RESEARCH ARTICLE

A Deep Learning Model Leveraging Time-Series System Call Data to Detect Malware Attacks in Virtual Machines

A. Alfred Raja Melvin[1] · Jaspher W. Kathrine[1] · Andrew Jeyabose[2,3] · D. Cenitta[2]

Received: 28 October 2024 / Accepted: 23 February 2025

Abstract
A Tenant Virtual Machine (TVM) user in the cloud may misuse its computing power to launch malware attack against other tenant VMs, Host OS, Hypervisor, or any other computing devices/resources inside the cloud environment of a Cloud Service Provider. The security solutions deployed within the TVM may not be reliable, as malware can disable them or remain undetected due to its hidden nature. Therefore, security solutions deployed outside the virtual machine are necessary. This research proposes deploying an Intrusion Detection System (IDS) at the Hypervisor layer, utilizing time series system call data and employing a Convolutional Neural Network (CNN) model to accurately detect the presence of malicious (malware) computer programs within virtual machines. The raw VMM system call traces are transformed into novel Time Series System Call patterns and utilized by a deep learning algorithm for training and building the classifier model. A deep learning model, CNN, is used to build the classifier model for detecting intrusions with high accuracy. It is capable of detecting both known and unknown malware. The CNN model is compared with machine learning algorithms for the results and discussions, and it outperforms ML algorithms in terms of intrusion detection accuracy when utilizing novel time series system call data..

Keywords VMI · VMM · CNN · Time series data · System calls · Deep learning · Virtual machine

1 Introduction

Computing endpoints are growing at a much faster pace due to the easy deployment of servers, applications, software stack, etc. using virtual and cloud computing facilities within the enterprise network and on a public network. VMware, Openstack, Virtual Box, Xen, KVM are a few virtualization technologies enterprises use to deploy private cloud environments. Similarly, public cloud providers like AWS, Google cloud, Microsoft's Azure, DigitalOcean, Salesforce, etc., provide faster deployment of applications, services, platform and IT infrastructure. The Enterprise does not have to purchase any hardware or worry about the space constrain to deploy the services & applications on these public cloud providers. In addition to that IoT devices are becoming very popular and deployed in many industries to automate manual tasks, integrating the cyber and physical world, and governments take up efforts to setup smart city projects to build a better place to live. Government and private sectors take up smart projects to automate everything with the aid of technologies. These technologies increase the productivity of individual users in their personal and professional life, flourishing businesses and expanding their horizons. Not only that it bridges the relationship between an employee and employer. Even though these technologies support human well being, threats keep hitting computer devices and resources making data and computer assets

 https://doi.org/10.1007/s44196-025-00781-z

Published online: 17 March 2025

at constant risk since the hacking tools like MSF are readily and easily available over internet and they are free and open source. Threat landscape is expanding with new types of attack vectors. Popular threat vectors include Malware, Intrusions, Vulnerabilities, Exploits etc.,. Encrypted threats were increased to 92% as per the sonicwall threat report where Cybercriminals make use of TLS encryption to deliver malware and threat over the network. Cryptojacking threat vector has increased to 409% in India though it was decreased globally. Mcafee reported in 2020 that the overall cloud service usage has increased by 50% with a 144% increase in usage by manufacturing industries and a 114% increase in education institutions [1]. According to the 2022 cloud security report of fortinet unauthorized access is still considered as one of the biggest security threats in the cloud platform [2]. Almost 95% of all the organizations using cloud platform are extremely concerned about cloud security.

Therefore, enterprises use defensive technologies to protect computer assets against new attack types and malware programs. These kinds of zero-day attacks are difficult to detect as prior knowledge about the attack is unknown. Many malware programs are intelligent enough to hide their presence from intrusion detection systems.

1.1 Problem Statements

1. The security solutions deployed at TVM are not reliable since the malware may disable (deactivate) the security solution (Antivirus program) or in some cases the Antivirus program may not detect these malicious activities due to the hidden nature of malware.
2. The Malware Attack datasets available in the Research community are based on static features and it is not sufficient to build a classifier model to detect unknown malware.
3. Many existing system call datasets are extracted from the Linux platform only except ADFA-WD dataset which contains intrusive traces from the Windows platform, but it contains offset values only.
4. Many malwares are obfuscated and hence static features are not sufficient to detect its malicious nature.

1.2 Key Contribution of this Research

1. A novel time series system call pattern from the raw VMM system call traces was developed. The time series system call data is developed in continuation of the previous work on the VMM malware attack dataset with dynamic system call traces for the development of an Intrusion Detection System in the cloud [3].
2. The proposed system is implemented with Convolutional Neural Network (CNN) deep learning algorithm for the detection of intrusions using time series system call patterns.
3. Performance evaluation of the proposed system through standard performance evaluation metrics.

Consider a cloud scenario shown in Fig. 1 where tenant virtual machines are hosted on a Infrastructure-as-a-Service model in a private environment. Here, the applications, software stack and services will be running inside the virtual machines. The users having laptops or mobile devices or any computing devices get access to these hosted services running inside the virtual machines using a web browser or any compatible client applications through the internet. In the IaaS model the customers/tenant can access the virtual machines to manage the application, software stack, services etc., hosted inside the virtual machines. The malware attack against Virtual Machine and the possibility of using virtual machines as attack source is considered in this work. In this environment, a tenant user may misuse the VM for malicious purpose by exploiting the weakness in the tenant VM. In this case, the malicious tenant users act as internal attacker in the cloud network. Therefore, the attacker may be a malicious tenant user or internal attacker. The objective is to detect these types of attacks from VM introspected data at Hypervisor. The proposed architecture is capable of detecting known and unknown (zero-day) malware attacks at VM as per the findings. It is assumed that the cloud service provider side—cloud administrator, VMM and Dom0 (monitoring VM) are trusted and secure. In addition, the privacy concerns of the tenant users are

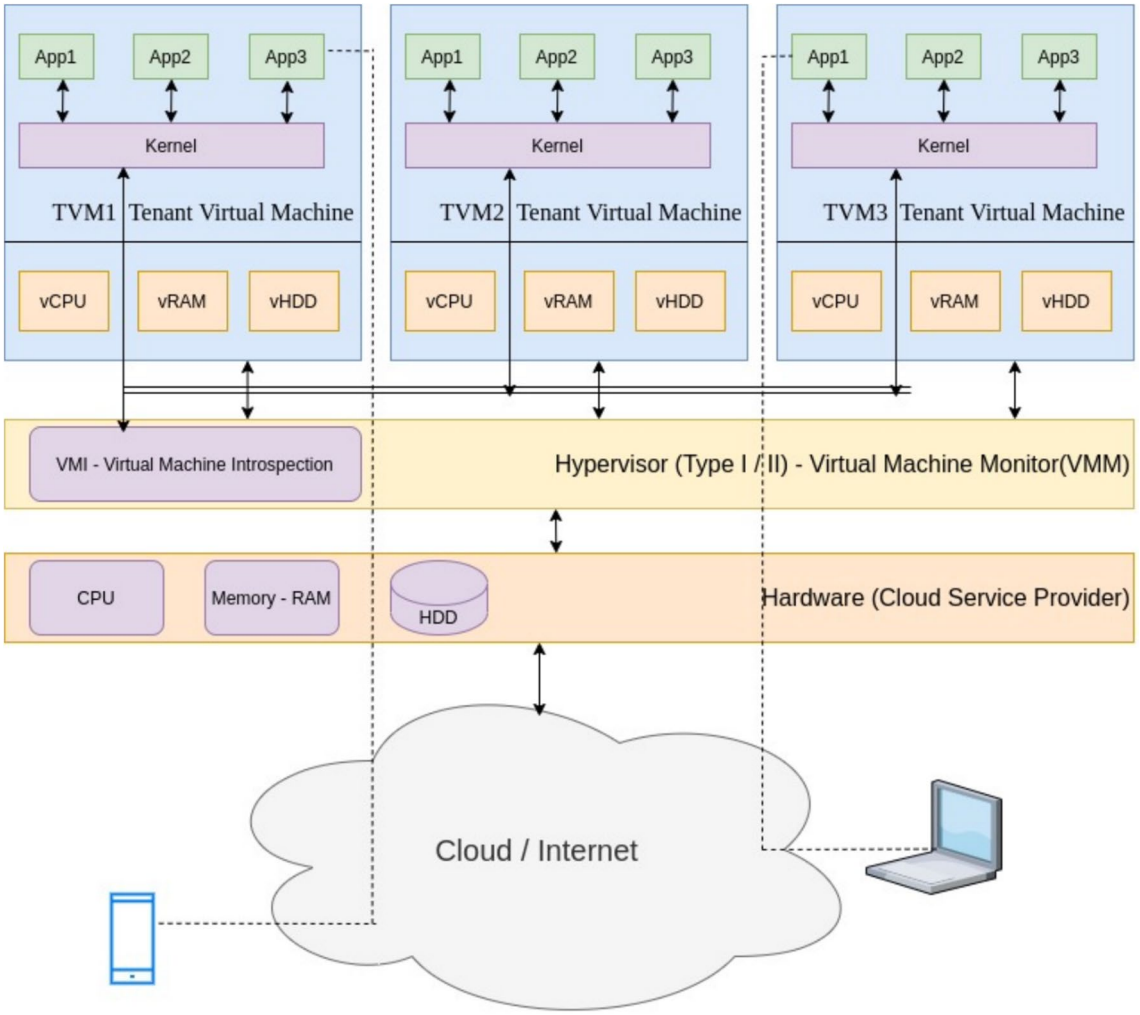

Fig. 1 Virtual machine introspection in a private cloud infrastructure

not considered. The cloud administrator has access to the application-specific information of VM through the hypervisor or virtual machine monitor. The VM control operations like VM creation, deletion, modification, reconfiguration, cloning, etc., are performed from the VMM or Hypervisor layer by the cloud administrator. The execution of malware or benign samples into the virtual machine is initiated from the VMM layer during the experiment. The system call traces of each sample during execution are also collected from the VMM layer. So there are no special agents needed or installed inside the virtual machine to collect the system call traces of program executions.

The Tenant Virtual Machines (TVM) are the end-points where applications and software services are executed with the guest operating system. The guest OS can be Microsoft Windows XP/Vista/7/10 or any Linux variants (kernel version 2.6 or later). During the training phase, these end-points act as the data source to build the classifier model. The tenant users or customers give details on the application and software services they use in the TVM during the SLA agreement with CSP. These software details act as the guideline to build the base profile for the TVM during the later sections in the training phase. Based on the customer requirements additional protection services such as Antivirus(AV), and Personal Firewall, may be deployed. For each registered service or process running in TVM, the system activity details are extracted by the system call tracer component.

The VMI technique extracts the sequence of system calls for each selected processes and forwards it to the Pre-processor component. The extraction feature of the Hypervisor is implemented using libVMI and Drakvuf.

The libVMI framework provides Virtual Machine Introspection capability and supports both Xen and KVM hypervisor. Xen hypervisor is used in this research work. The libVMI framework can be used to trace system call sequences from guest OS (supports both Microsoft Windows and Linux platforms). We have used Windows 7 as our guest OS since majority of malware attack are targeted towards the Windows platform. The VMI extracts the system call traces of TVM by continuously trapping all kernel functions. The same method can be employed to trace the system calls in both Windows and Linux guest OS. Windows system calls can be traced by extracting names starting with 'Nt' and Linux system calls can be traced by extracting names starting with 'sys_'.

This research proposes using a CNN model with time series system call data to classify and detect malware attacks in virtual machines on the cloud. While RNN and LSTM are well-suited for time series data and prediction problems, they are not as efficient for solving classification problems. In this research, RNN or LSTM can effectively predict the next system call in a sequence, but they cannot efficiently distinguish between malware and normal system behavior. Also, CNN was chosen over RNN/LSTM because CNN requires less computing power to build the models. CNN has a fixed number of inputs and outputs, making it more efficient in terms of computing resources compared to RNN/LSTM. In the proposed CNN model, the output is either a Malware or Normal sample. On the other hand, RNN/LSTM handles an arbitrary number of inputs and has variable output length, which affects the total computational time and efficiency.

Next section presents the existing work, Sect. 3 provide details on the Time Series system call dataset and CNN model for the detection of malware attacks. Section 4 discusses about results and comparison with ML algorithms. Section 5 concludes and provide details on future scope.

2 Existing Work

The idea of using a system call traces for the development of Anomaly-based Intrusion Detection System (IDS) was first proposed by Forrest [4]. In his work, short sequences of system calls are used to build the normal profile in the training phase. During the testing phase, sequences are compared and based on Hamming distance the anomaly score for the new sequences is calculated to detect intrusions. Later Forrest and others developed different data modelling to represent the normal behaviour of system call sequences [20]. An immediate system call sequence methodology was proposed by Gupta and Kumar [5]. They first develop signatures from system calls for programs in their normal operations. Signature is used as a baseline to identify intrusions during their abnormal executions. They used the UNM dataset to train the system. Even though it can detect zero-day attacks, the system has a very high false positive rate. Creech and Hu [6] developed a semantic structure methodology based on continuous and discontinuous system calls to differentiate between normal and anomalous program execution. They developed the ADFA dataset during the research work in addition to the usage of existing datasets – KDDCup and UNM datasets.

Virtual Machine Introspection (VMI) technique was first introduced by Garfinkel and Rosenblum [7] which is a monitoring technique used at the hypervisor layer to observe the state of the virtual machines. It is mainly helpful for debugging and forensic analysis. KVMInspector is a VMI-based malware detection system for the cloud proposed by Mishra [8]. They used the KVM hypervisor and deployed both inside & outside the virtual machines. After the basic process verification at the virtual machine level, detailed analysis is carried out in the hypervisor. They used the University of New Mexico and University of California datasets for the implementation. A hypervisor-based cloud anomaly detection model was proposed by Aldribi [9]. An in-VM agent-based malware detection system for virtual machines in cloud computing was proposed by Patil [10]. A security framework based on VMI for the cloud is proposed by Borisaniya [11]. An extensive survey on host-based IDS was done by Bridgeset al. VMI with memory forensics for detecting and characterizing unknown malware with the help of ML algorithms was proposed by Ajay Kumara [12]. VMGuard is a VMI-based security architecture for the detection of intrusions in the cloud proposed by Mishra et al. [13]. They used the Drakvuf tool to trace the system calls at the xen hypervisor. The system calls are pre-processed using

Bag-of-ngrams approach with Term Frequency – Inverse Document Frequency and applied Random Forest algorithm to build the classifier model. They used UNM datasets and Windows malware datasets from the University of California for training the classifier model. Varadharajan et al., has proposed the implementation of Integrated Security Architecture for the cloud by incorporating security policies for secure interactions between applications and virtual machines [14]. They tested the Chinese wall policy on Host machines with Xen hypervisor. As per policy, conflicting VMs are not allotted mistakenly on the same computing resource (Host). This improves the resilience of the cloud system. Many researchers have utilized host data to develop Host-based IDS [21]. HIDS monitors the processes, user activities, file system changes, etc., to detect the presence of suspicious activity and raise alarms if required. HIDS complements with VMI technique, and they can be integrated to develop effective IDS systems.

Researchers have also used CNN and different deep-learning models to detect intrusions. Yifei et al., proposed a behavioural semantics enhancement method of system call sequence to detect intrusions based on obfuscated system call sequences [15]. First they build a unified length of sequences for the system call traces given as input. Next, the unified sequences were converted into a vector representation. Finally, feature extraction and data classification were carried out using the text-CNN deep learning technique. As per the experimental results their implementation outperforms other system call-based HIDS implementation. The major drawback in their implementation is the requirement of high computing resources and hardware costs involved to train the deep learning model. Ying-Dar Lin et al. proposed the use of data from multiple sources with machine learning algorithms to build the classifier models for the detection of intrusions [16]. They used packet flows, system logs and host statistics data sources to train and classify the ML models to detect intrusions. As per their observation based on the F1 score value, the model built with host statistics yields better results than traffic flows and system logs. Ren-Hung et al., proposed the use of multi-data sources with deep learning models and used an ensemble technique to combine the results of each DL model to conclude between Normal or Intrusion. They built three different DL models each for different data sources. CNN model is applied to network traffic data for automatic feature selection, LSTM is applied to system log data and DNN is applied on host statistics data to improve the classification performance. They evaluated the proposed system based on F1 score value and achieved the value 1.0 (full value) so it outperforms the previous implementations by other researchers. The major drawback in their research is the use of a Simulator program to generate the datasets which includes the network traffic data, system log data and host statistics data. Vinayakumar et al., have proposed the use of CNN and other deep-learning models to detect intrusions using the NIDS dataset [17]. They used the KDDCup dataset and transformed the network traffic data into time series with a supervised learning method. They found that the CNN model performs extremely well compared to the machine learning algorithms. The major drawback in their implementation is the use of the KDDCup dataset which is very old and may not be efficient in detecting novel intrusions. Chawla et al. proposed Host-based anomaly IDS system using Recurrent Neural Networks with GRU [18]. They determine the probability of a specific system call sequence occurring from a language model trained on normal sequences using the ADFA dataset. They developed a sequence anomaly detection using language modelling. Any sequence with low probability is classified as an anomaly. They could achieve the highest detection accuracy of 81%. Nidhi Joravija et al., proposed a deep learning-based HIDS utilizing system call sequence with frequency and images in a containerized cloud environment. They transformed the host system logs into images and used Leipzig Intrusion Detection Dataset 2019 to develop the CNN model [22]. Silambarasan et al., proposed a cloud-based IDS using a Bidirectional Long Short Term Memory(Bi-LSTM) deep learning model with Adaptive Rat Swarm Optimization (ARSO) Algorithm. The significant features are selected based on the ARSO algorithm and used Bi-LSTM to build the classifier model. They used the NSL-KDD dataset to training the model [23]. Virtual Machine Profiler was proposed by Shun-Wen [19]. The NIDS implementation using the CNN model is not suitable for cloud environment to protect virtual machines. The HIDS implementation with the CNN-RNN model trained using the ADFA Linux dataset is not robust since it needs to be deployed inside the virtual machine. Next section describes the implementation of the proposed work.

 https://doi.org/10.1007/s44196-025-00781-z

3 Proposed Work

The proposed work is depicted in Fig. 2. There are two stages namely the training and testing phase. During the training phase, the VMM system calls in time series data were provided as input to the CNN model. The time series system call data were processed by the CNN model to recognize patterns by adjusting its parameters to minimize the loss function and maximize the accuracy. The outcome of the training phase is a classifier model for solving binary classification problem to accurately differentiate between benign and malware samples from the time series system call traces. The details of the CNN model and its implementation are described in Sect. 3.4. The steps involved in the training phase can be realized from algorithm 1. During the testing phase, the VMM system call traces will be collected and preprocessed to transform the data into time series system calls, and it will be evaluated using the CNN binary classifier model built during the training phase. These details can be realized from algorithm2. The experimental setup (shown in Fig. 3) and details of time series system call dataset generation from VMM system calls traces are explained in the next section (Sect. 3.1).

3.1 Generation of Time Series System Call Data

The existing System Call datasets are traced using special (agent) programs from the computing environment. For example, the strace program is used in the Linux platform and procmon is used in Windows platform. The making of KDD cup, UNM, ADFA and LID datasets use some sort of agent programs. Many other malware datasets (non-system call) available in public are based on static features. Researchers have used these datasets to develop IDS/MDS systems with their proposed algorithms and approaches. During IDS deployment, these systems need in-agent programs to be installed inside the computing environment to monitor the system call traces. Certain malware programs are intelligent enough to detect the presence of a monitoring agent and compromise its operation by its infection. Certain malware does not exhibit its malicious nature. Static malware datasets are not efficient since many malwares are obfuscated to look like benign but shows their malicious nature during their execution only. Therefore, a reliable way and dynamic feature datasets are important for the development of efficient IDS. Above all, it is essential that these monitoring (agent) programs to be deployed or present outside the computing environment (Outside VM), i.e., at VMM or hypervisor layer in the cloud, so it may not be compromised.

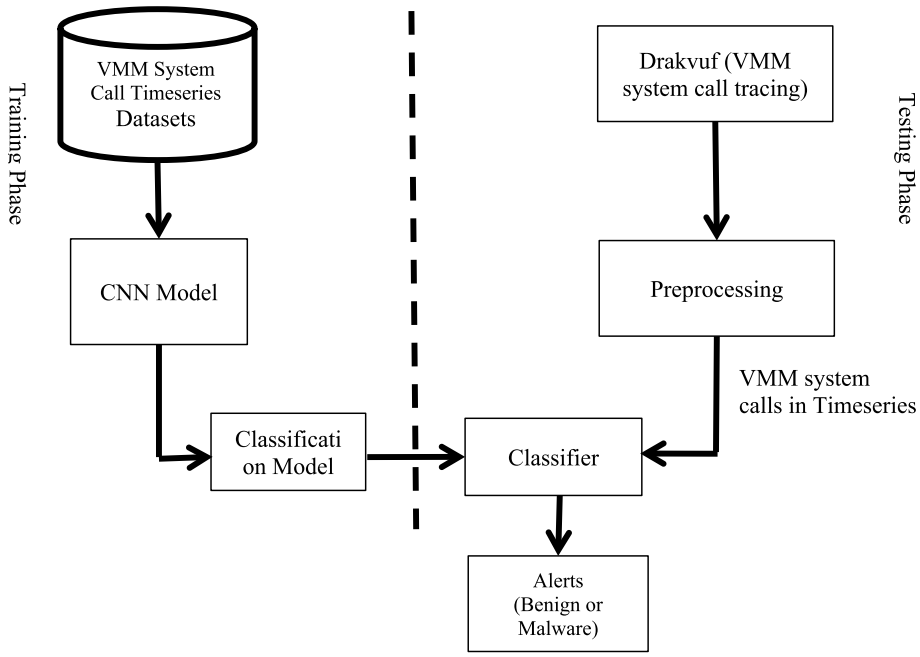

Fig. 2 Hypervisor based IDS with time series system call—proposed model

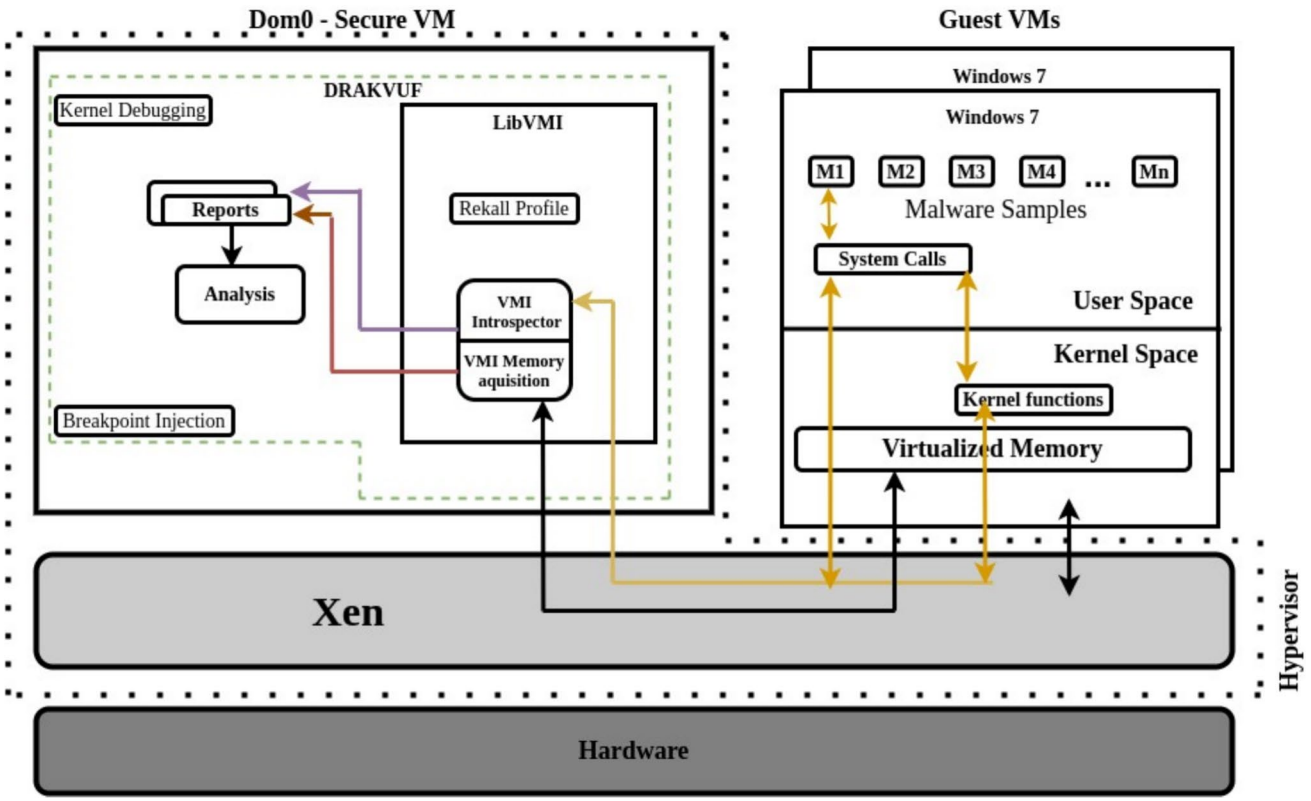

Fig. 3 Experimental setup—VMM system call tracing

A test-bed environment was created to generate the dataset, with the Dell workstation with i7 processor, 1 TB, 16 GB RAM, running Ubuntu 16.04 as Host OS. Xen with libVMI is configured as VMM. Xen tools (CLI) is used to manage virtual machines at the hypervisor. The experimental setup is depicted in Fig. 3. Drakvuf tool is used to trace the system calls of virtual machines. During the experiment, two VM's (same configuration) are instantiated each running Windows 7 OS (Guest OS) with 1 vCPU, 30 GB of storage, 1 GB of main memory. Virtual Machine Introspection is a technique that allows an admin user to collect digital traces (system call trace) of in-VM activities at the VMM layer (i.e., from outside the VM) without the need to install any in-agent programs inside VM. In-order to train the Hypervisor or VMM to become intelligent using Machine Learning algorithms, it needs access to the system call traces of in-VM activities of normal programs and malware. That is the primary reason for the proposal of this VMM malware attack dataset which may be bench-marked and can be used by other researchers for the development of new algorithms or IDS/MDS. Malware samples are collected from VxHeaven, and the Zoo project repositories. The details of the malware samples are shown in Table 1. Similarly, benign samples including system tools, accessory programs, network tools, office package programs are installed in the guest OS (see Table 2). Malware samples are executed one at a time inside the VM environment and its respective system call traces are collected. Similarly, each benign program is executed many times (different runs) and its respective system call traces are collected for each run. For example, the Adobe Reader program is executed for

Table 1 Malware samples execution trace details

#	Source	No. of samples	Total system call traces
1	VxHeaven	906	66,76,127
2	theZoo	31	3,29,293

 https://doi.org/10.1007/s44196-025-00781-z

Table 2 Benign samples execution trace details

#	Benign Program	Program(s)	Runs	Total System Call Traces
1	Adobe Reader	1	3	3,72,081
2	Chrome	1	2	6,32,769
3	Microsoft Excel	1	2	85,204
4	Microsoft Word	1	2	72,667
5	Powerpoint, Notepad, Wordpad	3	1	51,976
6	Windows Program set 1	23	1	1,28,040
7	Windows Program set 2	13	1	17,647

3 runs each with different settings and its respective system call traces are collected. The benign samples taken for an experiment are shown in Table 2.

Algorithm 1 Hypervisor based IDS—Training Phase

Input: *VMM Time Series System Call Traces*
Output: *CNN Binary Classifier Model*

Foreach *Time Series System Call Traces* (TS_i)
    Initialize count = 0
    Initialize sum = 0
    Foreach $Time_j$ *in* TS_i
        Calculate sum = sum + Time$_j$
        Increment count [count++]
        while *(sum > 0.1)*
            Append count to DS_i
            Reinitialize count = 0
            Reinitialize sum = 0
        end
    end
    Append padding bits if required to DS_i
    Append class label to DS_i
end
Build CNN Model (1d) with time series dataset DS_i
return *CNN binary classifier model*

```
syscall,1574757337.676809,0 0x1773d000,"\Device\HarddiskVolume2\Windows\System32\svchost.exe",0,ntoskrnl.exe,NtSetTimer,7,IN,HANDLE,TimerHandle,0
x9c,,,IN,PLARGE_INTEGER,DueTime,0x302f908,,,IN,PTIMER_APC_ROUTINE,TimerApcRoutine,0x0,,,IN,PVOID,TimerContext,0x0,,,IN,BOOLEAN,WakeTimer,0x7fefa6
0e300,,,IN,LONG,Period,0x0,,,OUT,PBOOLEAN,PreviousState,0x0,,
syscall,1574757337.676890,0 0x1773d000,"\Device\HarddiskVolume2\Windows\System32\svchost.exe",0,ntoskrnl.exe,NtWaitForWorkViaWorkerFactory,2,IN,H
ANDLE,WorkerFactoryHandle,0x88,,,OUT,PFILE_IO_COMPLETION_INFORMATION,MiniPacket,0x302fc00,,
syscall,1574757337.676946,0 0x1773d000,"\Device\HarddiskVolume2\Windows\System32\svchost.exe",0,ntoskrnl.exe,NtWaitForWorkViaWorkerFactory,2,IN,H
ANDLE,WorkerFactoryHandle,0x88,,,OUT,PFILE_IO_COMPLETION_INFORMATION,MiniPacket,0x302fc00,,
syscall,1574757337.720911,0 0x9870000,"\Device\HarddiskVolume2\Program Files\Windows Media Player\wmpnetwk.exe",0,ntoskrnl.exe,NtWaitForSingleObj
ect,3,IN,HANDLE,Handle,0x2f8,,,IN,BOOLEAN,Alertable,0x0,,,IN,PLARGE_INTEGER,Timeout,0x184fa08,,
syscall,1574757337.860787,0 0x174b3000,"\Device\HarddiskVolume2\Windows\System32\svchost.exe",0,ntoskrnl.exe,NtReleaseWorkerFactoryWorker,1,IN,HA
NDLE,WorkerFactoryHandle,0x88,,
```

Fig. 4 Raw traces collected from hypervisor or VMM layer

Table 3 Distribution of Samples in the Dataset

S.No	Criteria	Malware samples	Benign samples	Total samples
1	Full dataset	954 (96.36%)	36 (3.63%)	990
2	Training	600 (96.77%)	20 (3.22%)	620
3	Testing	354 (95.67%)	16 (4.32%)	370

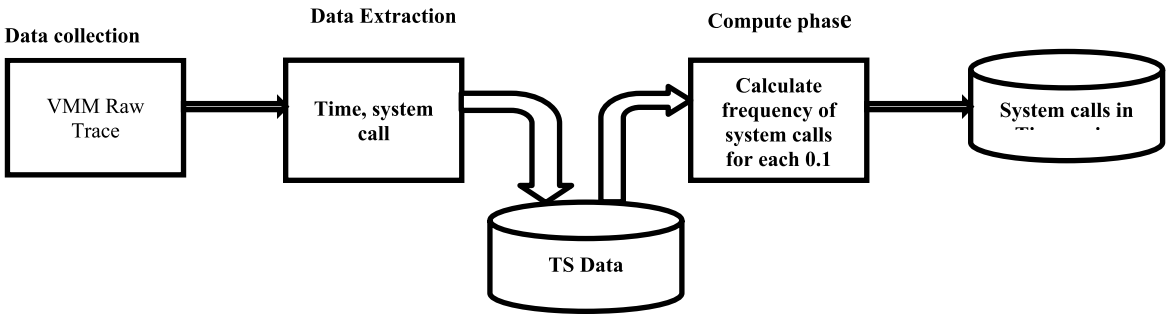

Fig. 5 Process of Time Series system call data generation

3.1.1 Structure of Raw VMM Trace

The raw audit data of VM (Fig. 4) collected contains a large number of unstructured data. It includes details about system calls, file tracer, registry changes, pool allocation, etc., From this raw data, the time of execution of the system call and system call names are extracted as Time system call vector. This Time system call vector (shown in Fig. 3 & 4) is a sequence of system calls executed during the execution of the program (malware or benign) and its time of execution inside the VM. For each name of the program (malware or benign), a separate file containing Time—system call vector was drawn out. The number of system call sequences rendered during each time intervals may be significant in accurately differentiating between Normal and Intrusive programs. Based on this hypothesis a novel time series dataset based on system calls executed was generated.

There are around 1004 malware samples and 48 benign samples taken to generate the training data. 50 out of 1004 malware samples and 12 out of 48 benign samples failed to execute in the test environment since some samples need an entry point to start execution and some failed because Windows failed to execute. Remaining samples (954 malware + 36 benign = 990 total samples) generated the system call (syscall) traces successfully which contributed to the development of the VMM dataset. The details of the distribution of samples are shown in Table 3. The VMM dataset containing 990 records as total was divided into 620 records for training and 370 records for testing data. This split-up of data into training and testing was done manually. The training and testing data includes both malware and normal samples. The detail of the split-up of this dataset is depicted in Table 3.

3.2 Pre-Processing Steps

The steps involved in the conversion of VMM raw traces to time series data are depicted in Fig. 5. The first stage is the data collection phase where the raw traces were collected from VMM or hypervisor. The second stage is the data extraction phase where the system call name and time of execution of system calls are extracted (shown in Figs. 4, 5).

 https://doi.org/10.1007/s44196-025-00781-z

Fig. 6 Time, system call sequence in a malware sample

```
1571306339.916519,NtOpenFile
1571306339.917086,NtQueryInformationProcess
1571306339.917159,NtOpenKey
1571306339.917405,NtQueryValueKey
1571306339.917611,NtOpenKey
1571306339.917940,NtQueryValueKey
1571306339.918072,NtClose
1571306339.918130,NtQueryInformationProcess
1571306339.918180,NtQueryInformationProcess
1571306339.918238,NtQueryInformationProcess
1571306339.918287,NtQuerySystemInformation
1571306339.918338,NtQuerySystemInformation
1571306339.918384,NtAllocateVirtualMemory
1571306339.918478,NtFreeVirtualMemory
1571306339.918588,NtAllocateVirtualMemory
1571306339.918651,NtQuerySystemInformation
1571306339.918758,NtAllocateVirtualMemory
1571306339.918810,NtFreeVirtualMemory
1571306339.918897,NtOpenDirectoryObject
1571306339.918981,NtOpenSymbolicLinkObject
1571306339.919033,NtQuerySymbolicLinkObject
1571306339.919104,NtClose
```

Fig. 7 Time, system call sequence in a benign sample

```
1574775587.788049,NtOpenFile
1574775587.788237,NtCreateSection
1574775587.788527,NtMapViewOfSection
1574775587.788643,NtQuerySection
1574775587.788689,NtClose
1574775587.788733,NtClose
1574775587.788787,NtProtectVirtualMemory
1574775587.788850,NtProtectVirtualMemory
1574775587.788910,NtOpenSection
1574775587.788956,NtAllocateVirtualMemory
1574775587.789005,NtQueryAttributesFile
1574775587.789184,NtOpenFile
1574775587.789335,NtCreateSection
1574775587.789535,NtMapViewOfSection
1574775587.789666,NtQuerySection
1574775587.789714,NtClose
1574775587.789772,NtClose
1574775587.789831,NtProtectVirtualMemory
1574775587.789896,NtProtectVirtualMemory
1574775587.789946,NtProtectVirtualMemory
1574775587.790051,NtOpenKey
1574775587.790134,NtQueryAttributesFile
1574775587.790443,NtOpenSection
1574775587.790492,NtMapViewOfSection
1574775587.790617,NtUnmapViewOfSection
1574775587.790678,NtClose
```

The extracted time, system call data are stored in the file system. Data extraction process iterates through each sample's raw execution traces. So the data stored by the end of the extraction phase includes time and system call details for all samples individually. From the trace file of each program (benign / malware), the system call's execution time and the name of the system calls are extracted, recorded and stored individually. An example of

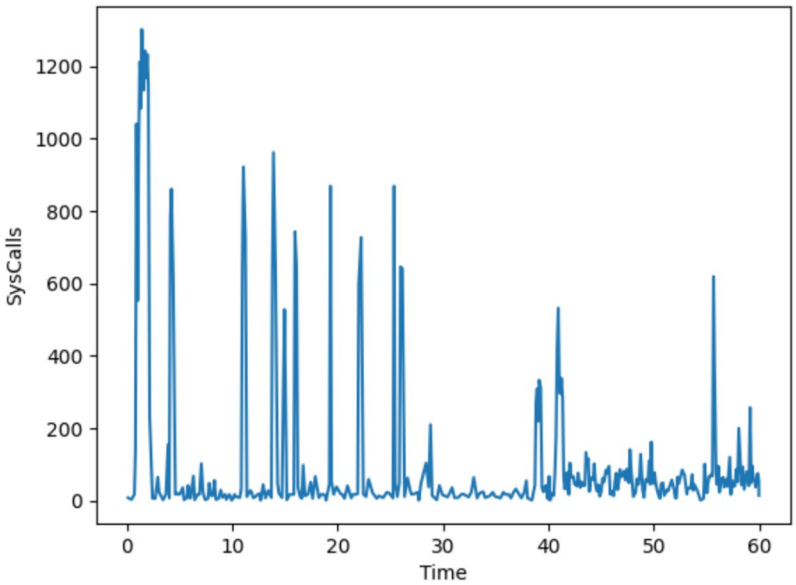

Fig. 8 Time Series data showing frequency of system calls over a time period of 60 s

a trace containing the time of execution and the system call name for a benign program (calculator) is shown in Fig. 6. Similarly, the screenshot of a malware sample is shown in Fig. 7. The recorded data are further analyzed and converted into time series system call data. The recorded data is Time Syscall data—TS_{data}. The VMM raw data was first pre-processed to extract the system call name and the time of execution of a syscall. The content of this pre-processed trace is shown in Figs. 6 and 7. The time value is represented in Unix epoch format. The pre-processed VMM traces were given as input for this algorithm to generate the time series system call data for the VMM system call traces. All the pre-processed VMM traces were stored in a directory path in the file system in a system that is used to train the model.

For each pre-processed trace file, initialize the sum and count variable to value zero and traverse through each line in the file and add up the time and increment the count value. When the sum reaches 0.1, store/append the calculated value of count to an array and reinitialize the value of sum and count to zero and repeat this process till the end of pre-processed trace file. By the end of each iteration, an array containing count values every 0.1 s will be generated for each trace file. The array represents time series data for a sample. An example of such time series data is shown in Fig. 8.

The algorithm to get the system call data in time series format from Time—System call data (TS_{data}) is depicted in algorithm 1. For each 0.1 s of time interval the no. of system calls rendered is calculated to get the time series system call data. A graph of such a time series pattern generated is shown in Fig. 8. The Time—System call data (shown in Figs. 4, 5) contain time and system call sequences. The time data is in epoch time format. No other existing system call datasets in this field of research contain time data which is a novelty in this research work.

 https://doi.org/10.1007/s44196-025-00781-z

Algorithm 2 Hypervisor-based IDS—Testing Phase

```
Input: VMM Time Series System Call Traces, CNN binary classifier model
Output: Benign or Malware
Foreach Time Series System Call Trace (TS_i)
    Initialize count = 0
    Initialize sum = 0
    Foreach Time_j in TS_i
        Calculate sum = sum + Time_j
        Increment count [count++]
        while (sum > 0.1)
            Append count to test Dataset TDS_i
            Reinitialize count = 0
            Reinitialize sum = 0
        end
    end
    Append padding bits to TDS_i  if required
end
Evaluate the test time series Dataset TDS_i using CNN binary classifier model
return Benign or Malware
```

3.3 Machine Learning algorithms for Malware Attack Detection

Machine learning algorithms are applied on the VMM system call time series dataset (labeled) to build the classifier models to detect the intrusions. This task to solve is a binary classification problem and falls under a predictive model. Supervised machine learning algorithms are applied to solve the classification problem. The labeled dataset was generated as described in the previous section. The time series system call patterns are translated into mathematical or tree representation when machine learning algorithms are applied. WEKA tool is used to carry out the experiment with tenfold cross validation for all ML algorithms. The input data is split into 10 equal parts. 9 subparts are used for training and one subpart is used for validation. The cross-validation process repeats 10 times for every combination. Finally, the outcome of each fold is averaged to get the overall performance of the ML models. Popular ML algorithms like J48 (C4.5 decision tree), Random Forrest, RIPPER, KNN, SVM and Naive Bayes are applied. The detection accuracy of ML algorithms is promising. The results are depicted in Table 4. Decision tree, Random Forrest and KNN algorithms provide better accuracy than others with an average of 98%.

The advantage of building ML models is to detect intrusions (malware) from system call traces.

i. Adds intelligence to the system (i.e. ability to detect malware or intrusions similar to known malware with the system already trained)
ii. The system will be trained to detect zero-day attacks also (i.e., ability to detect unknown malware)

Table 4 Performance of ML algorithms on VMM Malware dataset

	J48	RF	RIPPER	KNN	SVM	NB
Accuracy	98.18%	98.08%	97.78%	98.28%	96.36%	95.35%
FPR	1.82%	1.92%	2.22%	1.72%	3.64%	4.65%
Precision	0.981	0.98	0.977	0.982	0.964	0.977
Recall	0.98	0.99	0.98	0.96	0.95	0.93
F Measure	0.98	0.99	0.98	0.96	0.94	0.93
ROC	0.834	0.976	0.841	0.804	0.5	0.952

iii. Random Forrest algorithm gives a higher ROC value of 0.976 than other ML algorithms

3.3.1 Drawbacks

It is time consuming preparing the input data for training the model. There are two steps in the preparation of data. First, the system call sequences are extracted from raw trace, then the sequences are transformed into a time series patterns. However, the performance of detection accuracy and results are incomparable to the time consumption during the preprocessing phase.

The preprocessing time depends on the volume of system calls generated for the sample programs. Though it may take longer time its trivial compared to the performance of the proposed model and its accuracy.

3.4 1d CNN Model for Intrusion Detection on Time Series System Call Dataset

In this research, a deep learning model is proposed for detecting intrusions. The model implements a fully connected Convolutional Neural Network (CNN) using time series system call data. It aims to accurately classify benign and malware programs to solve the classification problem. The CNN model is set up with an input layer to feed the training data, three hidden layers to execute the convolution function and process the data, and an output layer to display the classified output (benign or malware). Since the input time series is 1D data, CNN filters slide over the time series sequence to extract the feature map. As the data is flattened and 1D, pooling layers are not necessary. The output from the stacked CNN layers undergoes processing to complete the convolution operation. The ReLU activation function is used in each neuron of the hidden layers. The summary of the CNN model built is shown in Fig. 9.

Hyperparameters are defined (listed in Table 5) in the CNN with a kernel size of 3, 64 filters, and batch normalization using a batch size of 32. These parameters are crucial for defining the hyperplane that accurately separates or differentiates two classes to solve the classification problem. The CNN model was constructed using

Fig. 9 CNN model summary for VMM system call time series data

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 598, 1)]          0
_________________________________________________________________
conv1d (Conv1D)              (None, 598, 64)           256
_________________________________________________________________
batch_normalization (BatchNo (None, 598, 64)           256
_________________________________________________________________
re_lu (ReLU)                 (None, 598, 64)           0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 598, 64)           12352
_________________________________________________________________
batch_normalization_1 (Batch (None, 598, 64)           256
_________________________________________________________________
re_lu_1 (ReLU)               (None, 598, 64)           0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 598, 64)           12352
_________________________________________________________________
batch_normalization_2 (Batch (None, 598, 64)           256
_________________________________________________________________
re_lu_2 (ReLU)               (None, 598, 64)           0
_________________________________________________________________
global_average_pooling1d (Gl (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 2)                 130
=================================================================
Total params: 25,858
Trainable params: 25,474
Non-trainable params: 384
```

 https://doi.org/10.1007/s44196-025-00781-z

Table 5 Hyper Parameters with Detection Accuracies

Hyperparameters	Optimizers				
	Adam	SGD	RMSProp	AdaDelta	AdaGrad
Epoch	97, 120, 150, 200, 350	73, 270, 355	63, 200, 403	51, 500	193, 268, 381
Batch size	32	32	32	32	32
Train data	42.62%	42.62%	42.62%	42.62%	42.62%
Validation data	20%	20%	20%	20%	20%
Test data	37.37%	37.37%	37.37%	37.37%	37.37%
Min. learning rate	0.0001	0.0001	0.0001	0.0001	0.0001
Test Accuracies	96.21—99.19	95.13—95.40	95.13—99.18	95.13—95.67	95.13—95.40

the Keras platform in Python with the TensorFlow (TF2) framework. The hyperparameters were configured using KerasTuner. The training parameters were set with a batch size of 32, and the model was trained for 500 epochs with early stopping. The training model was executed multiple times with different optimizers. The details of hyperparameters with different optimizers are listed in Table 5. The detection accuracies and validation losses of the CNN model with different optimizers are shown in Figs. 10, 11, 12, 13 and 14. Adam optimizer performs better with higher detection accuracies than other optimizers because it makes use of good properties from both

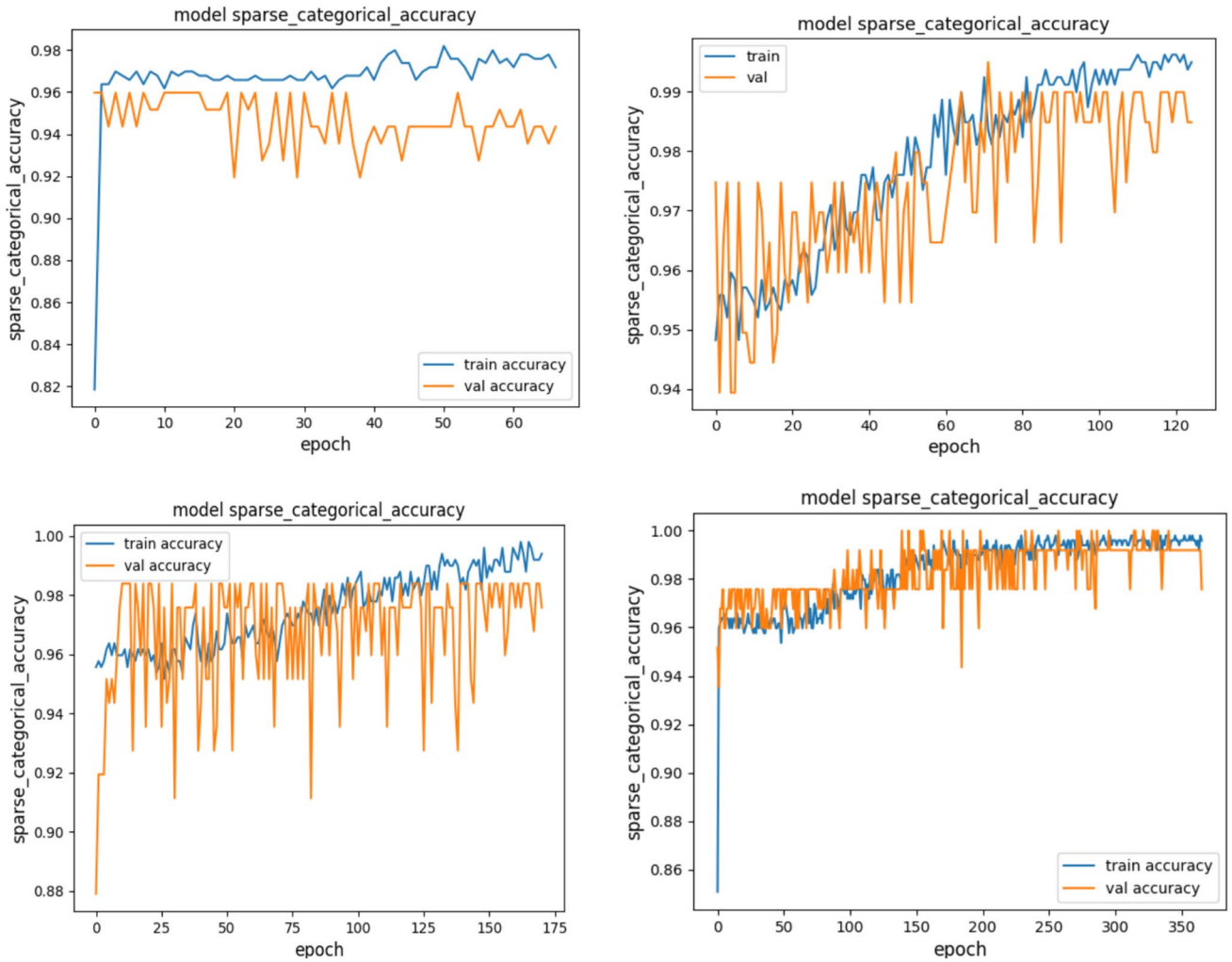

Fig. 10 Training and validation accuracies of CNN model with different epochs—Adam optimizer

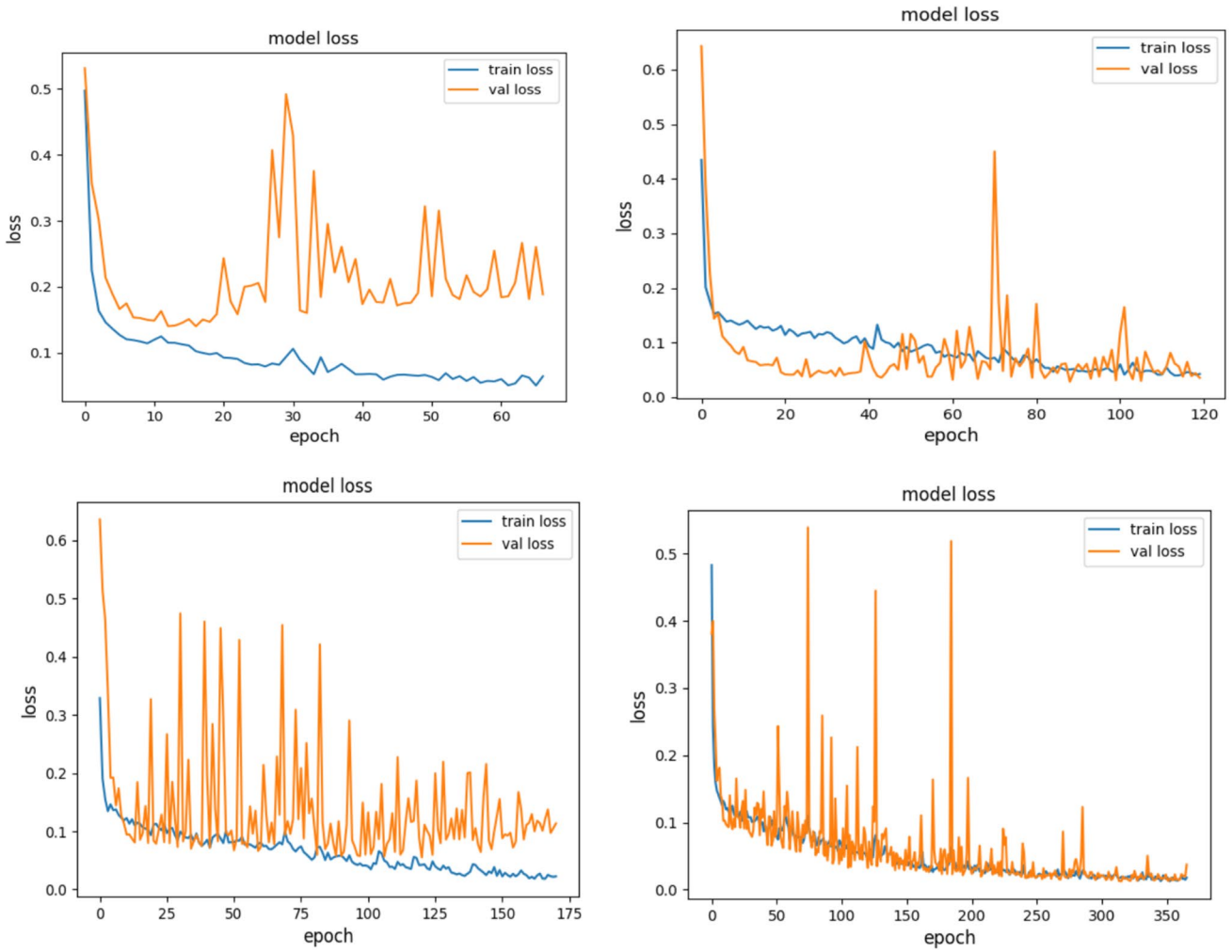

Fig. 11 CNN model loss with different epochs—Adam optimizer

Adadelta and RMSProp. As the number of epochs increased the model provide better results in terms of accuracy and loss irrespective of the optimizers used. The model attains stability starting from epoch 80 and the model starts overfitting when it reaches epoch 120. With early stopping is enabled the CNN model took was built with different epochs randomly to find out the optimal epoch. For Adam optimizer, the model took 267 epochs to train the model and reached a detection accuracy of 99.4% (training) and 98.39% (validation). During another run, the model took 97 epochs to train the model and reached a detection accuracy of 98.79% (training) and 94.35% (validation). Similarly, the model took 200 epochs to train the model and reached a detection accuracy of 99.6% (training) and 97.58% (validation). In addition, the model took 350 epochs to train the model and reaches a detection accuracy of 99.8% (training) and 100% (validation). The model's graph showing training and validation accuracy is depicted in Fig. 10. As observed, the model attains stable accuracy after 100 epochs. The CNN model's results on training and test data are presented in Table 6. The training data contains 620 records and test data contains 370 records. Both the training data and test data are labeled dataset.

The CNN model's graph with training and validation loss is shown in Fig. 11. The training loss is a metric used to assess the error on the training set and to find out how a deep learning model fits the training set of the dataset. Similarly, the validation loss is a metric used to assess the for the CNN model on the validation set of the dataset. The training and validation loss both are calculated from the sum of errors. The training loss depicts the performance of the model on training data and validation loss depicts the performance of the model on the

 https://doi.org/10.1007/s44196-025-00781-z

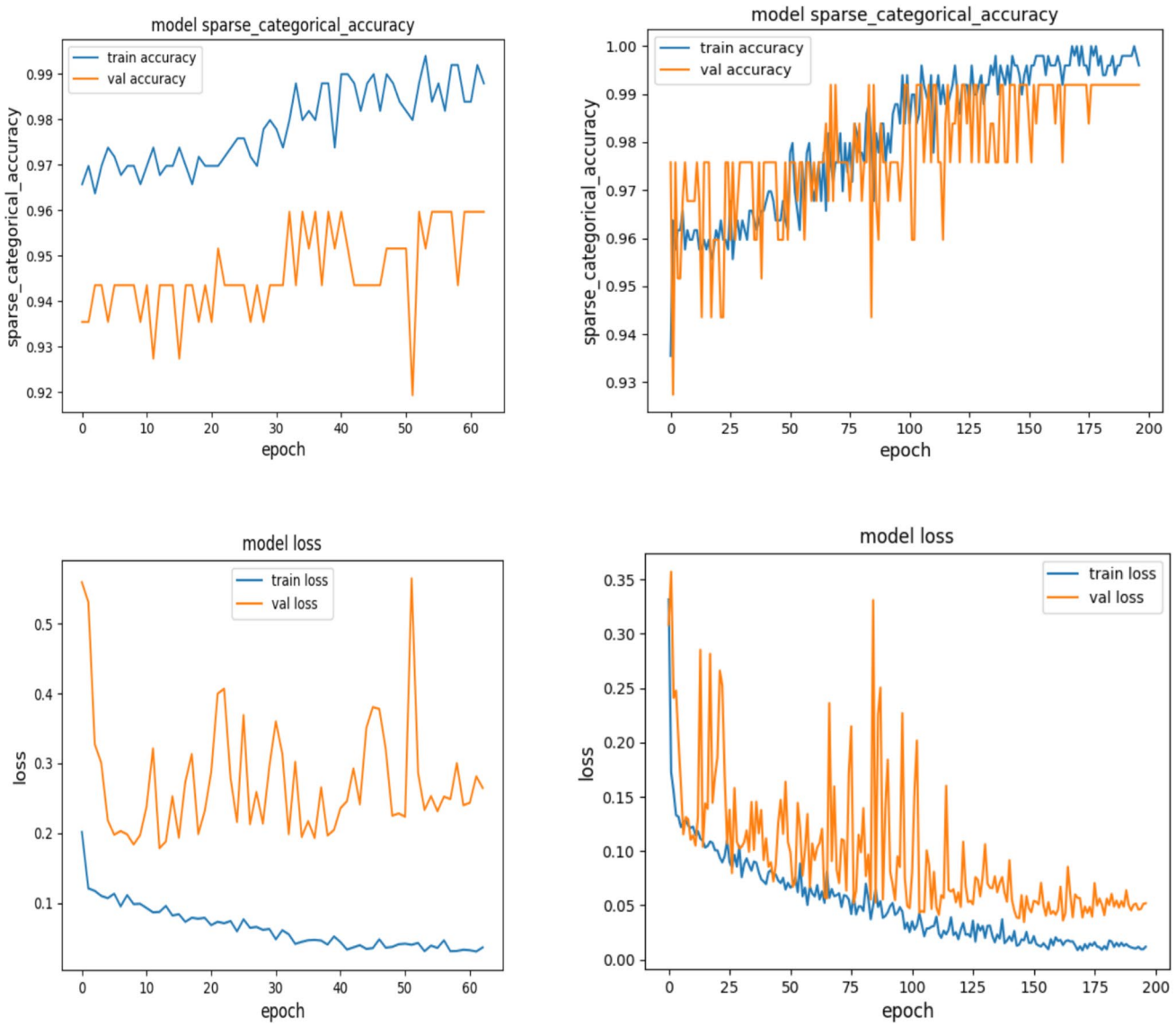

Fig. 12 CNN model showing accuracy and loss—RMSProp optimizer

test data. As observed the training and validation loss becomes stable and started reducing after epoch 80. This indicates an optimal fit and does not overfit or underfit.

The validation accuracy specifies how well the model is fitting for new or test data. As observed the validation accuracy reached 100% when the model's training accuracy reaches 99.8%.

4 Result analysis of Time Series System Call Data

The performance of the proposed system was evaluated based on detection accuracy. The equation to calculate the accuracy is shown in Eq. 1. Here, TP is True Positive, TN is True Negative, FP is False Positive and FN is False Negative. Two sets of experiments were carried out to evaluate the effectiveness of Time Series System Call data for the detection of intrusions. Experiment I (Sect. 3.2) tests the performance of Machine Learning algorithms on the proposed dataset. It is carried out using the WEKA tool with tenfold cross validation.

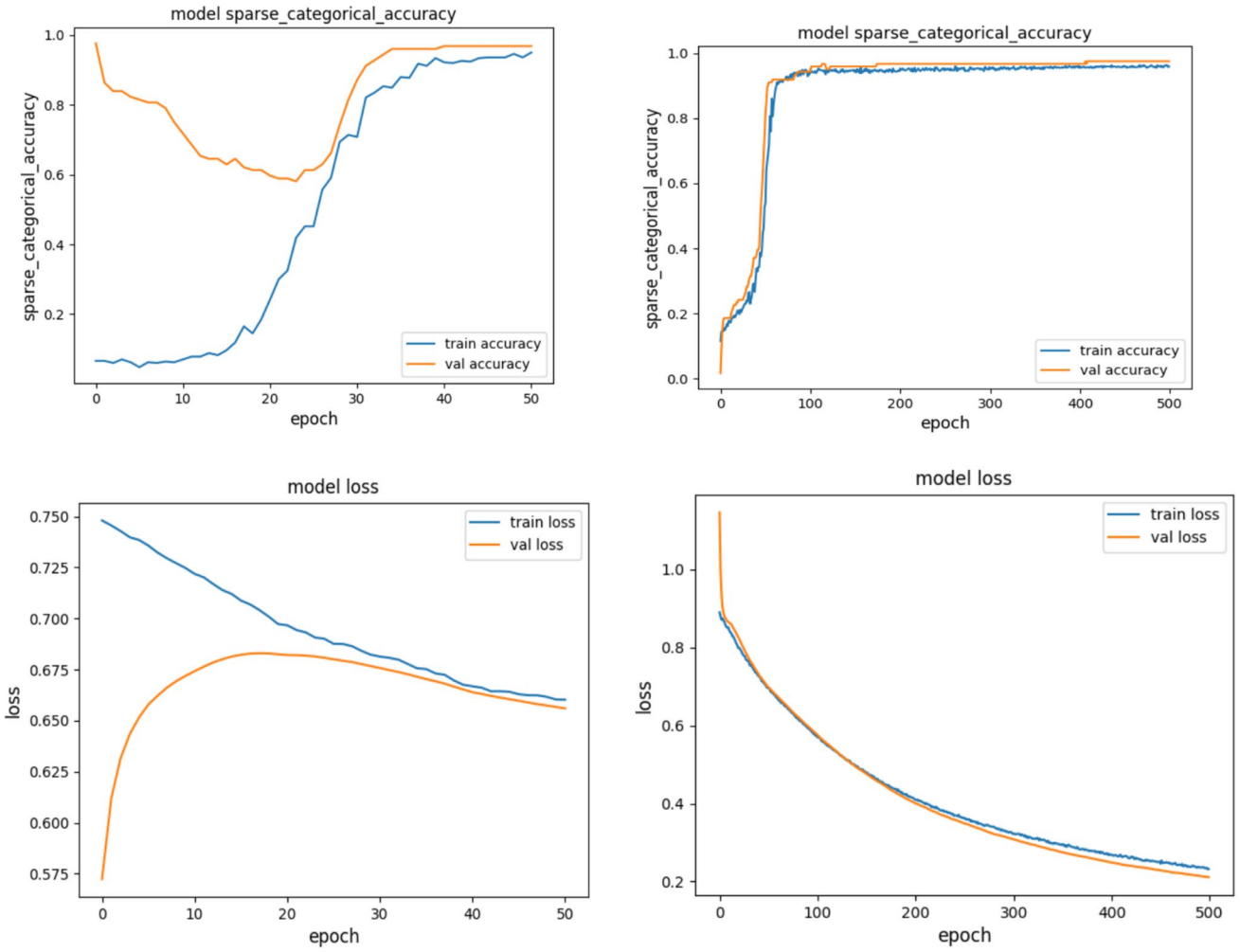

Fig. 13 CNN model showing accuracy and loss—AdaDelta optimizer

Experiment II (Sect. 3.3) is carried out using Python scripting with keras and TensorFlow framework. Both experiments are executed on the same host machine with similar hardware specifications. The aim of these experiments was to evaluate the effectiveness of the proposed Time Series System call data.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

During Experiment I, six popular machine-learning algorithms were used to measure the performance of the proposed dataset. The Random Forrest, Decision Tree and KNN algorithms perform well with higher detection accuracy compared to other variants (see Table 3). KNN has less false positive rate compared to other ML algorithms. The experiments are carried out with the default parameters and values defined in the WEKA tool. The results of experiment 1 arer displayed in Table 3.

Experiment II is carried out using Keras and TensorFlow framework to evaluate the effectiveness of CNN deep learning algorithms. As observed the detection accuracy of the CNN model reaches a detection accuracy of 99.8% which is 1.62% higher than J48, 1.72% higher than RF, 2.02% higher than RIPPER, 1.52% higher than KNN, 3.44% higher than SVM and 4.45% higher than Naive Bayes ML algorithms.

Table 7 showcases the intrusion detection accuracy of the proposed compared to previous implementations,

 https://doi.org/10.1007/s44196-025-00781-z

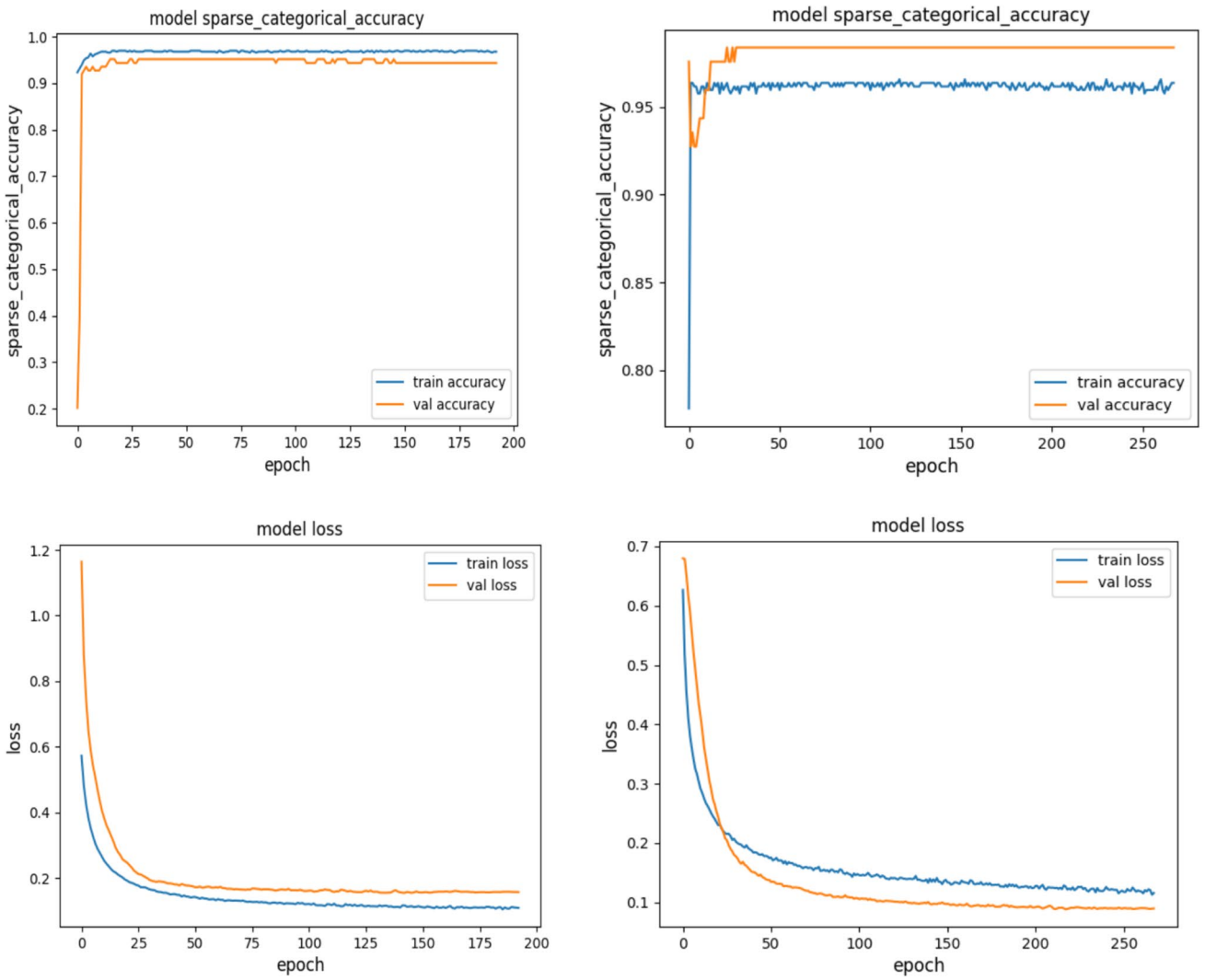

Fig. 14 CNN model showing Accuracy and Loss—AdaGrad optimizer

Table 6 CNN Model Detection Accuracies and Loss—Adam optimizer

#	Epochs	Training accuracy %	Validation accuracy %	Training loss	Validation loss	Test accuracy %	Test loss
1	97	98.79	94.35	0.0562	0.0357	96.21	0.1624
2	120	99.6	97.58	0.0237	0.1791	97.57	0.0907
3	150	98.99	99.19	0.0329	0.0894	98.10	0.1067
4	200	99.6	97.58	0.0258	0.2232	98.11	0.0938
5	350	99.8	100	0.0292	0.112	99.19	0.0852

Even though previous implementations provide good results in terms of intrusion detection accuracy, most of the system implementation make use of very old datasets – KDD98, DARPA 1998 and UNM datasets does not reflect any of the contemporary intrusions or malware attacks. Therefore these systems or models are not capable to detect current intrusions or malware programs. Patil's implementation makes use of static features to detect malware programs which is not suitable to handle the dynamic and hidden nature of malware programs. Therefore it can't detect the recent and zero-day malware programs. The first three implementations in Table 7 are targeted for cloud platform, the remaining are mainly done for traditional computing environment. Bhavesh et al. make

Table 7 Comparison of the proposed CNN model with previous implementation in-terms of detection accuracy

#	References	Dataset	Accuracy %
1	Mishra et al. (KVMInspector)		
		UNM CERT synthetic sendmail	94.7
		UNM mixed xlock	98.1
		UNM ps	97.5
		UNM live named	98
		UNM synthetic lpr	99
		UNM login	86.7
		UNM live inetd	97.6
		UNM ftp	93.8
2	Patil et al. (AMD framework)		
		Kaggle	99.8
		Malware classification	97.9
		ClaMP Integrated	98.2
		ClaMPRvaw	98.9
3	Bhavesh et al. (2019)	Anubis system call datasets (by Canali) with J48 (C4.5 decision tree) algorithm	99.4
4	Gideon et al. (ELM as HIDS DE)	KDD 98 & ADFA-LD dataset	100
5	Gupta (Immediate system call structure)		
		UNM Synthetic sendmail1	100
		UNM Synthetic sendmail2	98
		UNM CERT sendmail	100
		UNM Synthetic FTP	100
		UNM Synthetic IPR	4.66
		UNM inetd	100
		UNM ps	100
		UNM login	84
		UNM stide	59
		UNM live named	78
		UNM xlock	100
5	Yeung and Ding (dynamic & static behaviour models)		
	HMM and entropy analysis of system calls	UNM datasets	91.7
	Forrest et al. (Alternate Data Models) System call n-gram sliding window	UNM datasets	95.3 to 96.9
7	RBF ANN analysing system calls	UNM dataset	96
8	MLP ANN on subset of KDD98	KDD98 dataset	99.2
9	SVM on subset of KDD98	KDD98 dataset	99.6%
10	KNN with smooth Binary weighted RBF	DARPA 1998 database	96.3%
11	Rough set clustering	DARPA'98 BSM data	95.9
12	Proposed approach – CNN model	VMM Time series system call data set	100

use of Anubis systemcall dataset by Canali. Even though the first three implementations are targeted for cloud platform, the system call traces are traced from traditional host system. Not only that, all the implementations shown in Table 7 except the proposed method make use of datasets that are traced from traditional computing environments where the system calls are traced from typical agent programs. Procmon and strace are examples of agent programs. The making of KDD, UNM, DARPA uses some sort of traditional agent programs to get the system call traces.

The proposed CNN model make use of the VMM malware dataset (generated from the hypervisor layer) with system calls transformed into time series and without using any agent program. Since the dataset is traced from

Table 8 Comparison of Proposed approach with existing approaches

Parameter	Proposed CNN model	KVMInspector	Multi-threaded analysis
Technique	VMI with dynamic analysis	VMI with dynamic analysis	VMI with dynamic analysis
System Call Representation	Time Series	Bag of n-grams	1-g, 2-g, 3-g, 4-g, 5-g
Deployment	VMM	VMM	VMM
Machine Learning	CNN with 1d	Ensemble approach	C4.5, RIPPER, KNN, SVM, Naive Bayes
Accuracy	100%	81.25–99.92%	99.4%
Dataset	VMM malware dataset	UNM dataset	Anubis system call
Source of Dataset	From VMM/Hypervisor	From Unix Processes	Synthetic environment
Robustness	High	High	High

hypervisor it is best suitable to train a model or system for the hypervisor of the cloud environment. The proposed model delivers a high detection accuracy of 99.8% (training) and validation accuracy of 100% which is superior compared to other existing research works. The semantic approach with ELM by Gideon Creech claims a detection accuracy of 100% however the training overhead is much higher compared to other and proposed model.

Referring to Table 8, the proposed CNN model achieves the highest accuracy of 100% with the novel time series system call data traced from the hypervisor/VMM layer. Another highlight in the proposed system is the use of Convolutional Neural Network, a deep learning approach to train and validate the VMM time series system call dataset, whereas other implementations make use of traditional machine learning algorithms mostly.

5 Conclusion

This research work analyzed the effectiveness of convolution neural networks for the detection of intrusions by modelling the VMM malware dataset as time series of system calls. CNN model with VMM time series malware dataset has performed significantly well compared to traditional machine learning algorithms due to the fact that CNN have capability to extract high-level feature representations. The proposed CNN model with time series system calls has achieved a detection accuracy of 99.18% which is 1% higher than J48, 1.1% higher than random forest, 1.4% higher than RIPPER, 0.9% higher than KNN, 2.82% higher than SVM and 3.83% higher than naive bayes ML algorithms. The proposed approach provides advanced protection to virtual machines by incorporating VMI and deep learning techniques at the hypervisor layer of the cloud. The proposed CNN approach has achieved a validation accuracy of 100% which is 18.75% to 0.08% higher than KVMInspector and 0.6% higher than muti-threaded analysis with bag-of-ngrams. Since the proposed approach performs the security operations from outside the VM,.i.e., at the VMM layer, even the compromised VM and hidden malware can also be detected by the system. Therefore the system is robust. The system can be extended further by considering multiple Hosts and more than two VMs in the system. The proposed model can be integrated into Cloud security model by the service providers at the hypervisor layer as an antivirus service or Intrusion Detection service to detect the presence of malware or viruses without the need to install any additional program inside the server for protection. The main limitation is the computation and the volume of the VMM malware time series dataset required to train the proposed CNN model, however, this training time does not impact the performance of the IDS system since the training phase were done offline. The preprocessing required to transform the VMM system call traces into a time series introduces an extra clock into the IDS system. As a future direction, CNN models may be created with specific application types to secure the VM's more effectively. The proposed system can be remodeled with docker containers instead of virtual machines for running tenant applications.

Acknowledgements The authors thank their respective institutions for their support to carry out this research.

Author Contributions Conceptualization, A.A.R.M., and J.W.K.; Methodology J.W.K., and A.J.; Validation A.A.R.M., J.W.K., A.J., and D.C.; Formal analysis, A.A.R.M.; Writing – Original Draft Preparation, A.A.R.M., and J.W.K.; Writing – Review & Editing, A.J. and D.C.; Supervision, J.W.K.; All authors read and approved the final manuscript.

Funding Open access funding provided by Manipal Academy of Higher Education, Manipal. This research is carried out without any public/private/non-profit organization support.

Availability of Data and Materials The dataset used for this research is available in the Mendeley data portal in the following link https://data.mendeley.com/preview/jm44cs8ppf?a=41293174-be83-4a21-a560-2548328a90c1 [Accessed:18-Dec-24]. Researchers can get access to the dataset from the authors based on reasonable request.

Declarations

Conflict of interest The authors declare no competing interests.

References

1. [online]https://www.mcafee.com/enterprise/en-us/assets/reports/rp-cloud-adoption-and-risk-report-work-from-home-edition.pdf
2. [online]https://www.fortinet.com/content/dam/fortinet/assets/analyst-reports/report-2022-cloud-security.pdf
3. Melvin, A., Kathrine, G.J., Ilango, S., Shanmuganthan, V., Rho, S., Xiong, N., Nam, Y.: Dynamic malware attack dataset leveraging virtual machine monitor audit data for the detection of intrusions in cloud. Trans. Emerg. Telecommun. Technol. **33**, e4287 (2021). https://doi.org/10.1002/ett.4287
4. Hofmeyr, S.A., Forrest, S., Somayaji, A.: Intrusion detection using sequences of system calls. J. Comput. Secur. **6**, 151–180 (1998)
5. Gupta S, Kumar P (2014) An immediate system call sequence based approach for detecting malicious program executions in cloud environment. Springer Science+Business Media: New York, USA
6. Creech, G., Hu, J.: A semantic approach to host-based intrusion detection systems using contiguousanddis contiguous system call patterns. IEEE Trans. Comput. **63**, 807–819 (2014)
7. Garfinkel T, Rosenblum M (2003) A virtual machine introspection based architecture for intrusion detection. Netw. Distrib Syst. Secur. Symp
8. Mishra, P., Verma, I., Gupta, S.: KVMInspector: KVM based introspection approach to detect malware in cloud environment. J. Inf. Secur. Appl. **51**, 102460 (2020)
9. Aldribi, A., Traoré, I., Moa, B., Nwamuo, O.: Hypervisor-based cloud intrusion detection through online multivariate statistical change tracking. Comput. Secur. **88**, 101646 (2020). https://doi.org/10.1016/j.cose.2019.101646
10. Patil, R., Dudeja, H., Modi, C.N.: Designing in-VM-assisted lightweight agent-based malware detection framework for securing virtual machines in cloud computing. Int. J. Inf. Secur. **19**, 147–162 (2019)
11. Borisaniya, B., Patel, D.: Towards virtual machine introspection based security framework for cloud. Sådhanå **44**, 34 (2019). https://doi.org/10.1007/s12046-018-1016-6
12. Kumara, M.A., Jaidhar, C.D.: Leveraging virtual machine introspection with memory forensics to detect and characterize unknown malware using machine learning techniques at hypervisor. Digit. Investig. **23**, 99–123 (2017)
13. Mishra, P., Varadharajan, V., Pilli, E.S., Tupakula, U.: VMGuard: A VMIbased security architecture for intrusion detection in cloud environment. IEEE Trans. Cloud Comput. (2020). https://doi.org/10.1109/TCC.2018.2829202]
14. Varadharajan, V., Tupakula, U.K.: On the design and implementation of an integrated security architecture for cloud with improved resilience. IEEE Trans. Cloud Comput. **5**, 375–389 (2017)
15. Hwang, R.-H., Lee, C.-L., Lin, Y.-D., Lin, P.-C., Hsiao-Kuang, Wu., Yuan-Cheng, L., Chen, C.K.: Host-based intrusion detection with multi-datasource and deep learning. J. Inform. Secur. Appl. **78**, 103625 (2023)

16. Lin, Y.-D., Wang, Z.-Y., Lin, P.-C., Nguyen, V.-L., Hwang, R.-H., Lai, Y.-C.: Multi-datasource machine learning in intrusion detection: packet flows, system logs and host statistics. J. Inform. Secur. Appl. **68**, 103248 (2022). (**ISSN 2214-2126**)
17. Vinayakumar, R., Alazab, M., Soman, K.P., Poornachandran, P., Al-Nemrat, A., Venkatraman, S.: Deep learning approach for intelligent intrusion detection system. IEEE Access **7**, 41525–41550 (2019). https://doi.org/10.1109/ACCESS.2019.2895334
18. Chawla A, Lee BA, Fallon S, Jacob P (2018) Host Based Intrusion Detection System with Combined CNN/RNN Model. Nemesis/UrbReas/SoGood/IWAISe/GDM@PKDD/ECML
19. Hsiao S, Sun YS, Chen MC (2017) Virtual machine introspection based malware behavior profiling and family grouping. ArXiv, abs/1705.01697
20. Warrender CE, Forrest S, Pearlmutter BA (1999) Detecting intrusions using system calls: alternative data models. Proceedings of the 1999 IEEE Symposium on Security and Privacy (Cat. No.99CB36344), 133–145
21. Bridges, R.A., Glass-Vanderlan, T.R., Iannacone, M.D., Vincent, M.S., Chen, Q.: A survey of intrusion detection systems leveraging host data. ACM Comput. Surv. (CSUR) **52**, 1–35 (2018)
22. Joraviya, N., Gohil, B.N., Rao, U.P.: DL-HIDS: deep learning-based host intrusion detection system using system calls-to-image for containerized cloud environment. J. Supercomput. **80**, 12218–12246 (2024). https://doi.org/10.1007/s11227-024-05895-3
23. Silambarasan, E., Suryawanshi, R., Reshma, S.: Enhanced cloud security: a novel intrusion detection system using ARSO algorithm and Bi-LSTM classifier. Int. j. inf. tecnol. **16**, 3837–3845 (2024). https://doi.org/10.1007/s41870-024-01887-x

Authors and Affiliations

A. Alfred Raja Melvin[1] · Jaspher W. Kathrine[1] · Andrew Jeyabose[2,3] · D. Cenitta[2]

✉ Andrew Jeyabose
andrew.j@manipal.edu

✉ D. Cenitta
cenitta.d@manipal.edu

A. Alfred Raja Melvin
aarmelvin@gmail.com

Jaspher W. Kathrine
kathrine@karunya.edu

[1] Division of Computer Science and Engineering, Karunya Institute of Technology and Sciences, Coimbatore, India

[2] Department of Computer Science and Engineering, Manipal Institute of Technology, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India

[3] Department of Neurology, School of Medicine, University of North Carolina, Chapel Hill, NC, USA