker to look elsewhere for a vulnerable system, which is an advantage for the defenders.

6.4. *Other limitations*

A valid user may have a legitimate reason to destroy or encrypt data, and that should not trigger R2D2. 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 and is left for future work. However, some possible solutions are briefly described below.

One simple solution to support valid data destruction or encryption is through a manual out-of-band mechanism whereby the user notifies the system administrator to disable R2D2 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.

Alternatively, software within the guest VM could communicate with the R2D2 through the use of a time-based one-time password (TOTP[7]) algorithm. The TOTP could be generated outside the guest OS, say through a smartphone application or keyfob, where the attacker does not reside. The user then provides the TOTP to a trusted encryption application within the guest OS. R2D2 verifies the TOTP and the correctness of the encryption application. The R2D2 snapshot mechanism is then disabled to allow for encryption temporarily. The above mechanism allows trusted users to destroy data within the system. However, if the TOTP password fails, a snapshot of the file system is taken.

Alternatively, the TOTP could be replaced with a challenge–response scheme to authenticate that a valid user is encrypting or destroying data within the system. Hardware support could also help provide encryption or data destruction. Requests for encryption/destruction could be handled by a hardware component, such as a hardware security module, such that the storage of the decryption key is inaccessible from the Guest OS. If the user wishes to decrypt data within the Guest OS, she must authenticate with R2D2 before the decrypted file is accessible from within the guest OS. The above scheme works only if the user is trusted and is not conducting on data destruction attacks.

The extra storage requirement of R2D2 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 snapshots should be much smaller than the total size of all protected directories, and so the extra storage needed to use R2D2 is comparatively small, depending heavily on entropy of the writes.

Our prototype implementation of R2D2 only supports the snapshot/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 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 R2D2 is if destruction actions are intermingled with benign writes to the storage medium. It may be difficult to manage and revert the

[7] RFC6238: https://tools.ietf.org/html/rfc6238.

effects of destructive actions when benign applications are affected by the destruction. Fortunately, there are existing systems that help reduce the severity and work well with R2D2. For instance, the Taser Intrusion Recovery System (Goel et al., 2005) uses taint analysis to help identify and revert file changes after an attack discovery. Taser also resolves the conflicts that may occur when a tainted file or process interacts with a benign file or process. Taser requires snapshots which are already in place in R2D2. However, incorporating R2D2 will require additional engineering effort to work with a taint analysis system in future work.

7. Related work

Early works (2003) presented in Pennington et al. (2010) and Strunk et al. (2000) 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) (Strunk et al., 2000) 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 System (SBIDS) (Pennington et al., 2010), which observes disk access patterns for unauthorized actions. SBIDS watches 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 to detect unauthorized modifications to date and time stamps, deletion of log records in log files, and the creation of hidden files.

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. Further, both SBIDS and S4 do not provide detection indicators for Wiper Malware, data destruction anti-forensic tools, or Crypto Ransomware. While the implementation of S4 and SBIDS enhance NFS specifically, the authors in SBIDS suggest that their design of "compromised independence" can be actualized through a VMM, which we explore. Placing data destruction protection mechanisms within a VMM has its advantages when combined with VMI. Particularly, our implementation reports the active running process, date/timestamp, and user information that is responsible for the data destruction. Additional logging information is extendible to the security administrator's discretion.

Several detection methods from prior work inspire the approach in R2D2. Unveil (Kharaz et al., 2016) monitors for stealthy Ransomware within an anti-malware analysis system which monitors for writes that have high entropy as a sign of unauthorized data encryption. CryptoDrop (Scaife et al., 2016) uses file signatures and entropy as part of its detection features. Rather than using entropy, we found that our randomness test produces low false positives across a wide range of secure delete tools and Wiper Malware.

Closely related to our work, *ShieldFS* (Continella et al., 2016) protects a Windows OS against Crytpo Ransomware through the usage of a Copy on Write (CoW) system, which recovers files if a rogue process encrypts them. 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.

Unfortunately, there are a variety of methods to destroy data. As discussed throughout the paper, data destruction methodologies are more open-ended compared to Crypto Ransomware, which contributes to the challenge of protecting user files from Wiper Malware. The work presented here focuses on Wiper Malware while other work focuses on unauthorized data encryption (Continella et al., 2016; Kharaz et al., 2016; Scaife et al., 2016). Other forms of data destruction, such as unauthorized data replacement that compromises the authenticity of a file are left for future work.

The ShieldFS (Continella et al., 2016) inspired the design and architecture decisions. In particular, we found the proactive shadow copying of user files, in case of unauthorized encrypting, worked well in protecting files from destructive malware. 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, just as in S4 (Strunk et al., 2000) and SBIDS (Pennington et al., 2010). ShieldFS provides some defense against an attacker who attempts to disable ShieldFS. 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 design ultimately led to the decision of using a log-structured file system as in Strunk et al. (2000) to allow quick checkpoint creation and recovery of user files.

Drakvuf (Lengyel et al., 2014) is capable of capturing deleted files from memory by intercepting specific system calls. Drakvuf checks the system call parameters to determine if the file is cached in memory and set for deletion, a strategy used by malware droppers. If the conditions are met, Drakvuf calls Volatility from the VMM to save the file from memory before it is deleted. In our evaluation of R2D2, we explore adversaries who use secure delete methods to destroy data objects, rather than referencing objects in-memory. Since R2D2 is a plug-in for Drakvuf, it is possible to run R2D2 in conjunction with the in-memory file recovery.

8. Conclusion

Data destruction programs, such as Wiper Malware, cause substantial damage by overwriting critical digital assets on compromised machines, affecting the availability, utility, and integrity of critical files. We take inspiration from some recent work on detecting and preserving data in the face of Ransomware but realize that there are some fundamental differences between the two. We successfully demonstrate that some of the features from prior work may help identify data destruction attacks invoked by several Wiper Malware samples and we improve the design by isolating the analysis and data preservation away from the attacker.

Our system, R2D2, analyzes write buffers before they reach a storage medium, determines if the write is destructive, and preserves the data under destruction. We interpose the inspection in the Virtual Machine Monitor (VMM) through a technique known as Virtual Machine Introspection (VMI). This has the benefit that it does not rely on the entire OS as a root of trust, unlike prior systems that protect Crypto Ransomware, due to the isolation of the analysis and monitoring within a VMM. We demonstrate the effectiveness of our prototype implementation by preserving data targeted for destruction by Wiper Malware such as Shamoon and Stonedrill, and a host of secure delete methods. We discover that R2D2 detects data destruction for all of the secure delete methods considered, except for one, where its detection percentage is 99.8%. With a host of benign file operations, R2D2 suffers only a 0.2% false positive rate. We see that R2D2 can successfully preserve the integrity of the data under destruction against the latest wave of Wiper Malware, such as Shamoon 2 and Stonedrill, and provide an acceptable performance under several office and home use cases. While our prototype implementation is not optimized for performance, the overhead associated with R2D2 analysis contributes to about 0.1%–1.6%; an additional 9.5%–18.6% overhead is due to VMI. Efforts in reducing the overhead of VMI will directly benefit R2D2 and allow it to be used in scenarios where higher storage throughput is required than in a home or an office setting.

Acknowledgments

The authors would like to thank the National Science Foundation for supporting this research through the EAGER Grant #1548114 and Adam Hammer at the Center for Education and Research in Information Assurance and Security (CERIAS) for his technical assistance. Thank you Mathias Payer for your valuable feedback and discussion on topics presented in this paper.

Appendix A

PCMark work benchmark summary

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.

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.

Appendix B

PCMark storage benchmark summary

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.

Appendix C

Randomness classifier

The classification tree training uses the `scikit-learn` Python library (Pedregosa et al., 2011), which uses an optimized version of 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 at most of four randomness features and four Boolean statements.

For each interposed write buffer, we must consider the *sample size, sample offset, and sample frequency* for system write calls issued to a file. 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-segment of the write buffer, or combine several write buffers. We constrain the maximum buffer size as the smallest file size in Windows, 4096 bytes. We validate our classifier by exploring several buffer sizes between 16 bytes to 4096 bytes and identify a buffer size which provides high accuracy while not blowing the latency overhead.

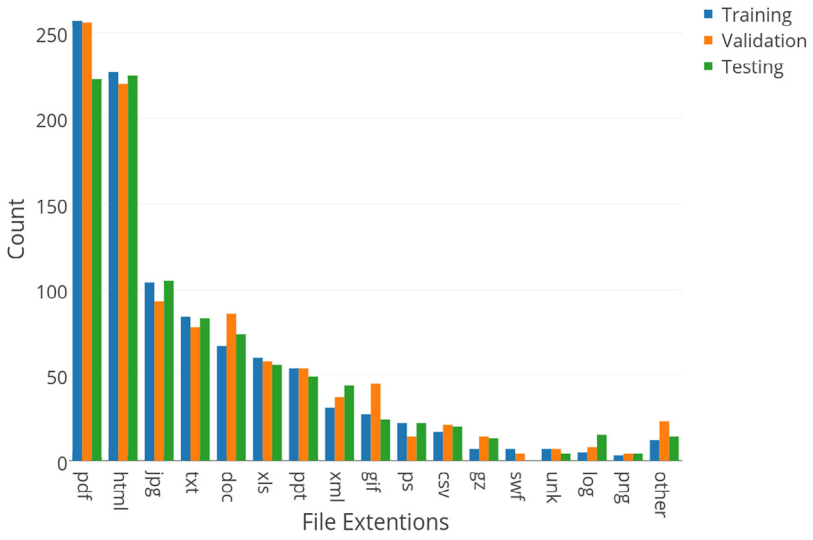

Fig. A1 – The distribution of files in training, validation, and testing sets.

The features for our classification tree consist of p-values returned by NIST randomness tests (Rukhin et al., 2010). Our evaluation shows that even with four randomness tests (or fewer), FP/FN rates 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 sample 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. However, an adversary can adjust their strategy to circumvent the conservative sample frequency. A discussion is provided in Section 6.

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 4.

For the Benign class, we use the forensics files corpora detailed in (Garfinkel et al., 2009). Specifically, R2D2 is evaluated against the Govdocs1[8] dataset, which consists of files gathered from .gov domains. The full distribution of training, validation, and testing sets are show in Fig. A1.

The *Govdocs1* dataset provides ten "threads," each consisting of about 1000 randomly selected files from the entire corpora. The intention for the "threads" is for researchers to select distinct threads for training, validation, and testing.

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 with pseudorandom bits using the shred (Free Software Foundation, Inc, 2016) utility. The training and validation sets each consists of an equal number of Benign and Destructive class samples.

Parameter settings for randomness classifier

For our evaluation, we measure the accuracy with increasing write buffer sizes for R2D2. We find the optimal buffer size by evaluating the precision and recall for buffers ranging from 16, 32, …, 4096 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 1, 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 a distribution to add to the entropy of the protected system.

Randomness tests

Through our evaluation, we decided on a tree that uses four different randomness tests that provide high accuracy and low computational cost: The 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. NIST recommends that this test be conducted first because if this test fails, other randomness tests are likely to fail as well.

*Frequency Test Within a Block*Rather than comparing the ratio of one-bit to zero-bit over the entire sequence, the Frequency

[8] http://digitalcorpora.org/corpora/govdocs.

Fig. A2 – Recall Score on validation set for increasing buffer sizes and maximum depth of tree parameters. For our experimental evaluation, we select the parameters of tree depth of two and buffer size 4096 bytes. (For interpretation of the references to color in the discussion of this figure, the reader is referred to the web version of this article.)

Fig. A3 – Precision Score on validation set for increasing buffer sizes and maximum depth of tree parameters. For our experimental evaluation, we select the parameters of tree depth of two and buffer size of 4096 bytes. (For interpretation of the references to color in the discussion of this figure, the reader is referred to the web version of this article.)

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.

*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" (Rukhin et al., 2010). The minimum sample size is 100 bits.

*Longest-Run-of-Ones in a Block Test*The longest-run-of-ones 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 (Rukhin et al., 2010).

Classification training and validation

Figs A2 and A3 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 represents 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 above 512 bytes 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

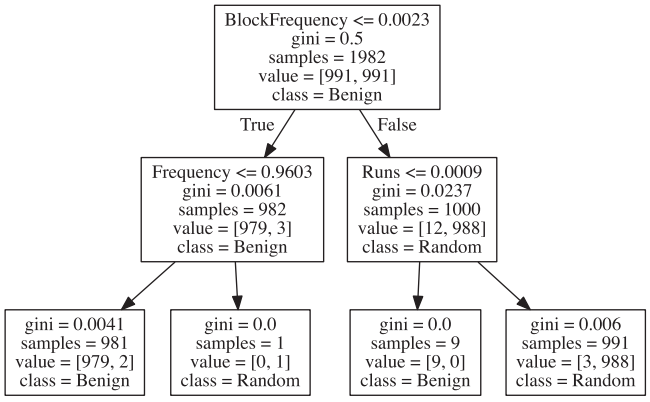

Fig. A4 – Classification tree selected for testing with input buffer size of 4096 bytes and tree depth of two.

increases. Note that even while inspecting small write buffers between 16 to 256 bytes, R2D2 is capable of recall rate above 0.9 and precision rate above 0.84.

The box highlighted in red in Figs A2 and A3 indicates the classification parameters selected for our testing of R2D2. We select a buffer size of 4096 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. Fig. A4 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 (Rukhin et al., 2010). 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.

REFERENCES

Bacs A, Giuffrida C, Grill B, Bos H. Slick: an intrusion detection system for virtualized storage devices. In: Proceedings of the 31st annual ACM Symposium on Applied Computing (SAC '16). New York: ACM; 2016. p. 2033–40. https://doi.org/10.1145/2851613.2851795.

Baumgartner K. Sony/Destover: mystery North Korean actor's destructive and past network activity. Technical Report. Kaspersky Lab; 2014. Available from: https://securelist.com/destover/67985/. [Accessed 6 May 2017].

Continella A, Guagnelli A, Zingaro G, De Pasquale G, Barenghi A, Zanero S, et al. ShieldFS: a self-healing, Ransomware-aware file system. In: Proceedings of the 32nd Annual Conference on Computer Security Applications (ACSAC '16). New York: ACM; 2016. p. 336–47. https://doi.org/10.1145/2991079.2991110.

Dolan-Gavitt B, Leek T, Zhivich M, Giffin J, Lee W. Virtuoso: narrowing the semantic gap in virtual machine introspection. In: Security and Privacy (SP), 2011 IEEE symposium on. IEEE; 2011. p. 297–312.

Free Software Foundation, Inc. GNU Coreutils 11.6 shred: remove files more securely. Free Software Foundation, Inc. Technical Manual; 2016. Available from: https://www.gnu.org/software/coreutils/manual/html_node/shred-invocation.html. [Accessed 27 November 2016].

Futuremark Corporation. 2016 a. PCMark 8 technical guide; 2016.

Garfinkel S, Farrell P, Roussev V, Dinolt G. Bringing science to digital forensics with standardized forensic corpora. Digit Investig 2009;6:S2–11.

Garfinkel T, Rosenblum M. A virtual machine introspection based architecture for intrusion detection. Proc Netw Distrib Syst Secur 2003;1:253–85. https://doi.org/10.1109/SP.2011.11.

Goel A, Po K, Farhadi K, Li Z, de Lara E. The Taser intrusion recovery system. In: Proceedings of the twentieth ACM Symposium on Operating Systems principles (SOSP '05). New York: ACM; 2005. p. 163–76. https://doi.org/10.1145/1095810.1095826.

Gutmann P. 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 – (SSYM'96), vol. 6. Berkeley (CA): USENIX Association; 1996. p. 8.

Ivanov A, Mamedov O. ExPetr/Petya/NotPetya is a wiper, not ransomware. Technical Report. Kaspersky Lab; 2017. Available from: https://securelist.com/expetrpetyanotpetya-is-a-wiper-not-ransomware/78902/. [Accessed 17 September 2017].

Jain B, Baig MB, Zhang D, Porter DE, Sion R. SoK: introspections on trust and the semantic gap. In: 2014 IEEE symposium on security and privacy. 2014. p. 605–20. https://doi.org/10.1109/SP.2014.45.

Jiang X, Wang X, Xu D. Stealthy malware detection through VMM-based out-of-the-box semantic view reconstruction. In: Proceedings of the 14th ACM conference on Computer and communications security. ACM; 2007. p. 128–38.

Kasumu K. CrystalDiskMark; 2017. Available from: http://crystalmark.info/?lang=en. [Accessed 2 June 2017].

Kessler GC. File signatures table; 2017. Available from: http://www.garykessler.net/library/file_sigs.html. .

Kharaz A, Arshad S, Mulliner C, Robertson W, Kirda E. UNVEIL: a large-scale, automated approach to detecting ransomware. In: 25th USENIX security symposium (USENIX security 16). Austin (TX): USENIX Association; 2016. p. 757–72. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/kharaz.

Kong J. Designing BSD rootkits. San Francisco (CA): No Starch Press; 2007.

Konishi R, Amagai Y, Sato K, Hifumi H, Kihara S, Moriai S. The Linux implementation of a log-structured file system. SIGOPS Oper Syst Rev 2006;40(3):102–7. https://doi.org/10.1145/1151374.1151375.

Lee W, Payne BD, Carbone M. Secure and flexible monitoring of virtual machines. Comput Secur Appl Conf Ann 2007;385–97. https://doi.org/doi.ieeecomputersociety.org/10.1109/ACSAC.2007.10.

Lengyel TK. Drakvuf. GitHub Repository; 2016. Available from: https://github.com/tklengyel/drakvuf. .

Lengyel TK, Maresca S, Payne BD, Webster GD, Vogl S, Kiayias A. Scalability, fidelity and stealth in the DRAKVUF Dynamic Malware Analysis System. In: Proceedings of the 30th annual computer security applications conference. 2014.

Mankin J, Kaeli D. Dione: a flexible disk monitoring and analysis framework. Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 127–46. https://doi.org/10.1007/978-3-642-33338-5_7.

McMillen D. Wiper malware analysis – research and intelligence report. Technical Report. IBM MSS; 2014.

Microsoft Corporation. ZwWriteFile Routine; 2016a. Available from: msdn.microsoft.com/en-us/library/windows/hardware/ff567121(v=vs.85).aspx. [Accessed 27 November 2016].

Microsoft Corporation. Naming files, paths, and namespaces; 2016b. Available from: https://msdn.microsoft.com/en-us/library/windows/desktop/aa365247(v=vs.85).aspx. .

Microsoft Corporation. Driver signing. Software documentation; 2017. Available from: https://docs.microsoft.com/en-us/windows-hardware/drivers/install/driver-signing. [Accessed 4 December 2017].

Parker DB. Toward a new framework for information security? John Wiley & Sons, Inc.; 2012. p. 3.1–23. https://doi.org/10.1002/9781118851678.ch3.

Payne B, Maresca S, Lengye TK, Saba A. LibVMI. GitHub Repository; 2016. Available from: github.com/libvmi/libvmi. .

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: machine learning in python. J Mach Learn Res 2011;12:2825–30. http://dl.acm.org/citation.cfm?id=1953048.2078195.

Pennington AG, Griffin JL, Bucy JS, Strunk JD, Ganger GR. Storage-based intrusion detection. ACM Trans Inf Syst Secur 2010;13(4):Article 30, 27 pages. https://doi.org/10.1145/1880022.1880024.

Perloroth N. In cyberattack on Saudi firm, U.S. sees Iran firing back. New York Times; 2012. Available from: http://www.nytimes.com/2012/10/24/business/global/cyberattack-on-saudi-oil-firm-disquiets-us.html. .

Piper E. Cyberattack hits 200,000 in at least 150 countries – Europol. Reuters; 2017. Available from: http://www.reuters.com/article/us-cyber-attack-europol-idUSKCN18A0FX. .

Raiu C, Hasbini MA, Belov S, Mineev S. From Shamoon to Stonedrill – Wipers attacking Saudi organizations and beyond. Technical Report. Kaspersky Lab; 2017. Available from: https://securelist.com/files/2017/03/Report_Shamoon_StoneDrill_final.pdf. [Accessed 6 May 2016].

Rukhin A, Soto J, Nechvatal J, Miles S, Barker E, Leigh S, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. National Institute of Standards and Technology; 2010. p. 800.

Russinovich M. SDelete. Version 2.0; 2016. Available from: https://technet.microsoft.com/en-us/sysinternals/sdelete.aspx. [Accessed 27 November 2016].

Sammes T, Jenkinson B. Forensic computing: a practitioner's guide. London (UK): Springer-Verlag; 2000.

Scaife N, Carter H, Traynor P, Butler KRB. CryptoLock (and drop it): stopping ransomware attacks on user data. In: 2016 IEEE

36th International Conference on Distributed Computing Systems (ICDCS). 2016. p. 303–12. https://doi.org/doi.ieeecomputersociety.org/10.1109/ICDCS.2016.46.
Schneier B. Applied cryptography: protocols, algorithms, and source code in C. 2nd ed. New York: John Wiley & Sons, Inc.; 1996.
Sinitsyn F. TeslaCrypt 2.0 disguised as CryptoWall. Technical Report. Kaspersky Lab; 2015. Available from: https://securelist.com/teslacrypt-2-0-disguised-as-cryptowall/71371/. [Accessed 6 June 2016].
Solomon A. A brief history of PC viruses. Comput Fraud Secur Bullet 1993;12:9–19. https://doi.org/10.1016/0142-0496(93)90263-V.
Strunk JD, Goodson GR, Scheinholtz ML, Soules CAN, Ganger GR. Self-securing storage: protecting data in compromised system. In: Proceedings of the 4th conference on symposium on Operating System Design & Implementation – (OSDI'00), vol. 4. Berkeley (CA): USENIX Association; 2000. p. Article 12. Available from: http://dl.acm.org/citation.cfm?id=1251229.1251241.
Suiche M. Petya.2017 is a wiper not a ransomware. Technical Report. Comae Technologies; 2017. [Accessed 17 September 2017].
Sun R, Porter DE, Oliveira D, Bishop M. The case for less predictable operating system behavior. In: 15th workshop on Hot Topics in Operating Systems (HotOS XV). HotOS; 2015. Available from: https://www.usenix.org/conference/hotos15/workshop-program/presentation/sun. [Accessed 27 November 2016].
Tarakanov D. Shamoon the wiper: further details (part II). Technical Report. Kaspersky Lab; 2012. Available from: https://securelist.com/shamoon-the-wiper-further-details-part-ii/57784/. [Accessed 6 May 2017].
Trant G, Low J, van Lith D. Eraser Appendix A: erasure methods; 2016. Available from: http://eraser.heidi.ie/appendix-a-erasure-methods/. [Accessed 27 November 2016].
Verizon Wireless. 2017 data breach investigations report. 10th ed. 2017.
Ziem A. BleachBit – clean your system and free disk space; 2016. Available from: https://www.bleachbit.org/. [Accessed 6 June 2016].