IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 18 April 2023, accepted 17 May 2023, date of publication 24 May 2023, date of current version 1 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3279819

RESEARCH ARTICLE

Detection of Ransomware Attacks Using Processor and Disk Usage Data

KUMAR THUMMAPUDI, PALDEN LAMA, AND RAJENDRA V. BOPPANA, (Senior Member, IEEE)
Department of Computer Science, The University of Texas at San Antonio, San Antonio, TX 78249, USA
Corresponding author: Rajendra V. Boppana (rajendra.boppana@utsa.edu)

This research was partially supported by funding from the U.S. National Security Agency under Contracts H98230-21-1-0171 and #FA8075-18-D-0008/FA807520F0060 (as a subcontract D9104-S4/SCI-10607 through the Georgia Tech Research Institute).

ABSTRACT Ransomware often evades antivirus tools, encrypts files, and renders the target computer and its data unusable. The current approaches to detect such ransomware include monitoring processes, system calls, and file activities on the target system and analyzing the data collected. Monitoring multiple processes has a very high overhead; newer ransomware may interfere with the monitoring and corrupt the collected data. This paper presents a robust and practical approach to detecting ransomware in execution on a virtual machine (VM). We collect data for selected processor and disk I/O events for the entire VM from the host machine and use a machine learning (ML) classifier to develop a detection model. This approach avoids the overhead of continuously monitoring every process on the target machine and prevents the risk of data contamination by ransomware. Furthermore, it is resilient to variations in user workloads. It provides fast detection with a high probability for known (used for training the ML model) and unknown (not used for training) ransomware. The random forest (RF) classifier performed the best of the seven ML classifiers we tested. Over six different user loads and 22 ransomware, the RF model detected ransomware within 400 milliseconds with a 0.98 probability.

INDEX TERMS Deep learning, disk statistics, hardware performance counters, machine learning, ransomware, virtual machines.

I. INTRODUCTION

Ransomware is malware that encrypts files on a target computer or locks the computer to render the target machine and its data unusable. Cyber attackers use ransomware attacks to extort victims' money. Nation-state actors may use ransomware attacks to inflict harm on the critical infrastructure of their adversaries. These attacks are often combined with the exfiltration of victims' data to compel the victim to pay a ransom or sell the data on the dark web. In 2022 around 70% of businesses were victimized by ransomware attacks [1]. Ransomware is expected to attack a business, consumer, or device every 2 seconds by 2031, up from every 11 seconds in 2021. The damage cost was $20 billion in 2021 and is likely to exceed $265 Billion By 2031 [2]. Several researchers have recently investigated the detection of ransomware attacks.

Signature-based detection [3], [4] relies on the hash values generated by antivirus software for known ransomware and checks the target machine for files that match the hash values. However, polymorphic and metamorphic versions of previously known ransomware can bypass such signature-based detection [4], [5]. Therefore, behavioral or runtime detection of ransomware in execution complements signature-based detection methods. The behavioral analysis is a dynamic analysis that looks into the virus's behavior—the sequence of actions performed by the ransomware after it infects the victim machine. While malware activities vary widely, ransomware must perform a specific sequence of activities to encrypt as many data files as possible quickly. Some recent ransomware such as LockBit2.0, Darkside, and BlackMatter encrypt only parts (the first few bytes) of files to render

The associate editor coordinating the review of this manuscript and approving it for publication was Vicente Alarcon-Aquino.

more files unusable quickly [6]. Therefore, the requirement of quickly encrypting user files is likely to differentiate ransomware's runtime behavior from that of a benign application. The premise is that a system under ransomware attack must exhibit some form of immutable anomalous behavior. For example, the ransomware must access files from the hard disk and use the processor for data encryption resulting in elevated activity; suitably trained machine-learning methods may detect the elevated activity.

Runtime detection implemented on the target machine requires continual monitoring of various processes or components and subsystems, collecting data related to various events, and analyzing the data for anomalous behavior [7], [8]. Ransomware may try camouflaging its runtime behavior by creating additional processes and activities. However, the fact remains that a system under attack exhibits some form of elevated activity, which is detectable with appropriate analysis. Runtime detection is resource-intensive and intrusive if the monitoring is done on the target machine since the process corresponding to ransomware may not be easy to identify, and multiple processes must be monitored. Also, such monitoring is prone to be disabled by the ransomware designed to shut down active processes before encrypting files.

Hardware performance counters (HPCs) are special-purpose registers that count processor and system events per process or the entire system. The current processors can count hundreds of processor and system events, such as the number of instructions executed, cache misses, and accesses to off-chip memory. The data collected using HPCs are frequently used for performance analysis and tuning of the system software. However, several recent research efforts have investigated their use for malware detection [9], [10], [11], [12], [13], [14]. Alam et al. [15] utilized HPC data collected for each process running on the system. However, monitoring many processes is impractical as it can cripple the system's performance. Pundir et al. [7] collected the data at the machine level. However, their experiments are limited to a single workload of a Windows virtual machine (VM), and a change in the workload of a VM (varying the number of applications) may significantly impact the detection accuracy.

This paper investigates the fast detection of ransomware in execution on a Windows 10 virtual machine (VM). We collect both the HPC and disk I/O data at the host-machine level. The target (VM) is ignorant of the monitoring and data collection; also, there is little or no impact on its performance. We use machine learning (ML)-based models to analyze the data and detect ransomware in execution. Our method is particularly suited for protecting users of VMs in a cloud environment. The key contributions of this paper are as follows.

- We present an approach to detect ransomware accurately using HPC and disk I/O data observed from the host machine. Our approach avoids the overhead of monitoring many processes on the target machine and prevents data contamination by the ransomware designed to thwart such monitoring activities.
- We evaluate the detection effectiveness under different user workloads. To the best of our knowledge, we are the first to analyze the impact of workload variation on the runtime detection of active ransomware. Furthermore, we demonstrate that machine learning (ML) and deep learning (DL) based detection models improve detection accuracy when trained with varying workloads.
- We demonstrate that combining the HPC and disk I/O data to build an ML-based detection model can improve the detection accuracy compared to using a model based on HPC or I/O data alone.
- Our ML models can detect ransomware within a few hundred milliseconds of the start of ransomware execution with high probability, even with varying user workloads.
- We make the experimental datasets, codes, and scripts used in this study freely available to all researchers for further investigations and verification of our results [16], [17], [18], [19], [20].

The rest of the paper is organized as follows: Section II reviews the literature on runtime ransomware detection. Section III describes the experimental setup and the tools used. Section IV describes the ML models used. Section V presents the results of the detection of ransomware activity. Section VI concludes the paper.

II. BACKGROUND AND PRIOR WORK

This section focuses on the current work on HPC-based malware or ransomware detection mechanisms. We also briefly discussed other detection techniques and their limitations. We summarize the most relevant works, data collection methodologies, ML classifiers, and performance analyses in Table 1.

A. FILE AND I/O BASED MALWARE DETECTION

Many approaches have been proposed to detect crypto-ransomware activity using file I/O operations [22], [23], [24], [25], [26]. For example, Kharraz et al. [27], [28] presented UNVEIL, a dynamic analysis system, to detect ransomware activity by observing the filesystem I/O activity such as writing and delete of various benign and malicious applications. Despite a high 0.96 TPR (true positive rate, also called recall) with zero false positives, their experiments are designed to identify ransomware samples that are not in execution. Similarly, Continella et al. [29] proposed ShieldFS, an add-on driver, to make Windows native filesystem immune to ransomware attacks. They collect benign I/O access patterns on real-world machines and malicious access patterns by running ransomware in a controlled environment. A custom ML classifier is trained on the filesystem activity of collected samples and is leveraged at runtime to detect malicious activity. Likewise, Shukla et al. [30] tried to expose the gaps in existing works and proposed a customized FileTracker which learns new behavior while under attack by ransomware.

TABLE 1. Comparison of the most relevant results on ransomware detection. Random Forest (RF), Decision Trees (DT), Gradient Boost (GB), eXtreme Gradient Boost (XGB), Long Short-Term Memory (LSTM). Benignware (BW), Ransomware (RW).

	Alam et al. [15]	Pundir et al. [7]	Aurangzeb et al. [14]	Rhode et al. [21]	**Our Work**
Data Collection Tools	perf	likwid	perf	psutil	perf, virsh
Features Considered	5 HPC Events: instructions, branches, branch-misses, cache-misses, cache-references	16 groups of HPC events. Eg., TLB_DATA, L3_CACHE, ICACHE, etc.	11 HPC Events: Cache misses, TaskClock, Branches, SecondsTimeElapsed, etc.	26 process-level, 9 global metrics: CPU use, Memory use, I/O operation bytes on disk, etc.	5 HPC, 8 I/O Events
ML Classifiers	LSTM with 2 autoencoders	LSTM with 4 different optimizers	RF, DT, GB, XGB	RF, DT, etc.	RF, DT, XGB, LSTM etc.
OS	Sandboxed Linux	Linux hosted Windows VM	Linux hosted Windows VM	Sandboxed Windows 7	Linux hosted Windows 10 VM
Observation Level	Not indicated	Machine	Not indicated	Process & machine	Machine
Default system load levels	Not indicated	Single Load	Not indicated	Varied by launching upto 36 apps (95 processes)	6 different loads
Training Data	Not indicated	76 BW, and 80 RW	80 non-RW malware, and 80 RW	3604 BW, 22 RW, and 2792 malware	1 dryrun, 3 BW, 22 RW trained over 6 loads, for 7 rounds.
Testing Data	4 RW samples	70:30, 80:20, and 90:10 splits tested	80:20 split tested	Train test split is 50:50, and 10% of train set is held out for validation	5 rounds of 4 BW, and 4 RW trained. 2 rounds of 4 BW, and 2 rounds of all 22 RW tested.
Repetition / Cross-Validation (CV)	Not indicated	50 iterations	10-fold CV	Not indicated	21-fold CV
Sampling Interval	10 ms for 100 timestamps (1 sec)	100 µs for 20 timestamps (2 ms)	1 reading at the end of each application (atmost 240 sec)	1 reading at the end of 30 sec for RW experiments	5ms for 12000 times (60 sec)
Detection Accuracy / Time	1-10 sec	About 0.97 averaged over 50 iterations, with TLB_DATA using Adadelta optimizer	0.94 accuracy with RF classifier	0.88 TPR with 99.9% files unchanged with DT classifier	Accuracy is 0.92 with known and unknown ransomware and six different workloads using the RF classifier. 95% detection rate within 200 ms and 98% detection rate within 400 ms.

On the other hand, Scaife et al. [31] used file type changes, similarity measure, and Shannon entropy as primary indicators and presented, Crypto-Drop, an early-warning detection system. However, their model fails to differentiate between benign and malicious encryption. Similarly, Mehnaz et al. [8] introduced a hybrid ransomware detection mechanism, RWGuard, that uses entropy for file change monitoring and decoy techniques. Meanwhile, Sgandurra et al. [32] proposed a framework called EldeRan to identify dynamic features of ransomware using the ML classifierRegularized Logistic Regression. EldeRan observes Windows API calls, Registry Key operations, File I/O operations, and strings in binaries as features. Similarly, Zavarsky et al. [33] demonstrated that ransomware detection is possible on Windows and Android devices by monitoring abnormal filesystem and registry activities.

Most of the above detection mechanisms intercept filesystem I/O requests that pass through the I/O scheduler, analyze various features like decoy file access, access patterns, entropy, and file types, and suspend or kill the process based on the engine/model decision. Such a detection mechanism could lead to a single-point failure where the attacker can manipulate the detection engine as it also resides inside the same OS (operating system) under attack. Besides that, an entropy-based identification may fail to identify the difference between benign and malicious encryption. Furthermore, an entropy-based identification could lead to false alarms due to high entropy resulting from a benign activity such as file compression [34]. Similarly, Genc et al. [35] explored the limitations of existing decoy-based ransomware mitigation strategies by implementing a proof-of-concept anti-decoy ransomware and showed that it could bypass an existing

decoy-based technique. They also proposed possible solutions to mitigate anti-decoy ransomware.

B. OTHER MALWARE DETECTION

Besides filesystem-based detection, a few works explored the possibility of identifying ransomware based on its short-lived activities [36], [37], [38], [39]. Ahmadian et al. [40] proposed an approach called Connection-Monitor & Connection-Breaker which observes DNS requests assuming that most ransomware Command & Control center domains are generated by Domain Generation Algorithms (DGAs). Kolodenker et al. [41] proposed a prototype called PayBreak which holds the symmetric keys as escrow to thwart unauthorized encryption. The prototype relies on the insight that attackers heavily rely on Crypto-API provided by the OS under attack. Recent ransomware developers include a copy of the crypto-library in their code to bypass this protection technique. Kiraz et al. [42], on the other hand, proposed an approach called ExpMonitor which builds on the concept that most crypto-ransomware use public-key cryptographic algorithms which involve large integer arithmetic.

C. HPC-BASED MALWARE DETECTION

Zhou et al. [13] studied the usability of HPCs for malware detection. They conducted an extensive study of 2000 programs, including malware and benignware applications. They argued that there is no causation between low-level microarchitectural events and high-level software behavior. Das et al. [11] extensively studied the state-of-the-art work on using HPCs for security. They proved the high fidelity of HPCs at the process level and how ransomware can manipulate them.

On the contrary, Malone et al. [34] generated signatures of benign applications using the HPC data. They claimed that HPC data could be used to identify the deviations in the behavior of a program from its normal behavior. Demme et al. [9] investigated the feasibility of malware detection using HPCs and ML classifiers. The authors claim that their models identify the malware with an accuracy of about 0.90. Their study was conducted on Android, ARM, and Intel Linux platforms, while most ransomware specifically targets Windows OS [43]. Similarly, Tang et al. [10] proposed unsupervised anomaly-based malware detection using HPCs. They used the ML classifiers to train on the expected behavior of Windows programs, and any deviation from it is considered malicious. The work is focused on malware other than ransomware. Kadiyala et al. [12] proposed an HPC-based fine-grained malware detection through a three-step process of extracting HPCs of the system calls, dimensionality reduction, and feeding the components to an ML classifier. Aurangzeb et al. [14] tried to identify the difference between ransomware and non-ransomware malware using HPCs data and ML classifiers such as RF, decision trees (DT), gradient boost (GBoost), and extreme gradient boost (XGBoost).

Alam et al. [15] used HPCs to detect ransomware activity and proposed a framework called RATAFIA with the help of a Long Short-Term Memory (LSTM) based autoencoder and Fast Fourier Transformation. The *perf tool* [44] was used to collect HPC events data such as *branches, branch-misses, cache-misses, and cache-references*. They used a window-based technique to compare the behavioral patterns of ransomware and normal systems. While this work is conducted in a sandbox environment, their framework is for Unix operating systems. Pundir et al. [7] proposed a quicker detection technique, RanStop. They used the *likwid tool* [45] to capture HPC event data at an interval of 100 μs for 20 timestamps. They analyzed 80 crypto-ransomware using Recursive Neural Networks (RNNs), LSTM, and Global Average Pooling (GAP) for noise reduction. The authors claim to detect ransomware activity in less than 2 ms with an accuracy of 0.97. However, the experiments are conducted on a VM with no user activity. Therefore, any change in the user activity would affect the results. The other possible limitation of the technique is the duration of the data collection, 2 ms. Most recent ransomware acts stealthy and may not exhibit malicious behavior in such a short period.

III. EXPERIMENTAL SETUP

This section describes the experimental setup, the applications and ransomware used, and the data collection process.

A. TESTBED AND DATA COLLECTION

We used a bare-metal instance with dual Intel(R) Xeon(R) Gold 6240R CPUs (central processing units) at 2.40GHz clock rate on the Chameleon cloud [46], with Ubuntu 20.04.3 LTS as the host operating system (OS). We set up a Windows 10 VM on top of a Kernel-based Virtual Machine (KVM) hypervisor. The VM is the target of ransomware attacks. We consider the Windows 10 OS for the target owing to its popularity with ransomware developers. Eight virtual CPUs (vCPUs) out of the available ninety-six are pinned to the VM to ensure consistent allocation of resources to the VM.

To capture the variations in HPC and I/O data due to varying user activities, we configure the target VM with different user workloads ranging from an idle machine to a heavily loaded machine running tens of user applications. The three primary workloads are (1) Base-Load (BL.0): no user activities, only the default Windows system activities; (2) CPU-intensive Load (CL.0): about 10 Windows applications, such as MS Office applications, document reader, image viewer, movie player, browser windows loaded with Alexa top-10 websites [47], running on top of the base load; (3) Network-intensive Load (NL.0): more than 30 Windows applications installed from Ninite [48], such as OneDrive, GoogleDrive, and Microsoft Teams. Three additional secondary workload variants (CL.1, CL.2, and NL.1) are generated by varying the number of applications specified in one of the primary workload variants. We created six different snapshots of the VM, each with a specific level of user workload. The user

workloads can have a significant impact on the data collected (Section IV-A) and on the detection accuracy (Section V-D).

From the host machine, We collect HPC values for processor, memory, and system events of the process that corresponds to the VM using the *perf* tool [44] and disk I/O stats of the virtual disk allocated to the VM using the *virsh* tool [49] with the *domblkstats* option. Sampling the events at the host level ensures that ransomware running on the VM cannot directly interfere with the data collection process. Since we collect HPC and disk I/O data at the host machine level and treat the guest OS as a black box, the choice of the OS used for the guest machine will not significantly impact the experimental setup or the data collection methodology.

We collected over 100 ransomware samples downloaded from Virusshare [50]. We selected twenty-two ransomware samples that encrypted one or more files on the infected machine within the first 60 seconds of their execution. Table 2 gives the selected ransomware with their hashes and family names. For easier reference, we indicate the ransomware by the four-character strings at the start of their filenames.

Our selection includes ransomware samples that are active and able to encrypt. Some ransomware samples in our list start encrypting/modifying files immediately. The other ransomware samples stay quiet or scope the system and become active only towards the end of the 1-minute period we used for experiments. Figure 1 shows the percentage of files each ransomware encrypted. Out of the 22 ransomware samples, 12 encrypted at least 50% of some 9,500 user files within the first minute, six encrypted 10-40% of the files, while the remaining four encrypted less than 5% of the files. Therefore, our selection included a wide range of ransomware encryption behavior.

We also selected three benign applications–7zip, aesCrypt, and sDelete, which use operations such as compressing data, encrypting data, and deleting files likely to be used by ransomware and exhibit ransomware-like behavior. We selected these three benign applications to help the detection models differentiate malicious and benign activities that are similar more accurately. Our default benign case is with VM running none of the benign or ransomware codes; we call it a dryRun. Therefore, we ran 26 applications (22 ransomware and four benign cases) on each of the six snapshots.

Each run starts with a clean copy of one of the six snapshots and the execution of a benign or ransomware application. The benign applications are pre-installed on the snapshot and are initiated to perform a task that lasts at least 60 seconds. Ransomware applications are copied to the VM before their execution. Once a benign application or ransomware starts, we collect, on the host machine, the HPC and I/O data of the process that corresponds to the VM for approximately 60 seconds. Figure 2 depicts the workflow for data collection.

The sampling interval determines the detection speed. So a short sampling interval is desired, provided the data can be collected without loss or errors. The *perf* tool allows sampling intervals as short as 1 ms. We tried multiple sampling intervals, including 1 ms, 5 ms, 10 ms, and 100 ms for HPC events. The 5 ms interval provided a tradeoff between latency and consistency of data collection. For the disk I/O stats, the minimum sampling interval feasible with domblkstats is 20 ms; the data collected was consistent at this interval. So we used that as the sampling interval for disk I/O data collection. Therefore, in all our experiments, the HPC events are sampled in 5 ms intervals for a total of 12,000 times using the *perf* tool, and the disk I/O events are sampled in 20 ms intervals for a total of 3000 times using *virsh domblkstats*. A data trace consists of the HPC and disk I/O samples collected for an application run.

B. FEATURE SELECTION

A data instance is the primary data unit used for training or testing. An HPC data instance consists of forty consecutive HPC samples, and an I/O data instance ten consecutive disk I/O samples. Each trace has 300 nonoverlapping HPC and I/O data instances. We tried various HPC instance sizes with the number of samples per instance in the range [10, 1000]. We determined that 200 ms instances with 40 samples provide a good tradeoff between ransomware detection accuracy and latency. Since HPC data provided better prediction results than disk I/O data, we also used 200 ms for disk instances.

For the LSTM classifier, we used the raw HPC and I/O data contributing to each instance as inputs to the models. For all other classifiers, we computed statistics from the raw data for each instance and used them as inputs to the models. We computed four statistical features–the mean, median, standard deviation, and inter-quartile range (IQR) for each instance. The mean and standard deviation are likely sufficient; however, we included the median and IQR to handle any possible skewed data distributions resulting from ransomware selectively active for very short durations. (We tested various combinations of statistical features to train and test the HPC model. The model trained using the four statistical features of 5 HPC events outperformed the model trained with other combinations of events. The same is the case with the Disk-I/O model as well.)

The current Intel processors have three fixed counters that count *instructions executed, reference cycles, and clock cycles* and four programmable counters, which can be configured to count hundreds of processor events [51]. However, attempting to count more than four events at a time using the programmable counters leads to time-multiplexing of the events, which seems to result in inconsistent readings. Therefore, our experiments limited the number of events counted using the programmable counters to four. We used the following feature selection process to determine the specific four HPC events to count using the programmable counters. We started with a list of 40 HPC events that we identified as the most likely events impacted by ransomware activity. Table 3 lists these events. We ran each application ten times to collect the data for these 40 HPC events with four programmable events specified per run. To capture the variations in experimental conditions and ransomware behavior,

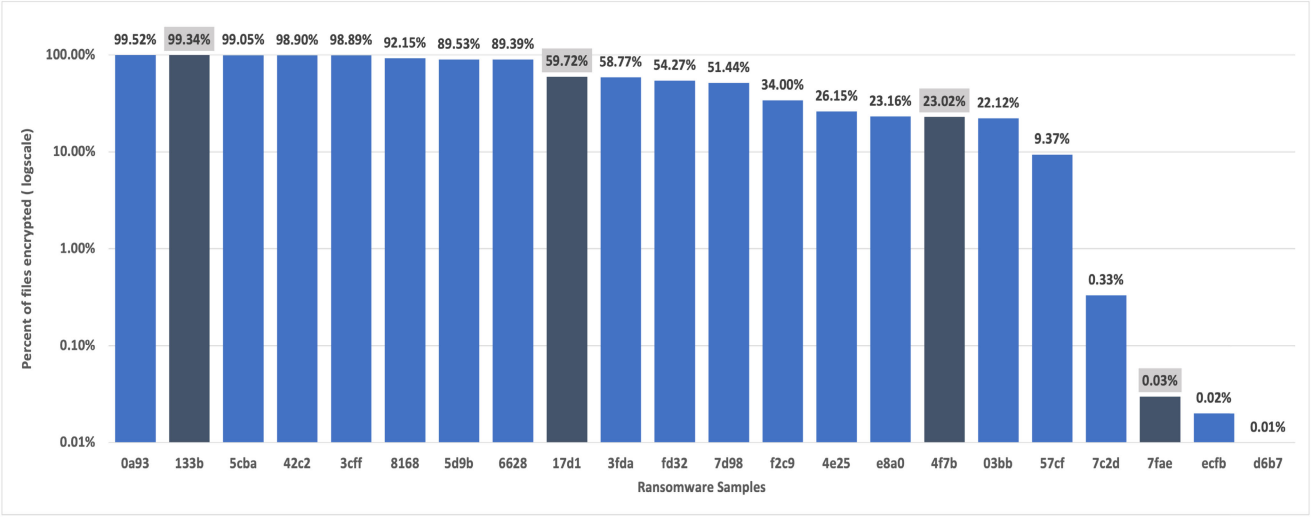


FIGURE 1. Percent of files encrypted by each ransomware sample by the end of first 60 seconds. Each value represent the average of 5 rounds of 6 different loads. The ransomware samples included in training are highlighted.

TABLE 2. List of ransomware hashes and families.

File Name	Family (MalwareBytes)	Family (Microsoft)
03bb4d5c0179fdceacc5df7645e6e7fa93e931ac9beb1968c7bbe40ba3499194	Ransom.Phobos	Ransom:Win32/Phobos.SW!MSR
0a937d4fe8aa6cb947b95841c490d73e452a3cafcd92645afc353006786aba76	Ransom.LockBit	Ransom:Win32/LockBit.PA!MTB
133bf8be0cf7003b83b03579970997d408a930e58ec2726715140520900c06de	Ransom.Sodinokibi	Ransom:Win32/Revil.SI!MTB
17d153a225ea04a229862875795eeec0adb8c3e2769ba0e05073baaf86850467	Ransom.Sodinokibi	Ransom:Win32/Revil.SI!MTB
3cff33197edc918d47d08f44d6ddbdda157337f0ad58288d15746cf72c0e4c57	Ransom.Sodinokibi	Ransom:Win32/Revil.SI!MTB
3fdad99a17a6766fe396081f82394f5e2da0142651427da64a5b6e28c9df2fd4	Ransom.Sodinokibi	Ransom:Win32/Sodinokibi.C
42c28feb23c992a350673d63413bf11bc816d00a079462ab524934219d46430d	Ransom.Sodinokibi	Ransom:Win32/Sodinokibi.DSB!MTB
4e2554b448424859b508433e6c4a043718343febc7894a849f446dd197b47d1e	Ransom.Maze	Ransom:Win32/Maze.PA!MTB
4f7bdda79e389d6660fca8e2a90a175307a7f615fa7673b10ee820d9300b5c60	Ransom.NetWalker	Ransom:Win32/Filecoder.DD!MTB
57cf4470348e3b5da0fa3152be84a81a5e2ce5d794976387be290f528fa419fd	Ransom.NetWalker	Ransom:Win32/NetWalker!MSR
5cba3e44271279e747a67dd312d4dca18832b5a850ea6b85a460846ef0101fb6	Ransom.Sodinokibi	Undetected
5d9b75e2cb84333c6b56604ce47af75b11f80bf9079054f6619251b68357d87c	Ransom.Sodinokibi	Trojan:Win32/Occamy.C
6628de7ffbbe168a4fa9ff0a1a29b54e88a32e5963db0dd1aea4b80102c8ce01	Ransom.Sodinokibi	Ransom:Win32/Revil.SI!MTB
7c2dbad516d18d2c1c21ecc5792bc232f7b34dadc1bc19e967190d79174131d1	Ransom.DeathRansom	Undetected
7d98972d5c78e1d4969da76856d6818942b606c267efa67fd31d39ae77497e9c	Ransom.Rapid	Ransom:Win32/Robinhood.AR!MTB
7faeb64c50cd15d036ca259a047d6c62ed491fff3729433fefba0b02c059d5ed	Ransom.Ryuk	Ransom:Win32/Sodinokibi.AD!MTB
81689f1be92c8fb7e94fe241441c7eb43cfb77c6d23592b0248566bd709ff2ed	---	Ransom:Win32/Revil.SI!MTB
d6b7b27e13700aaa7f108bf9e76473717a7a1665198e9aafcc2d2227ca11bba9	Ransom.Ryuk	Ransom:Win32/Ruyk.A!ibt
e8a091a84dd2ea7ee429135ff48e9f48f7787637ccb79f6c3eb42f34588bc684	Malware.AI.4262356839	Ransom:Win32/Maze
ecfb5c95d0f3d112650ef4047936e8fa5244c21c921f6c7a6963e92abab4949d	Ransom.FileCryptor	Ransom:MSIL/Kelnoc.A
f2c9ae3735430b930a81148c0bb470fcb733e456a2a942f859a1b59c4a7b2150	Malware.AI.2057448532	Ransom:Win32/Locky.A
fd32cec288cec4f16dc5430cf86dc17e1d4cf941d635979fc17a59c8d6d83d44	Ransom.Povlsomware	Ransom:MSIL/RnToad.SL!MTB

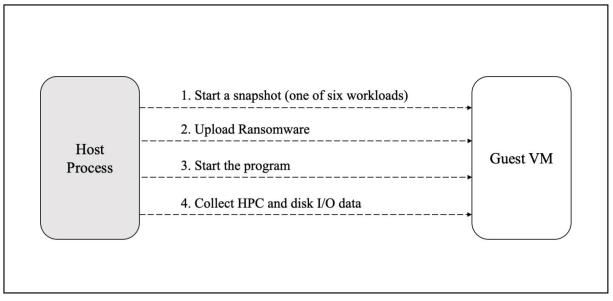


FIGURE 2. Data collection workflow.

we repeated the data collection for five rounds per user workload, per application; these traces are publicly available [16].

To reduce the HPC features from 40 programmable events to four, we applied Recursive Feature Elimination with Cross-Validation (RFECV) [52] to select the top twenty HPC events that impact detection prediction the most. We calculated pairwise Pearson Correlations [53] of these features and picked the top four with low correlations ($< |0.3|$) among them. Table 3 highlights the four selected HPC features. The four HPC events we selected are also identified as the most useful for ransomware detection in literature [7], [13], [15]. In addition to these four, we also selected the *instruction counts* provided by one of the fixed counters since it is helpful in prior studies [15]. Therefore, we collected and used the data from five HPC events–LLC-stores, L1-icache-load-misses, branch-load misses, node-load misses, and instructions–for the experiments in the remainder of the paper.

The domblkstats tool gives the data for all eight disk I/O events it monitors (Table 3), Since our analysis of the data [17] indicated that using all eight events as features gives

TABLE 3. HPC and I/O Events considered. Of the 41 HPC events considered in preliminary stage, the highlighted events are the ones with short-listed using feature-selection, and are considered in second stage. All 8 I/O events are considered in both the stages.

HPC Events			
branches	node-loads	cache-references	node-stores
branch-misses	fp_arith_inst_retired.scalar_single	LLC-store-misses	dTLB-load-misses
LLC-loads	fp_arith_inst_retired.scalar_double	L1-dcache-loads	iTLB-load-misses
LLC-stores	sw_prefetch_access.nta	L1-dcache-load-misses	L1-dcache-stores
branch-loads	sw_prefetch_access.t0	LLC-load-misses	machine_clears.count
dTLB-stores	sw_prefetch_access.t1_t2	L1-icache-load-misses	machine_clears.smc
cache-misses	sw_prefetch_access.prefetchw	branch-load-misses	node-store-misses
instructions	ld_blocks.store_forward	dTLB-store-misses	iTLB-loads
fp_assist.any	hw_interrupts.received	node-load-misses	ld_blocks.no_sr
dTLB-loads	arith.divider_active	power/energy-pkg/	other_assists.any
fp_arith_inst_retired.128b_packed_single			

I/O Events			
rd_req	rd_bytes	rd_total_times	wr_req
wr_bytes	wr_total_times	flush_operations	flush_total_times

better performance, we did not attempt to reduce the number of disk I/O features.

C. DATA COLLECTION FOR MODEL TRAINING AND TESTING

We ran additional experiments and collected data traces to train the ML/DL models and test their detection effectiveness. The data used for feature selection is not used for final model training and testing. To obtain a data trace, we ran one application (from 22 ransomware, three benign applications, and a dryRun) at a time on one of the six VM snapshots (representing different user workloads). We collected data for the selected five HPC events and the eight disk I/O events. For each scenario, we repeated the runs six more times for $26 \times 6 \times 7 = 1092$ traces. We published the data on Harvard Dataverse [18], [19].

IV. MACHINE LEARNING MODELS

We investigated five ML and two DL classifiers for their suitability for detecting ransomware based on the HPC and disk I/O data [52]. The ML classifiers are Random Forest (RF), Support Vector Machine (SVM), Decision Trees (DT), k-Nearest Neighbor (kNN), and eXtreme Gradient Boosting (XGBoost). The DL classifiers are Deep Neural Network (DNN) and Long Short-Term Memory (LSTM). We developed three models: an HPC model, which uses only HPC data; a Disk I/O model, which uses only disk I/O data; and an Integrated model, which uses both HPC and disk I/O data. For the integrated model, a data instance consists of an HPC instance and an I/O instance. We evaluated the three models using the seven ML/DL classifiers. The ML classifiers used the default hyper-parameters provided by scikit-learn [52], a representative machine learning library in Python. The DL classifiers required tuning hyperparameters appropriately. Due to the high sensitivity of the DNN classifier to hyper-parameters, it was automatically tuned with autokeras, a representative AutoML library [54]. The LSTM model consisted of four layers. We used padding to address the length mismatch between the I/O and HPC data samples in an instance and fed the raw data to the model. We used a masking layer to identify and remove the padded data while processing the data by the downstream layers. We used the LSTM layer with 64 memory units and a global average pooling layer to avoid overfitting. The final layer was a dense output layer with a single neuron and a sigmoid activation function. We used Python library modules scikit-learn v1.1.2 for SVM, kNN, DT, and RF, XGBoost v1.6.2 for XGBoost, autokeras v1.0.20 for DNN, and Keras v2.9.0 with TensorFlow backend for LSTM.

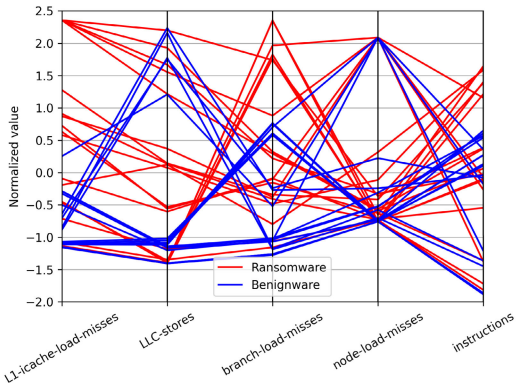
A. MULTIVARIATE ANALYSIS OF HPC AND I/O DATA IN THE PRESENCE OF USER WORKLOADS

We analyzed the multivariate relationship between HPC and IO data and the application type (ransomware and benign applications) for different user workloads. Figure 3 shows plots of the five HPC events for four ransomware, with file names beginning with 133b, 17d1, 4f7b, and 7fae, and the four benign applications–7zip, aesCrypt, sDelete, and dryRun. The four ransomware samples represent the Sodinokibi, Netwalker, and Ryuk families. The vertical axis represents the normalized means of the features in instances halfway through an application execution. Each polyline represents an application execution for a workload; the red lines denote ransomware, and the blue lines denote benign applications. (Although there is no interpolation of values between different events on the x-axis, we used lines to show the data from the same run.)

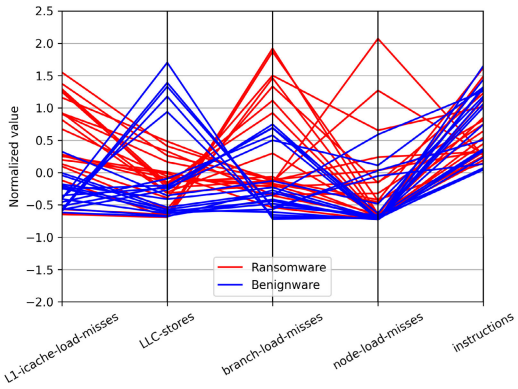
Figures 3 (a), (b), and (c) show that the HPC data vary significantly under three different workloads–basic, BL.0, compute-intensive, CL.0, and network-intensive, NL.0. Figures 4 (a), (b), and (c) show a similar impact of loads on I/O data. These observations imply that an ML/DL model trained with data collected under the base load only may not be reliable in distinguishing between ransomware and benign ware under different workload conditions (Section V-D). To our knowledge, the prior work on ransomware detection only considers the base-load case [7].

V. RESULTS

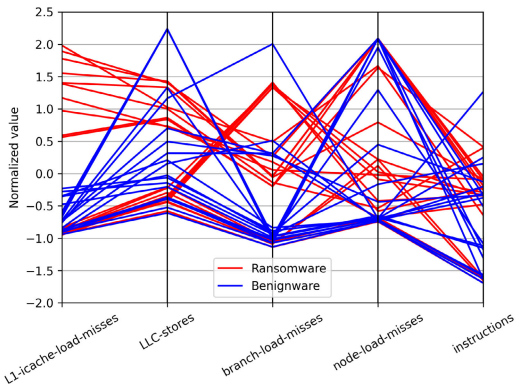
This section presents our analysis of the experimental data and the impact of workloads on detection effectiveness.



(a) BL.0

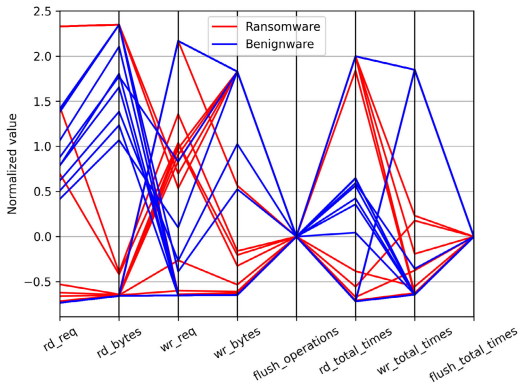


(b) CL.0

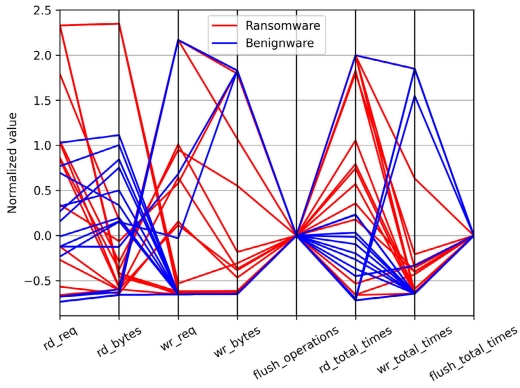


(c) NL.0

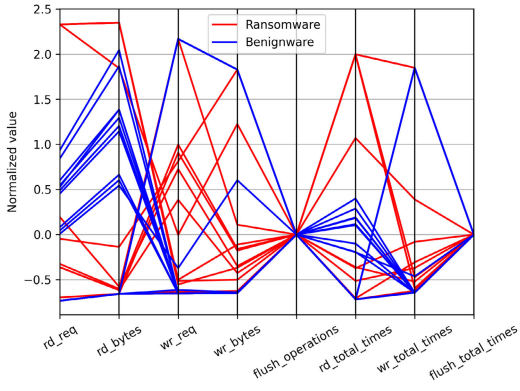
FIGURE 3. Impact of load variation on the multivariate relationship between HPC data and the application type (ransomware vs. benign).



(a) BL.0



(b) CL.0



(c) NL.0

FIGURE 4. Impact of load variation on the multivariate relationship between I/O data and the application type (ransomware vs. benign).

A. EVALUATION METHODOLOGY

We ran the four benign applications and the 22 ransomware, one at a time, on six VM snapshots, each with a different user workload. We repeated these runs six more times for a total of $26 \times 6 \times 7 = 1092$ runs. For ease of reference, we denote the data collected in a run as a trace. Each trace provides 300 instances of HPC data and 300 instances of disk I/O data. A data round consists of the data traces for all benign and ransomware applications for the six workloads; so there are $26 \times 6 = 156$ data traces in a round. There are seven rounds of data.

The ML and DL models are trained with data from five rounds for four ransomware (file names beginning with 133b, 17d1, 4f7b, and 7fae) belonging to three families (Sodinokibi, Netwalker, and Ryuk) and all four benign applications. As explained in Section III-A, the 22 ransomware samples we selected start encrypting files at different points during the execution: early in the experiment to nearly at the end of the experiment. The 4 ransomware samples we chose for training also reflect this pattern: 133b encrypts nearly all user files (about 9,500) within the experiment's duration of one minute; 17d1 encrypts about 60% of the files; 4f7b encrypts about 23% of the files; and 7fae encrypts less than

1% of the files. We selected only four ransomware samples for training because we are interested in developing ML/DL models that are robust enough to detect previously unknown ransomware. Since ransomware codes evolve over time, the detection model should be robust enough to detect new ransomware when they actively encrypt files. Also, using only four ransomware samples for training ensures that the amount of benign and malicious data used is balanced. The dryRun represents the default benign scenario with user activities; the specific three benign applications we used mimic ransomware-like activity and thus provide challenging test scenarios.

For testing, we used the remaining two rounds of data for all 22 ransomware and four benign applications. This allows us to evaluate the models for known ransomware (there are four in our test data) and unknown ransomware (there are 18 that were not used for training). For cross-validation purposes, we repeat this evaluation with different combinations of training rounds. Since there are $\binom{7}{5} = 21$ possible ways to select the training rounds, this serves as a 21-fold cross-validation.

B. EVALUATION METRICS

A ransomware instance detected by the model as an attack is a true positive (TP), and an undetected ransomware instance is a false negative (FN). Similarly, a benign application instance detected as benign activity is a true negative (TN), and a benign instance undetected as benign is a false positive (FP). We use balanced accuracy, F1-score, precision, recall, FPR (false positive rate), and FNR (false negative rate), defined below, as the metrics. [55].

$$BalancedAccuracy = \frac{TPR + TNR}{2} \tag{1}$$

$$F1score = \frac{2TP}{2TP + FP + FN} \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$FPR = \frac{FP}{FP + TN} \tag{5}$$

$$FNR = \frac{FN}{FN + TP} \tag{6}$$

C. ANALYSIS USING THE ML MODELS

Table 4 displays the results of the HPC, disk I/O, and the integrated HPC-I/O models using each of the seven classifiers. Each value in the FPR column indicates the average of $2 \times 4 \times 6 \times 21 \times 300 = 302,400$ benign instances tested. Each value in the FNR column indicates the average of $2 \times 22 \times 6 \times 21 \times 300 = 1,663,200$ malicious instances tested. The table shows that the Integrated model outperformed both individual models for each classifier. Therefore, we will use only the integrated models for the remainder of the paper.

Of the seven classifiers, the RF classifier achieves the highest balanced accuracy of 0.92. This result is consistent with the observations by other researchers [8], [13], [14]. The XGBoost classifier is the nearest contender; however, its computational time complexity is three times that of RF. While XGBoost, DNN, and LSTM could potentially improve the accuracy, they require a significant amount of hyperparameter tuning and increase the training/testing time compared to RF and other models. The I/O models gave better FNRs but also had high FPRs indicating that they failed to differentiate benign applications' disk activity to that from ransomware. The poor performance of the I/O models could be due to the three benign applications mimicking file activities that ransomware exhibits. Therefore, training with only I/O data is not sufficient for accurate detection.

Our primary goal is to catch ransomware in action. We consider ransomware an active threat if it encrypts files; we want to detect that activity quickly. Therefore, we investigated an alternate way to measure the detection speed: we compute the number of instances taken (the time to detect) for the first positive prediction in a ransomware run. Figure 5 shows the times to detect ransomware for the RF classifier. There are a total of $2 \times 22 \times 6 \times 21 = 5544$ ransomware traces in the 21-fold cross-validation. For each trace, we compute the time to detect ransomware. In the Integrated model, ransomware that starts execution is detected in the first test instance (200 ms) 95% of the time. The model takes only 400 ms (two instances) or less to detect ransomware 98% of the time. While the I/O model gives better detection times, it has a high FPR compared to that of the integrated model, as indicated in Table 4 for the RF classifier.

For the best-performing RF-integrated model, the false positive rate is 0.03, and the false negative rate is 0.12. Based on the time-to-detect analysis, the model performs well in catching ransomware activity quickly. The false positive rate while low, could result in frequent false positives. However, the model could be tweaked to reduce false positives at the risk of higher false negatives by lowering the classification threshold from the default 0.5 to 0.4 or 0.3. The change in classification threshold from 0.5 to 0.3 leads to an improved FPR of 0.01 with FNR increased to 0.18. On the other hand, of the 5544 ransomware traces, the model with a classification threshold of 0.3 detected the ransomware in the first instance (200 ms) more than 90% of the time and more than 95% of the time in two instances (400 ms).

An effective way to address frequent false positives is to add a secondary verification that confirms a positive prediction. This could be a guest (target) based monitor [8]; if the monitor is running, its data could be analyzed to confirm a positive prediction from our model. If the monitor is not running or does not respond, then the system could be suspended to check more carefully. Also, disk writes could be throttled until the secondary verification is done to avoid additional files from being encrypted [56]. Using a guest-based monitor only could be problematic if the ransomware kills such

TABLE 4. The performance of HPC, I/O, and integrated models using seven classifiers with a classification threshold of 0.5. The values indicted are averages from 21-fold cross-validation. The best values for each metric are highlighted.

Model	Classifier	Accuracy	F1-score	Precision	Recall	FPR	FNR
all HPC	SVM	0.855	0.869	0.983	0.777	0.071	0.223
	kNN	0.869	0.884	0.986	0.800	0.061	0.200
	DT	0.853	0.880	0.980	0.799	0.095	0.201
	RF	0.891	0.906	0.988	0.834	0.054	0.166
	DNN	0.853	0.867	0.986	0.776	0.070	0.224
	xGB	0.891	0.902	0.990	0.830	0.049	0.171
	LSTM	0.882	0.896	0.989	0.823	0.055	0.177
all I/O	SVM	0.820	0.960	0.940	0.970	0.331	0.030
	kNN	0.829	0.853	0.979	0.770	0.112	0.231
	DT	0.819	0.927	0.950	0.904	0.266	0.096
	RF	0.831	0.940	0.950	0.926	0.261	0.074
	DNN	0.819	0.956	0.940	0.970	0.334	0.031
	xGB	0.831	0.940	0.950	0.928	0.267	0.072
	LSTM	0.838	0.916	0.961	0.880	0.205	0.120
Integrated HPC-I/O	SVM	0.884	0.891	0.993	0.810	0.034	0.191
	kNN	0.894	0.900	0.992	0.827	0.038	0.173
	DT	0.901	0.919	0.987	0.859	0.058	0.141
	RF	0.923	0.933	0.994	0.880	0.031	0.121
	DNN	0.891	0.896	0.995	0.815	0.034	0.185
	xGB	0.920	0.927	0.994	0.869	0.030	0.131
	LSTM	0.909	0.920	0.992	0.859	0.041	0.141

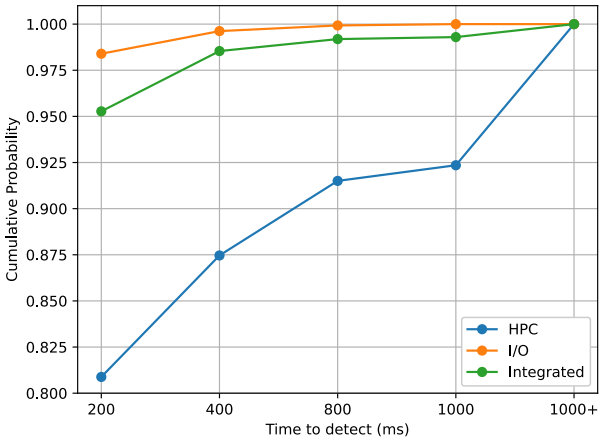


FIGURE 5. Ransomware detection probability over time.

processes when it starts. We believe our method can serve as an indicator of potential ransomware activity without being directly manipulated or stopped by the ransomware.

D. IMPACT OF USER WORKLOADS ON THE ROBUSTNESS OF ML MODELS

We investigated the robustness of ransomware detection by the RF-integrated model in the presence of various Windows user workloads. Tables 5 and 6 show the impact of workload variation on the accuracy of the model. The accuracy is higher when the model is trained and tested under the same workload (the cases corresponding to the diagonal entries); this is the case for most of the results reported in the literature. The results are even better if the tested ransomware is also used for training (Table 6). For example, Pundir et al. [7] trained and tested on an idle machine with a typical workload (equivalent to our BL.0 workload) with known ransomware and reported an accuracy of about 0.97 with 90% of the data used for training and 10% for testing. Our RF-integrated model trained and tested on the workload BL.0 shows an accuracy of 1.00 for known ransomware (Table 6) and 0.94 for test data with a

TABLE 5. The RF-integrated model's accuracy for various workloads. The model is trained with 5 rounds of the four benign applications' data and 4 ransomware data; it is tested with the remaining 2 rounds of benign and 22 ransomware, 18 of which are unknown, data. The values indicated are averages from 21-fold cross-validation.

Training Workload	Testing Workload						
	BL.0	CL.0	CL.1	CL.2	NL.0	NL.1	Avg
BL.0	0.937	0.730	0.828	0.790	0.940	0.824	0.842
CL.0	0.859	0.905	0.892	0.912	0.864	0.895	0.888
CL.1	0.820	0.885	0.923	0.899	0.823	0.844	0.866
CL.2	0.865	0.859	0.890	0.927	0.871	0.866	0.880
NL.0	0.927	0.752	0.823	0.784	0.944	0.828	0.843
NL.1	0.857	0.693	0.754	0.731	0.882	0.914	0.805
BL.0 CL.0 NL.0	0.930	0.908	0.880	0.891	0.941	0.840	0.898
All	0.932	0.915	0.930	0.921	0.940	0.902	0.923

TABLE 6. The RF-integrated model's accuracy for various workloads. The model is trained with 5 rounds of data for the four benign applications and four ransomware samples; it is tested with the remaining 2 rounds of data for the same benign applications and ransomware samples. The values indicated are averages from 21-fold cross-validation.

Training Workload	Testing Workload						
	BL.0	CL.0	CL.1	CL.2	NL.0	NL.1	Avg
BL.0	1.000	0.737	0.836	0.795	0.988	0.835	0.865
CL.0	0.894	0.938	0.906	0.939	0.894	0.929	0.927
CL.1	0.826	0.925	0.958	0.925	0.830	0.884	0.891
CL.2	0.912	0.879	0.902	0.952	0.912	0.878	0.906
NL.0	0.983	0.755	0.826	0.786	0.991	0.841	0.864
NL.1	0.883	0.696	0.739	0.735	0.905	0.946	0.817
BL.0 CL.0 NL.0	0.999	0.945	0.900	0.915	0.990	0.857	0.934
All	0.998	0.954	0.960	0.955	0.990	0.935	0.965

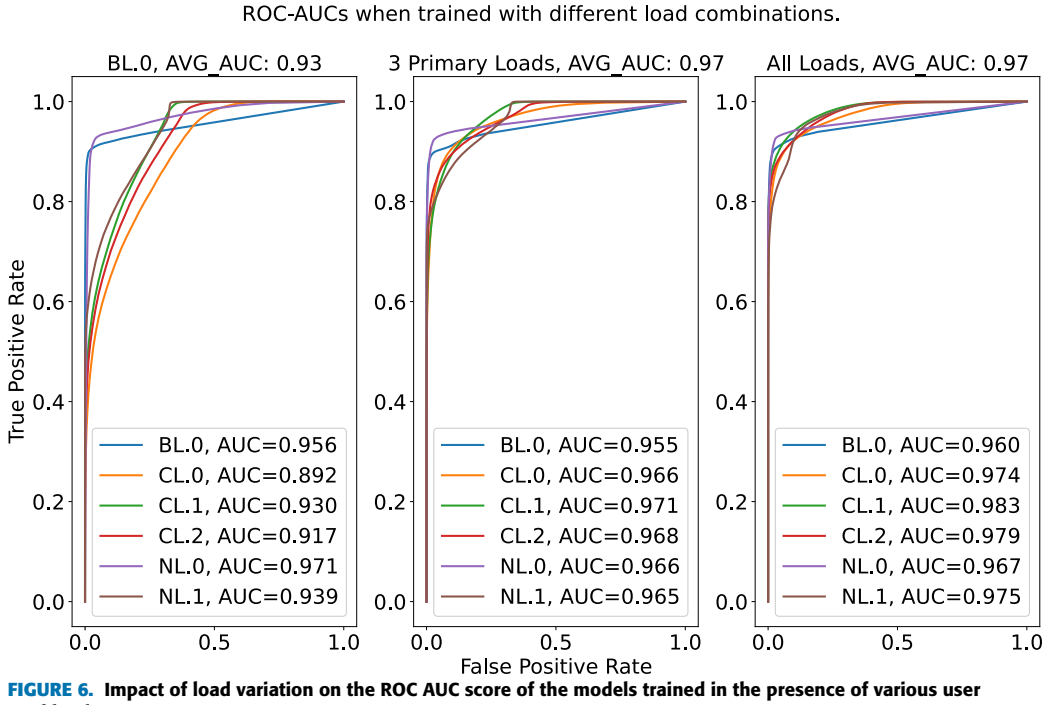


FIGURE 6. **Impact of load variation on the ROC AUC score of the models trained in the presence of various user workloads.**

combination of four known and 18 unknown ransomware (Table 5).

When the model is tested with previously unseen workloads, we see a significant drop in accuracy, indicated in the off-diagonal values in the first six rows of Tables 5 and 6. It is noteworthy that previously unseen user workload adversely impacts the model's performance more than unknown ransomware. On the other hand, as the model is trained with data from different workloads (the last two rows of Tables 5 and 6), the model's performance improves significantly. When comparing the variation from training with three loads to six loads, the performance improvement is less than 3%. From this, we can conclude that a model trained with a few variations of user workloads can consistently detect ransomware activity, even when the load at the time of detection deviates from the trained loads.

Figure 6 shows the Receiver Operator Characteristics (ROC) curves of the RF-integrated models trained and tested under various user workloads and test data of known and unknown ransomware activity. When the model is trained with only one of the primary loads, the average Area under the ROC Curve (AUC) ranges from 0.93 to 0.95. However, when trained with the three primary loads or all six loads, the average AUC went up to 0.97. This analysis shows that while training with one load may result in a model nonresilient to other workloads, training with even a few variations of workloads makes the model more resilient to previously unseen workload conditions.

VI. CONCLUSION AND FUTURE WORK

This paper presents an approach to detect ransomware executing on a VM quickly and accurately by collecting processor and disk I/O activity events for the VM from the host machine and using machine learning techniques to analyze the data. The processor-event data are collected using the *perf* tool and hardware performance counters (HPCs) for five events, selected from more than 40 events using recursive feature elimination with cross-validation; disk I/O-event data are collected for eight events using *virsh domblkstats*. We considered five ML and two DL classifiers. For each classifier, we developed three models: one uses HPC data only, the second uses disk I/O data only, and the third is an integrated model that uses both HPC and I/O data. The integrated model performed the best for all seven classifiers. The random forest (RF) classifier has the best detection accuracy among the seven classifiers, and its training times are lower than those of the other classifiers. Overall, the RF-integrated model shows promising results in detecting known ransomware (used for training) and unknown ransomware (not used in training).

Our approach has two advantages over the prior approaches that monitor ransomware activities directly on the target machine. First, the target machine does not experience any overhead since the monitoring is done from the host machine. Secondly, ransomware running on the target machine cannot interfere with or corrupt the data collection.

A significant contribution of this study is the evaluation of the ML/DL-based detection models under multiple user workloads on the target VM. Most prior studies evaluate their models for one workload scenario. Our analysis shows that a model trained using the data collected without any user activity as the background workload has low detection accuracy when there is user activity such as using productivity applications (for example, MS Word, Excel, or Adobe tools), Web browsing, or playing audio/video clips.

Our detection methodology applies to virtual machines monitored from the host or hypervisor level. One advantage of this approach is data collection process remains the same regardless of the operating system used for the VM. Another advantage is the ransomware running on the VM is unaware of the monitoring and cannot interfere with the data collection. Cloud service providers can use our detection methods to provide their users additional protection against ransomware attacks. Given the ongoing migration of computing to VMs and the cloud, this additional protection against ransomware is timely and critical.

In this study, we did not explicitly investigate the exfiltration of data by ransomware. We plan to investigate the detection of exfiltration activities by collecting the target VM's network traffic from the host machine and analyzing it with HPC and I/O data.

We label all data collected in ransomware runs as malicious. However, some ransomware scout the target machine and the network quietly for some time (tens of seconds or longer) before starting their malicious activities. The execution of such ransomware yields noisy data—ransomware is not active for large portions of the experiment's time, and the sampled data corresponds to the default system load but is labeled as malicious. Since we are interested in detecting ransomware that is actively modifying files, such data could lead to less accurate detection. Suppose we can find a way to label the data collected during the scouting period as scouting and the data collected during encryption activities as malicious. In that case, we believe the detection accuracy will improve and will be a better detection indicator of ransomware in its most destructive mode. We plan to investigate more accurate data labeling in our future work. Another direction for our future work is to use more levels of workloads to make the detection models more resilient to varying user activity.

In this paper, we presented models and tested them using the data collected from additional rounds of experiments. In the future, we plan to use the models for live ransomware detection while in execution.

While our model limits its applicability to VMs, we plan to adapt it to stand-alone machines in our future work. We have not evaluated whether the models developed for a machine configuration work well for another machine configuration, such as increased memory or more CPU cores. We plan to investigate this in the future.

ACKNOWLEDGMENT

The opinions expressed and any errors contained in the article are those of the authors. This article is not meant to represent the position or opinions of the author's institution or the funding agencies.

REFERENCES

[1] SR Department. (2022). *Ransomware victimization rate 2022*. Accessed: Apr. 6, 2022. [Online]. Available: https://www.statista.com/statistics/204457/businesses-ransomware-attack-rate/

[2] D. Braue. (2022). *Ransomware Damage Costs*. Accessed: Sep. 16, 2022. [Online]. Available: https://cybersecurityventures.com/global-ransomware-damage-costs-predicted-to-reach-250-billion-usd-by-2031/

[3] Logix Consulting. (2020). *What is Signature Based Malware Detection*. Accessed: Apr. 3, 2023. [Online]. Available: https://www.logixconsulting.com/2020/12/15/what-is-signature-based-malware-detection/

[4] W. Liu, P. Ren, K. Liu, and H.-X. Duan, "Behavior-based malware analysis and detection," in *Proc. 1st Int. Workshop Complex. Data Mining*, Sep. 2011, pp. 39–42.

[5] (2021). *Polymorphic Malware*. Accessed: Apr. 3, 2023. [Online]. Available: https://www.thesslstore.com/blog/polymorphic-malware-and-metamorphic-malware-what-you-need-to-know/

[6] M. Loman. (2021). *Lockfile Ransomware's Box of Tricks: Intermittent Encryption and Evasion*. Accessed: Nov. 16, 2021. [Online]. Available: https://news.sophos.com/en-us/2021/08/27/lockfile-ransomwares-box-of-tricks-intermittent-encryption-and-evasion/

[7] N. Pundir, M. Tehranipoor, and F. Rahman, "RanStop: A hardware-assisted runtime crypto-ransomware detection technique," 2020, *arXiv:2011.12248*.

[8] S. Mehnaz, A. Mudgerikar, and E. Bertino, "RWGuard: A real-time detection system against cryptographic ransomware," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2018, pp. 114–136.

[9] J. Demme, M. Maycock, J. Schmitz, A. Tang, A. Waksman, S. Sethumadhavan, and S. Stolfo, "On the feasibility of online malware detection with performance counters," *ACM SIGARCH Comput. Archit. News*, vol. 41, no. 3, pp. 559–570, Jun. 2013.

[10] A. Tang, S. Sethumadhavan, and S. J. Stolfo, "Unsupervised anomaly-based malware detection using hardware features," in *Proc. Int. Workshop Recent Adv. Intrusion Detection*. Cham, Switzerland: Springer, 2014, pp. 109–129.

[11] S. Das, J. Werner, M. Antonakakis, M. Polychronakis, and F. Monrose, "SoK: The challenges, pitfalls, and perils of using hardware performance counters for security," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 20–38.

[12] S. P. Kadiyala, P. Jadhav, S.-K. Lam, and T. Srikanthan, "Hardware performance counter-based fine-grained malware detection," *ACM Trans. Embedded Comput. Syst.*, vol. 19, no. 5, pp. 1–17, Sep. 2020.

[13] B. Zhou, A. Gupta, R. Jahanshahi, M. Egele, and A. Joshi, "Hardware performance counters can detect malware: Myth or fact?" in *Proc. Asia Conf. Comput. Commun. Secur.*, May 2018, pp. 457–468.

[14] S. Aurangzeb, R. N. B. Rais, M. Aleem, M. A. Islam, and M. A. Iqbal, "On the classification of microsoft-windows ransomware using hardware profile," *PeerJ Comput. Sci.*, vol. 7, p. e361, Feb. 2021.

[15] M. Alam, S. Bhattacharya, S. Dutta, S. Sinha, D. Mukhopadhyay, and A. Chattopadhyay, "RATAFIA: Ransomware analysis using time and frequency informed autoencoders," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2019, pp. 218–227.

[16] K. Thummapudi, R. Boppana, and P. Lama, "HPC 41 events 5 rounds," Harvard Dataverse, 2022, doi: 10.7910/DVN/MA5UPP.

[17] K. Thummapudi, R. Boppana, and P. Lama, "IO 41 events 5 rounds," Harvard Dataverse, 2022, doi: 10.7910/DVN/GHJFUT.

[18] K. Thummapudi, R. Boppana, and P. Lama, "HPC 5 events 7 rounds," Harvard Dataverse, 2022, doi: 10.7910/DVN/YAYW0J.

[19] K. Thummapudi, R. Boppana, and P. Lama, "Io 5 events 7 rounds," Harvard Dataverse, 2022, doi: 10.7910/DVN/R9FYPL.

[20] K. Thummapudi, R. Boppana, and P. Lama, "Scripts to reproduce results," Harvard Dataverse, 2023, doi: 10.7910/DVN/HSX6CS.

[21] M. Rhode, P. Burnap, and A. Wedgbury, "Real-time malware process detection and automated process killing," *Secur. Commun. Netw.*, vol. 2021, pp. 1–23, Dec. 2021.

[22] A. Kharraz and E. Kirda, "Redemption: Real-time protection against ransomware at end-hosts," in *Proc. Int. Symp. Res. Attacks, Intrusions, Defenses*. Cham, Switzerland: Springer, 2017, pp. 98–119.

[23] F. Mbol, J.-M. Robert, and A. Sadighian, "An efficient approach to detect torrentlocker ransomware in computer systems," in *Proc. Int. Conf. Cryptol. Netw. Secur.* Springer, 2016, pp. 532–541.

[24] K. Lee, S. Lee, and K. Yim, "Machine learning based file entropy analysis for ransomware detection in backup systems," *IEEE Access*, vol. 7, pp. 110205–110215, 2019.

[25] C. J. Chew and V. Kumar, "Behaviour based ransomware detection," in *Proc. Int. Conf. Comput. Their Appl.*, in EPiC Series in Computing, vol. 58. 2019, pp. 127–136.

[26] S. Homayoun, A. Dehghantanha, M. Ahmadzadeh, S. Hashemi, and R. Khayami, "Know abnormal, find evil: Frequent pattern mining for ransomware threat hunting and intelligence," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 341–351, Apr. 2020.

[27] A. Kharaz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A large-scale, automated approach to detecting ransomware (keynote)," in *Proc. IEEE 24th Int. Conf. Softw. Anal., Evol. Reengineering (SANER)*, Feb. 2017, pp. 757–772.

[28] A. Kharraz, W. Robertson, D. Balzarotti, L. Bilge, and E. Kirda, "Cutting the gordian knot: A look under the hood of ransomware attacks," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2015, pp. 3–24.

[29] A. Continella, A. Guagnelli, G. Zingaro, G. De Pasquale, A. Barenghi, S. Zanero, and F. Maggi, "ShieldFS: A self-healing, ransomware-aware filesystem," in *Proc. 32nd Annu. Conf. Comput. Secur. Appl.*, Dec. 2016, pp. 336–347.

[30] M. Shukla, S. Mondal, and S. Lodha, "POSTER: Locally virtualized environment for mitigating ransomware threat," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1784–1786.

[31] N. Scaife, H. Carter, P. Traynor, and K. R. B. Butler, "CryptoLock (and drop it): Stopping ransomware attacks on user data," in *Proc. IEEE 36th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2016, pp. 303–312.

[32] D. Sgandurra, L. Muñoz-González, R. Mohsen, and E. C. Lupu, "Automated dynamic analysis of ransomware: Benefits, limitations and use for detection," 2016, *arXiv:1609.03020*.

[33] P. Zavarsky and D. Lindskog, "Experimental analysis of ransomware on windows and Android platforms: Evolution and characterization," *Proc. Comput. Sci.*, vol. 94, pp. 465–472, Jan. 2016.

[34] T. McIntosh, J. Jang-Jaccard, P. Watters, and T. Susnjak, "The inadequacy of entropy-based ransomware detection," in *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2019, pp. 181–189.

[35] Z. A. Genc, G. Lenzini, and D. Sgandurra, "On deception-based protection against cryptographic ransomware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment* Cham, Switzerland: Springer, 2019, pp. 219–239.

[36] S. Song, B. Kim, and S. Lee, "The effective ransomware prevention technique using process monitoring on Android platform," *Mobile Inf. Syst.*, vol. 2016, pp. 1–9, Mar. 2016.

[37] K. Cabaj, M. Gregorczyk, and W. Mazurczyk, "Software-defined networking-based crypto ransomware detection using HTTP traffic characteristics," *Comput. Electr. Eng.*, vol. 66, pp. 353–368, Feb. 2018.

[38] R. Moussaileb, B. Bouget, A. Palisse, H. Le Bouder, N. Cuppens, and J.-L. Lanet, "Ransomware's early mitigation mechanisms," in *Proc. 13th Int. Conf. Availability, Rel. Secur.*, 2018, pp. 1–10.

[39] Z. A. Genc, G. Lenzini, and P. Y. Ryan, "No random, no ransom: A key to stop cryptographic ransomware," in *Proc. Int. Conf. Detection Intrusions Malware, Vulnerability Assessment*. Cham, Switzerland: Springer, 2018, pp. 234–255.

[40] M. M. Ahmadian, H. R. Shahriari, and S. M. Ghaffarian, "Connection-monitor & connection-breaker: A novel approach for prevention and detection of high survivable ransomwares," in *Proc. 12th Int. Iranian Soc. Cryptol. Conf. Inf. Secur. Cryptol. (ISCISC)*, Sep. 2015, pp. 79–84.

[41] E. Kolodenker, W. Koch, G. Stringhini, and M. Egele, "PayBreak: Defense against cryptographic ransomware," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 599–611.

[42] M. S. Kiraz, Z. A. Genc, and E. Ozturk, "Detecting large integer arithmetic for defense against crypto ransomware," *Cryptology ePrint Arch., Rep.*, vol. 558, p. 2017, Jan. 2017.

[43] (2022). *What Systems Have You Seen Infected by Ransomware?* Accessed: Apr. 3, 2023. [Online]. Available: https://www.statista.com/statistics/701020/major-operating-systems-targeted-by-ransomware/

[44] (2022). *Linux Profiling With Performance Counters*. Accessed: Apr. 3, 2023. [Online]. Available: https://perf.wiki.kernel.org/index.php/MainPage

[45] (2022). *Likwid Performance Tools*. Accessed: Apr. 3, 2023. [Online]. Available: https://hpc.fau.de/research/tools/likwid/

[46] K. Keahey, J. Anderson, Z. Zhen, P. Riteau, P. Ruth, D. Stanzione, M. Cevik, J. Colleran, H. S. Gunawi, C. Hammock, J. Mambretti, A. Barnes, F. Halbach, A. Rocha, and J. Stubbs, "Lessons learned from the chameleon testbed," in *Proc. USENIX Annu. Tech. Conf.*, Jul. 2020, pp. 219–233.

[47] *Alexa Top Websites*. Accessed: Sep. 13, 2021. [Online]. Available: https://www.alexa.com/topsites

[48] *Ninite*. Accessed: Apr. 3, 2023. [Online]. Available: https://ninite.com

[49] API. (2023). *Libvirt: Virsh Tool Manual*. Accessed: Apr. 3, 2023. [Online]. Available: https://libvirt.org/manpages/virsh.html

[50] (2021). *Virusshare*. Accessed: Nov. 16, 2021. [Online]. Available: https://virusshare.com

[51] Intel. (2023). *System Programming Guide*. Accessed: Apr. 3, 2023. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011.

[53] J. Benesty, J. Chen, Y. Huang, and I. Cohen, "Pearson correlation coefficient," in *Noise Reduction in Speech Processing*. Berlin, Germany: Springer, 2009, pp. 1–4.

[54] H. Jin, Q. Song, and X. Hu, "Auto-Keras: An efficient neural architecture search system," in *Proc. 25th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Jul. 2019, pp. 1946–1956.

[55] Wikipedia. (2021). *Sensitivity and Specificity*. Accessed: Apr. 3, 2023. [Online]. Available: https://en.wikipedia.org/wiki/Sensitivityandspecificity

[56] Hat Enterprise. (2023). *Block I/O Tuning*. [Online]. Available: https://access.redhat.com/documentation/en-us/redhatenterpriselinux/7/html/virtualizationtuningandoptimizationguide/sect-virtualizationtuningoptimizationguide-blockio-techniques

KUMAR THUMMAPUDI is currently pursuing the Ph.D. degree with the Department of Computer Science, The University of Texas at San Antonio (UTSA). He is also working on analyzing and detecting ransomware and exfiltration attacks. Prior to joining the Ph.D. degree, he was an Assistant Professor for 14 years in India. He is also a Teaching/Research Assistant with UTSA for the past five years. His research interests include computer security, blockchains, TOR networks, and data science.

PALDEN LAMA received the B.Tech. degree in electronics and communication engineering from the Indian Institute of Technology, in 2003, and the Ph.D. degree in computer science from the University of Colorado, Colorado, in Spring 2013. He is currently an Associate Professor with the Department of Computer Science, The University of Texas at San Antonio. He has four years of professional experience in the software industry. His research interests include cloud computing, edge computing, big data, and cybersecurity. He has published over 34 peer-reviewed conference papers and journal articles on these topics. He served as a principal investigator (PI) or a co-PI for over seven research grants from United States federal funding agencies, including NSF, NSA, and DoD.

RAJENDRA V. BOPPANA (Senior Member, IEEE) was the Department of Computer Science Chair, from 2012 to 2018. He is currently a Professor with the Department of Computer Science, The University of Texas at San Antonio (UTSA). His research interests include computer network security, performance, and high-performance computing. He is also working on analyzing and detecting ransomware and exfiltration attacks and network security. He has published over 75 peer-reviewed conference papers, journal articles, and book chapters on these topics. He served as a principal investigator (PI) or a co-PI for over 15 research grants from United States federal funding agencies and is the sole or lead inventor for three patents. He directed UTSA's Quantitative Literacy Program (QLP) and the University-Wide Curriculum Enhancement Program (2011–2016).