

Investigating Linux Random Number
Generator for Virtualization Detection from
the Non Privileged User Space

A. A. D. Harshani

2025



Investigating Linux Random Number Generator for Virtualization Detection from the Non Privileged User Space

A. A. D. Harshani

Index No: 20000723

Supervisor: Mr. Kenneth Thilakarathna

May 2025

Submitted in partial fulfillment of the requirements of the
B.Sc. (Honours) in Computer Science Final Year Project



Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organizations.

Candidate Name : A. A. D. Harshani



Signature

26.06.2025

Date

This is to certify that this dissertation is based on the work of Ms. A. A. D. Harshani under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle Supervisor's Name : Mr. Kenneth Thilakarathna



Signature

27.06.2025

Date

Co-Supervisor's Name : Dr. H. N. D. Thilini



Signature

27.06.2025

Date

External Advisor's Name : Dr. Sachintha Pitigala

Abstract

The detection of virtualization presence is a critical problem in malware analysis, where malicious software may attempt to identify if it is being tested within a virtual environment. Existing methods often require special privileges, creating a gap for non privileged approaches. This study investigates the feasibility of detecting virtualization presence from the non privileged user space by analyzing the behaviour of the Linux Random Number Generator (LRNG), a component not previously used for this purpose.

A series of experiments were conducted across bare metal and virtual environments under varying impact levels to assess differences in random number generation rates and quality. The evaluation included both single and multiple VM setups, across desktop, private, and public cloud infrastructures. Results revealed measurable distinctions in LRNG behaviour between bare metal and virtual environments through distinct timing distributions, where early peaks were observed. These early peaks refer to instances where random number generation took significantly longer in virtual environments in the beginning compared to bare metal systems. Additionally, differences in dispersion patterns across bare metal and virtual environments were identified, which were collectively used for the detection of virtualization environments through derived thresholds, achieving a detection accuracy of up to 94.44%.

The study also examined the role of entropy enhancing tools designed to improve the randomness of generated data, in obscuring virtualization presence. The results proved approach is ineffective, suggesting the need for further research into obscure such detection. The influence of the operating system on LRNG behaviour was identified as a significant factor, with notable differences observed between Debian and Red Hat based Linux systems.

These findings demonstrate the potential of LRNG characteristics for non privileged virtualization detection with a novel direction. Unlike traditional detection methods that rely on privileged access, this approach operates entirely from the user space, demonstrating the feasibility of using user space behaviours to address virtualization detection challenges and opening possibilities for further research in this domain. Future work should focus on extending the scope of these findings, addressing the limitations identified, and exploring additional methods to enhance the robustness of detection and obfuscation techniques.

Acknowledgement

This research would not have been possible without the guidance and unwavering support of my supervisor, Mr. Kenneth Thilakarathna. His exceptional dedication, insightful contributions, and active involvement in every aspect of this study have been fundamental to its success. His support extended beyond the research, providing assistance with managing my academic responsibilities and emotional well-being, and it is through his efforts that this thesis has reached its completion.

I would also like to express my sincere gratitude to my co-supervisor, Dr. H. N. D. Thilini, and external advisor, Dr. Sachintha Pitigala, for their invaluable feedback, guidance, and expertise throughout the course of this research.

The support of the staff at the Network Operations Centre of the University of Colombo School of Computing was instrumental in providing the technical assistance necessary to complete this work. Their willingness to offer support and guidance throughout the study is greatly appreciated.

Conducting this research while balancing academic responsibilities was a challenging task, and I am deeply grateful to my supervisor and all the lecturers at the University of Colombo School of Computing for their understanding, encouragement, and support in making this possible.

I would also like to extend gratitude to my family for their ongoing support, and to my friends for their encouragement and help throughout this journey. Finally, I wish to express my appreciation to the University of Colombo School of Computing for providing the opportunity and resources to conduct this research.

Table of Contents

1	Introduction	1
1.1	Research Aims and Objectives	3
1.1.1	Aim	3
1.1.2	Objectives	3
1.2	Research Questions	3
1.3	Research Methodology	4
1.4	Research Scope	5
1.5	Thesis Structure	6
2	Literature Review	7
2.1	Virtualization	7
2.1.1	System Virtual Machines	7
2.1.2	Isolation between Host and Guest	10
2.2	Existing Approaches to Detect Virtualization Presence from within the Guest	11
2.3	Research Gap and Potential Common Parameters to Detect Virtualization	13
2.4	Behaviour of LRNG in Virtual Environments	14
2.5	Randomness	15
2.5.1	Random Number Generation	15
2.5.2	Measures of Randomness	17
2.6	Linux Random Number Generator	17
2.7	Identification of Research Gaps and Questions	19
3	Research Methodology	20
3.1	Experimental Design	20
3.1.1	Preliminary Experiments	20
3.1.2	Evaluating Preliminary Experiments	26
3.1.3	Experimental Approaches	27
3.1.4	Establishing Baselines and Impact Levels	28
3.2	Experimental Setup	30
3.2.1	Environments used for Experiments	30
3.2.2	Virtual Machine Monitors (VMMs) used for Experiments	30

3.2.3	Operating Systems used for Experiments	30
3.2.4	Hardware Specifications used for Experiments	31
3.3	Data Collection	33
3.3.1	System Preparation for Testing	33
3.3.2	Script Implementation and Interface Selection	33
3.3.3	Selection of Buffer Sizes	34
3.3.4	Sample Size and Repetition for Generalization	35
3.3.5	Initial Execution and Observed Network Effects	35
3.3.6	Data Collection with Entropy Enhancing Tools	37
3.4	Data Analysis Methods	37
3.4.1	Data Visualization	37
3.4.2	Basic Statistics	38
3.4.3	Error Metrics	39
3.4.4	Peak Detection Metrics	39
3.4.5	Assessing the Quality of Random Numbers	41
3.5	Summary of the Chapter	43
4	Results Evaluation and Discussion	44
4.1	Results of Research Question 1	44
4.1.1	Rate of Random Number Generation	44
4.1.2	Quality of Random Number Generation	64
4.1.3	Evaluation of RQ1 Results	66
4.2	Results of Research Question 2	70
4.2.1	Rate of Random Number Generation	71
4.2.2	Quality of Random Number Generation	88
4.2.3	Evaluation of RQ2 Results	89
4.3	Results of Research Question 3	92
4.3.1	Ubuntu 24.04 with kernel version 6.8	92
4.3.2	Ubuntu 22.04 with kernel version 5.15	96
4.3.3	AlmaLinux 9.4 with kernel version 5.14	98
4.4	Threshold Evaluation	103
4.4.1	Need for Threshold Merging	103
4.4.2	Proof of Concept Program and Threshold Evaluation	104

4.4.3	Final Detection Rules	105
4.5	Summary of the Chapter	105
5	Conclusion	107
5.1	Conclusion of the Study	107
5.2	Critical Reflection on Research Outcomes	108
5.3	Contributions	110
5.4	Limitations	111
5.5	Future Works	113
A	Appendix : Detailed Statistics for RQ 1 - Rate of Generation	119
B	Appendix : Detailed Statistics for RQ 1 - Quality of Generation	125
C	Appendix : Detailed Statistics for RQ 2 - Rate of Generation	129
D	Appendix : Detailed Statistics for RQ 2 - Quality of Generation	139

List of Figures

2.1	Paravirtualization, Binary Translation and H/W assisted Virtualization .	10
2.2	Comparison of bitmap images from PHP PRNG and true random sources	16
3.1	Overview of Experiments	20
3.2	Entropy starvation during GPG key generation on Ubuntu 14	23
3.3	Entropy Level fluctuation while generating GPG Keys	24
3.4	Entropy Level & System Parameter fluctuation while generating GPG Keys	25
3.5	Time taken to generate 6MB random numbers during SSH disconnection	36
4.1	Distribution of time taken to generate random numbers under high and low impact scenarios on Ubuntu 24.04 bare metal and VM environments	46
4.2	Distribution of time taken to generate random numbers under high and low impact scenarios on Ubuntu 22.04 bare metal and VM environments	48
4.3	Distribution of time taken to generate random numbers under high and low impact scenarios on AlmaLinux 9.4 bare metal and VM environments	52
4.4	Presence of early peaks in the distributions of virtual environments . . .	55
4.5	Dispersion of time taken to generate random numbers under high and low impact scenarios of Ubuntu 24.04 bare metal and VM environments . . .	59
4.6	Dispersion of time taken to generate random numbers under high and low impact scenarios of Ubuntu 22.04 bare metal and VM environments . . .	62
4.7	Distribution of time taken to generate random numbers on Ubuntu 24.04 multi-VMs	72
4.8	Distribution of time taken to generate random numbers on Ubuntu 24.04 cloud environments	73
4.9	Presence of early peaks in the distributions of desktop virtual environments	76
4.10	Absence of early peaks on Ubuntu 24.04 and 22 in the distributions of private and public cloud environments	78
4.11	Absence of early peaks on AlmaLinux 9.4 in the distributions of desktop and private cloud environments	79
4.12	Dispersion of time taken to generate random numbers on Ubuntu 24.04 multi-VM environments	81
4.13	Dispersion of time taken to generate random numbers on Ubuntu 24.04 private cloud environment	82

4.14	Dispersion of time taken to generate random numbers under high and low impact scenarios of Ubuntu 24.04 on public cloud environment	83
4.15	Dispersion of time taken to generate random numbers on Ubuntu 22.04 on multi-VM environments	84
4.16	Dispersion of time taken to generate random numbers on Ubuntu 22.04 private cloud environment	85
4.17	Dispersion of time taken to generate random numbers on Ubuntu 22.04 public cloud environment	86
4.18	A comparison of random number generation rate curves of Ubuntu 24.04 on Virtual Box across bare metal, with and without entropy enhancing tools	94
4.19	A comparison of random number generation rate curves of Ubuntu 24.04 on VMWare across bare metal, with and without entropy enhancing tools	95
4.20	A comparison of random number generation rate curves of Ubuntu 22.04 on QEMU across bare metal, with and without entropy enhancing tools .	99
4.21	A comparison of random number generation rate curves of Ubuntu 22.04 on VMWare across bare metal, with and without entropy enhancing tools	100
4.22	A comparison of random number generation rate curves of AlmaLinux 9.4 on Virtual Box across bare metal, with and without entropy enhancing tools	102

List of Tables

3.1	Listing entropy starvation activities in the order of the fluctuation they made on older kernels	26
3.2	Summary of Experimental Configurations across Research Questions . . .	33
3.3	Summary of Interfaces, Buffer Sizes and Impact Levels of Experiments .	36
4.1	Mean value differences of time taken on Ubuntu 24.04 environments . . .	49
4.2	Mean value differences of time taken on Ubuntu 22.04 environments . . .	50
4.3	Mean value differences of time taken on AlmaLinux 9.4 environments . .	50
4.4	Drop after the peak percentages across multiple OSs, VMMs, and buffer sizes	56
4.5	Variance ratio and SSE ratio of time taken to generate 2MB random numbers on Ubuntu 24.04	57
4.6	Mean skewness ratio of time taken on Ubuntu 22.04 from /dev/random interface	63
4.7	Percentage of samples passing all NIST tests on Ubuntu 22.04 /dev/random interface	64
4.8	Percentage of samples passing all NIST tests on Ubuntu 24.04 /dev/urandom interface split by peak presence in rate of generation	65
4.9	Mean value differences of time taken on Ubuntu 24.04 bare metal and multi-VM environments	71
4.10	Mean value differences of time taken on Ubuntu 22.04 bare metal and multi-VM environments	73
4.11	Mean value differences of time taken on AlmaLinux 9.4 bare metal and multi-VM environments	74
4.12	Drop after the peak percentages on Ubuntu 24.04 Desktop multi-VMs . .	77
4.13	Drop after the peak percentages on Ubuntu 22.04 Desktop multi-VMs . .	77
4.14	Drop after the peak percentages on AlmaLinux 9.4 Desktop multi-VM . .	77
4.15	Variance ratio and SSE ratio of time taken to 2MB generate random numbers on Ubuntu 24.04 in multi-VM setups	80
4.16	Variance ratio and SSE ratio of time taken to 2MB generate random numbers on Ubuntu 22.04 in multi-VM setups	87
4.17	Mean skewness ratio of time taken on Ubuntu 22.04 multi-VM	87

4.18	Percentage of samples that passed all NIST tests on Ubuntu 22.04 /dev/random interface in multi-VM desktop and cloud environments	88
4.19	Entropy Enhancement on Ubuntu 24.04 VMs only using Haveged	93
4.20	Entropy Enhancement on Ubuntu 24.04 VMs only using Jitterentropy . .	96
4.21	Combined Entropy Enhancement on Ubuntu 24.04 VMs using Haveged and Jitterentropy	96
4.22	Entropy Enhancement on Ubuntu 22.04 VMs only using Haveged	97
4.23	Entropy Enhancement on Ubuntu 22.04 VMs only using Jitterentropy . .	97
4.24	Combined Entropy Enhancement on Ubuntu 22.04 VMs using Haveged and Jitterentropy	98
4.25	Entropy Enhancement on AlmaLinux 9.4 VMs only using Haveged	101
4.26	Entropy Enhancement on AlmaLinux 9.4 VMs only using Jitterentropy .	101
4.27	Combined Entropy Enhancement on AlmaLinux 9.4 using Haveged and Jitterentropy	101
4.28	Evaluation results for virtualization detection	105
A.1	RQ1 Ubuntu 24.04 - /dev/random Detailed Statistics Sheet	119
A.2	RQ1 Ubuntu 24.04 - /dev/urandom Detailed Statistics Sheet	120
A.3	RQ1 Ubuntu 22.04 - /dev/random Detailed Statistics Sheet	121
A.4	RQ1 Ubuntu 22.04 - /dev/urandom Detailed Statistics Sheet	122
A.5	RQ1 AlmaLinux 9.4 - /dev/random Detailed Statistics Sheet	123
A.6	RQ1 AlmaLinux 9.4 - /dev/urandom Detailed Statistics Sheet	124
B.1	RQ1 - /dev/random - Percentage of samples that passed all NIST quality tests	125
B.2	RQ1 - /dev/urandom - Percentage of samples that passed all NIST quality tests	126
B.3	RQ1 - /dev/random - Percentage of samples that passed all NIST quality tests, split by peak presence	127
B.4	RQ1 - /dev/urandom - Percentage of samples that passed all NIST quality tests, split by peak presence	128
C.1	RQ2 - Desktop - Ubuntu 24.04 - /dev/random Detailed Statistics Sheet .	129
C.2	RQ2 - Desktop - Ubuntu 24.04 - /dev/urandom Detailed Statistics Sheet	130
C.3	RQ2 - Cloud - Ubuntu 24.04 - /dev/random Detailed Statistics Sheet . .	131

C.4	RQ2 - Cloud - Ubuntu 24.04 - /dev/urandom Detailed Statistics Sheet .	132
C.5	RQ2 - Desktop - Ubuntu 22.04 - /dev/random Detailed Statistics Sheet .	133
C.6	RQ2 - Desktop - Ubuntu 22.04 - /dev/urandom Detailed Statistics Sheet	134
C.7	RQ2 - Cloud - Ubuntu 22.04 - /dev/random Detailed Statistics Sheet . .	135
C.8	RQ2 - Cloud - Ubuntu 22.04 - /dev/urandom Detailed Statistics Sheet .	136
C.9	RQ2 - Desktop - AlmaLinux 9.4 - /dev/random Detailed Statistics Sheet	137
C.10	RQ2 - Desktop - AlmaLinux 9.4 - /dev/urandom Detailed Statistics Sheet	138
D.1	RQ2 - Desktop - /dev/random - Percentage of samples that passed all NIST quality tests	139
D.2	RQ2 - Desktop - /dev/urandom - Percentage of samples that passed all NIST quality tests	140
D.3	RQ2 - Cloud - /dev/random - Percentage of samples that passed all NIST quality tests	141
D.4	RQ2 - Cloud - /dev/urandom - Percentage of samples that passed all NIST quality tests	142
D.5	RQ2 - Desktop - /dev/random - Percentage of samples that passed all NIST quality tests, split by peak presence	143
D.6	RQ2 - Desktop - /dev/urandom - Percentage of samples that passed all NIST quality tests, split by peak presence	144
D.7	RQ2 - Cloud - /dev/random - Percentage of samples that passed all NIST quality tests, split by peak presence	145
D.8	RQ2 - Cloud - /dev/urandom - Percentage of samples that passed all NIST quality tests, split by peak presence	146

List of Acronyms

ISA	Instruction Set Architecture
LRNG	Linux Random Number Generator
MSE	Mean Squared Error
OS	Operating System
PoC	Proof of Concept
PRNG	Pseudo Random Number Generator
RNG	Random Number Generator
SSE	Sum of Squared Errors
TRNG	True Random Number Generator
VM	Virtual Machine
VMM	Virtual Machine Monitor

1 Introduction

Virtualization has become an essential part of modern computing, enabling better resource utilization and isolation. They are widely used in various applications, such as cloud computing, software development, and malware analysis. In the context of malware analysis, Virtual Machine (VM)s are commonly used to safely execute and study malicious software, as any damage caused remains contained within the VM and can be easily discarded without affecting the host system. However, if the malware is capable of detecting that it is running within a virtualized environment, it may alter its behavior or remain dormant, thereby evading detection. This detection ability is a significant challenge for security researchers and it creates a need for effective methods for both detecting virtualization and obscuring VM presence. This study will be primarily focusing on the ability of an application executed within a VM to detect whether it is running inside a VM.

Upon reviewing the existing literature on virtualization detection, it became apparent that even though it is theoretically not possible under modern hardware assisted virtualization techniques, certain loopholes still exist for virtualization detection. However, most of the identified methods rely on elevated privileges such as the execution of privileged instructions or root access, limiting their applicability in scenarios where such access is restricted. In contrast, this research aims to explore a non privileged approach that operates entirely from the user space, without the need for privileged access.

While exploring potential measures that can be utilized from the user space for detecting virtualization, the study identified that the Linux Random Number Generator (LRNG) exhibits different behaviours in VMs compared to bare metal systems. The LRNG is a key component essentially present in every Linux system, responsible for generating random numbers for various system needs. According to literature, it shows distinct variations in parameters such as the rate of random number generation and entropy values when running in virtualized environments. Further, the LRNG has been subjected to various updates over the recent years, which lack references in formal research literature. However, to the best of the author's knowledge, no research has been conducted on the potential of using behaviours of LRNG in virtual environments as a mean of detecting virtual environment presence.

The primary aim of this research is to investigate how the behaviour of the LRNG can be used to detect the presence of virtual environments without requiring privileged access or specific system configurations. The study further explores how these detection capabilities could be extended to environments involving multiple VMs operating on a single host. Furthermore, the study identified that direct entropy level based measures to identify LRNG behaviour are no longer applicable in modern kernel versions due to the evolution of the LRNG. This finding lead to the exploration of alternative indirect measures such as time taken to generate random numbers and quality of generated random numbers, investigating the feasibility of using user space behaviours to detect virtualization. Notably, the findings show a detection accuracy of up to 94.44% in single VM setups, demonstrating the potential of this approach.

Additionally, the study examined methods to obscure the presence of virtual machines, based on the assumption that differences in entropy between virtualized and bare metal environments contribute to the observed detection capability. Experiments were repeated under the activation of commonly available entropy enhancing tools and found that these tools are ineffective in concealing the detection of virtualization. This finding highlights a significant avenue for future research, particularly in exploring direct entropy measurement techniques, evaluating the effectiveness of entropy enhancement mechanisms, and identifying other potential factors that may influence virtualization detection.

This research presents a novel approach to detecting virtualization from the user space, an area previously underexplored in existing literature. The findings highlight the importance of considering user space behaviour in strengthening security measures for malware analysis. The study also provides a foundation for future research to improve detection accuracy, explore new obfuscation methods, and expand the approach to more complex virtualization configurations as well.

1.1 Research Aims and Objectives

1.1.1 Aim

To investigate methods for detecting virtualization presence using the Linux Random Number Generator and related parameters in virtual environments, and to explore techniques to obscure such detection from user space programs.

1.1.2 Objectives

1. To analyze how a program running in the user space of a Linux VM can detect the underlying virtualization platform using the LRNG and related parameters.
2. To evaluate whether this detection capability can be extended to environments with multiple guest VMs on a single host.
3. To identify and propose modifications to the LRNG and related parameters to obscure the presence of virtualization from user space programs running within the VM.

1.2 Research Questions

1. How can a program running in the user space of a Linux VM detect the underlying virtualization platform using Linux Random Number Generator and related parameters?
2. Can this detection capability be extended to an environment with multiple guest VMs on a single host?
3. How to obscure the VM presence from user space programs running within the VM?

1.3 Research Methodology

This study follows a deductive research approach, beginning with the synthesis of a research gap based on insights gathered from existing literature. Specifically, two key observations were identified from previous works,

- A significant challenge exists in malware analysis due to malicious software detecting virtualized environments and altering its behaviour. Existing detection mechanisms require elevated privileges hence the detection can be prevented by eliminating required access levels. However, what if a program can detect VMs from the user space without any special privileges?
- The LRNG, which is accessible to operate from the user space, exhibits behavioural differences between virtualized and bare metal environments

By integrating these two findings, this research identified a novel gap, the potential to use LRNG behaviour from user space as a technique to detect virtualization, which had not been previously explored. Based on this gap, it was assumed that the behaviour of the updated LRNG could be utilized to detect virtualized environments without privileged access.

The initial phase of the study involved an exploration of potential entropy based measures available at the user space level. During this process, it was discovered that direct measurements of entropy related parameters were no longer showing fluctuations in newer Linux kernel versions due to architectural updates of the LRNG. This limitation guided a shift toward indirect approaches for assessing LRNG behaviour.

Addressing that, an experimental framework was developed to measure the rate and quality of random number generation under different levels of system activity. A systematic evaluation of possible impact activities was carried out, focusing on depleting the entropy pool. Preliminary experiments were conducted on an older Linux version where entropy fluctuations could be directly observed. This allowed the identification of activities that had the most consistent and significant effects on LRNG behaviour. The selected activities were then incorporated into the final experimental design, which was applied across a range of VM and bare metal instances.

A further challenge was the comparison between bare metal and VMs. Due to the variability across hardware platforms and Virtual Machine Monitor (VMM)s, direct com-

parisons were not feasible. To address this, a delta based evaluation approach was introduced, focusing on the relative change in LRNG behaviour under controlled activity conditions.

Building on this approach, the collected data from both bare metal and VMs were analyzed to identify criteria that consistently showed differences in LRNG behaviour between the two environments. Based on this analysis, specific thresholds were developed to distinguish VMs and bare metal systems. These identified thresholds were then evaluated to determine the most effective combinations for reliable detection. The study further extended the evaluation to multiple VM environments, including private and public cloud platforms, and briefly explored approaches to obscure the presence of virtualization from user space programs. A detailed description of the experimental design, data collection procedures, activity selection process, and evaluation strategies is presented in Chapter 3.

1.4 Research Scope

- Developing programs that runs in the non privileged user space of a Linux environment (x86 architecture) to collect the rate of random number generation and associated entropy values
- Collecting data related to LRNG and entropy from linux VMs of different virtualization configurations
- Analyze the behaviour of the latest version of LRNG and entropy values in virtualized Linux environments using collected data
- Comparing data collected from Linux VMs with data from non virtualized Linux systems to identify where there are significant differences indicative of virtualization
- Evaluating the effectiveness of the finding in an environment with multiple guest VMs on a single host, assessing the consistency of entropy behaviour across different VMs
- Identifying potential changes or mechanisms that can be used to obscure the VM detection from the user space

1.5 Thesis Structure

The remaining sections of the thesis are organized as follows,

Chapter 2 : Literature Review, provides a detailed review of the background concepts and existing literature on virtualization, existing approaches to virtualization detection, potential parameters for detecting virtualization, role of random numbers and the internal workings of LRNG.

Chapter 3 : Research Methodology, outlines the research design, detailing the experimental setup and methodologies employed throughout the study. It discusses the preliminary experiments, approaches used to establish baselines and impact levels, and the selection of experimental environments, and configurations. It further covers the data collection process, from system preparation and script implementation to the selection of buffer sizes, sample sizes, and repetitions for generalization. This chapter also includes a discussion on methods used for data analysis such as data visualization, basic statistics, error metrics, peak detection, and quality assessment of random number generation.

Chapter 4 : Results Evaluation and Discussion, is structured according to the research questions. For each research question, experiments related to the rate and quality of random number generation are analyzed using visual graphs and statistical metrics. The evaluation includes threshold detection for virtualization in both Research Question 1 and Research Question 2, followed by an investigation into methods for obscuring VM presence in Research Question 3. The results are categorized according to observations made in different Operating System (OS)s. The chapter concludes with a threshold evaluation section, which discusses the need for threshold merging, proof of concept program, threshold evaluation and final detection rules along with their accuracy, precision, and other relevant evaluation metrics.

Chapter 5 : Conclusion, summarizes the findings of the study, critically reflects on the research outcomes, highlights the contributions made, discusses the limitations of the study, and suggests directions for future research.

2 Literature Review

2.1 Virtualization

Virtualization refers to the creation of a virtual version of computing resources such as hardware platforms, storage devices, and network resources on top of another computing system. It enables multiple OSs to run concurrently on a single physical machine by abstracting hardware resources. According to the formal definition of virtualization, presented by Popek and Goldberg (1974), virtualization is creating an isomorphism of a host machine into a virtual guest system. Virtualization can be applied to an entire machine, as well as for various subsystems of a host machine such as the processor, disk, memory, I/O devices etc (Smith and Nair 2005).

Virtualization can be mainly categorised into two types, process virtualization and system virtualization. The Instruction Set Architecture (ISA) and other architectural interfaces play a major role in this classification. In process VMs, the virtualization happens at the Application Binary Interface (ABI) which invokes the OS using the system calls interface and communicates with hardware components using the user instructions interface. This virtualization emulates both of the system calls and user instructions interfaces. Therefore, a process VM sees a combination of the OS and underlying hardware as the ‘machine’. In system VMs, the virtualization happens at the ISA interface which handles the communication between hardware and software, including the OS. The virtualization software used in system VMs is commonly known as VMM. The proposed study primarily focuses on system VMs, hence classifications and methods of virtualization used in system VMs will be explored further in the following subsection (Smith and Nair 2005).

2.1.1 System Virtual Machines

A system VM supports the virtualization of an entire computing system with all functionalities of an OS. The VMM that is used to emulate the ISA translates the ISA used in the hardware components of the host system to another ISA, creating a virtualization layer that can run an OS that is developed for a different ISA from the host machine (Smith and Nair 2005).

In the context of x86 architecture, which is commonly employed in modern virtual-

ization environments, virtualization techniques rely on manipulating privilege levels or "ring levels" to manage access to system resources. The x86 architecture provides multiple privilege levels, ranging from Ring 0, which is the highest where the OS kernel operates, to Ring 3 which is the lowest where user applications run. (Intel 2023)

According to the classic approach of system VMs introduced by Popek and Goldberg (1974), the VMM executes directly on hardware of the host machine with the highest privilege level and the guest OS runs on top of that with a lesser privilege level. These VMMs are also known as 'Type I Hypervisors'. The guest OS, while running in a virtual environment, is unaware that it is running on a VM. This aligns with the Popek and Goldberg criteria, which asserts that a guest OS running on a VM should not be able to know that it is running in a virtualized environment (Popek and Goldberg 1974). However, methods to detect VM presence, as explored in this study, challenge this criterion, revealing potential violations of the principle.

There is another approach to system VMs, known as 'Hosted VM', as well as 'Type II Hypervisor'. In hosted VMs, the VMM is positioned upon the OS of the host machine enabling it to get support from the host OS for the execution of privileged instructions (Tanenbaum and Bos 2014).

When a VM operates, it directly accesses the host hardware for non privileged instructions (Ring 3). However, for privileged operations, such as Ring 0 instructions, the VM requires intervention from the VMM. The VMM ensures that privileged instructions are handled correctly by trapping them or using other mechanisms to ensure proper execution without violating isolation between the guest OS and the host machine (Barham et al. 2003; Goldberg 1974; Smith and Nair 2005). This interaction is crucial for maintaining the separation between virtual environments and the underlying physical hardware. The VMM is responsible for intercepting any privileged operations and executing them in the context of the host machine (Tanenbaum and Bos 2014).

The handling of privileged and non privileged instructions in virtualized environments involves a variety of virtualization techniques. These include trap and emulate (Goldberg 1974), paravirtualization (Barham et al. 2003), and binary translation (Adams and Agesen 2006) which enable the execution of VMs by using different strategies for handling privileged instructions and managing interactions between VMs and the host system (Figure 2.1).

The trap and emulate technique is one of the earliest methods used for virtualization. It involves the VMM intercepting privileged instructions executed by the guest OS, which typically would require direct access to hardware. Whenever a privileged instruction is invoked by the guest OS, a trap is triggered, and control is transferred to the VMM. The VMM then emulates the instruction and ensures that the execution proceeds correctly (Popek and Goldberg 1974).

However, this method faces a critical limitation with sensitive instructions in the x86 architecture. Sensitive instructions are non privileged instructions that exhibit different behaviours when executed in different privilege levels (Ring 0 and Ring 3). Certain instructions that would behave differently in a privileged environment might produce incorrect results when executed in the VM because the VMM cannot trap or emulate them accurately. (Popek and Goldberg 1974).

To address the limitations of trap and emulate, the approach of paravirtualization has been introduced by the study of Barham et al. (2003). In this technique, the guest OS is modified to be aware of the virtualized environment. This modifies the guest OS and replace sensitive instructions with hypercalls. A hypercall is a special instruction that directly calls the VMM to perform the privileged operation.

While paravirtualization can overcome the trap and emulate limitations, it requires modification of the guest OS, which is not always feasible, especially for proprietary OSs. Additionally, since the guest OS is aware of the virtualization, it violates the Popek and Goldberg criterion, which states that a VM should not be aware that it is virtualized (Popek and Goldberg 1974).

An alternative to paravirtualization is binary translation which involves the VMM operating as a compiler. Instead of modifying the guest OS itself, the VMM dynamically translates sensitive instructions into safe, executable code that can run correctly within the virtualized environment. Although binary translation avoids the limitations of trap and emulate and paravirtualization, it still has some weaknesses. Specifically, the guest OS might detect virtualization through subtle indicators, such as the through segmentation registers which contains the Ring level it operates on. (Adams and Agesen 2006).

The most recent advancement in virtualization is hardware assisted virtualization, which uses hardware support for more efficient and transparent virtualization. With

hardware assisted virtualization, the CPU itself provides special modes, such as the VMX mode in Intel processors, to support VMs. In this mode, the VMM has direct access to the hardware resources, and the guest OS operates in a special pseudo Ring 0, which behaves like Ring 0 but is controlled by the hypervisor. This method makes sure that the guest OS cannot detect that it is running inside a virtual environment, since the hardware handles the isolation and access control. This method addresses all the issues found in previous approaches, providing a fully transparent virtualization layer where the guest OS remains unaware of the virtualization (Adams and Agesen 2006; Pék, Buttyán, and Bencsáth 2013).

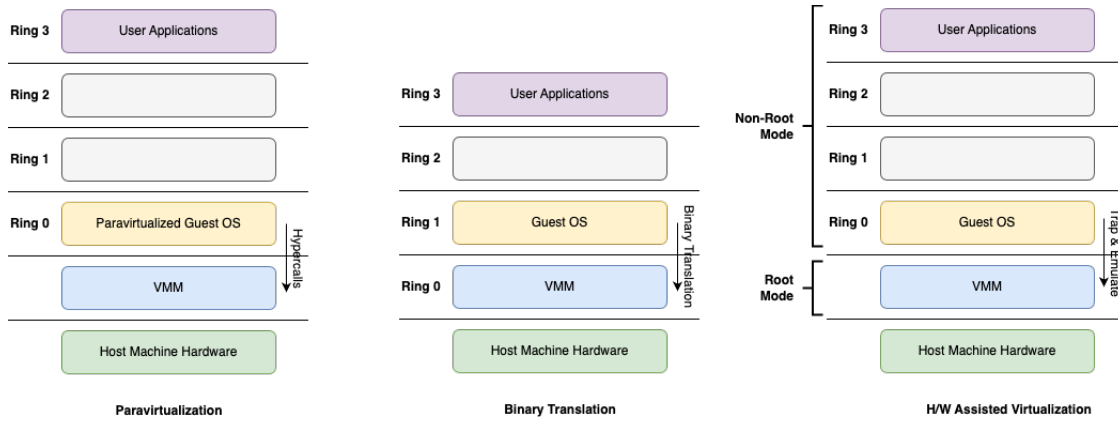


Figure 2.1: Paravirtualization, Binary Translation and H/W assisted Virtualization

2.1.2 Isolation between Host and Guest

Isolation between the host and guest systems is a fundamental requirement for virtualization. It makes sure that the operations of a guest system do not affect the host or other guests, maintaining a secure and stable environment (Popek and Goldberg 1974).

Isolation is important in environments where multiple VMs are used, such as in cloud computing and data centers. In these settings, different VMs often include applications and data for different clients, and isolation ensures that the actions of one VM do not interfere with another. This is essential for maintaining data privacy and security. Also, isolation helps in improving the reliability and stability of systems. If one VM fails, the isolation ensures that the host and other VMs are not affected by that failure. Proper isolation also supports resource management by preventing one VM from consuming excessive resources and affecting the performance of others. (Jithin and Chandran 2014; Smith and Nair 2005; Raffetseder, Kruegel, and Kirda 2007).

Additionally, isolation allows to safely analyze potentially harmful software without risking the security of the host machine, preventing malicious activities in VMs from impacting the entire system. Modern malicious programs often include checks to determine if it is running inside a virtual environment or actual hardware. If the malicious program detects that it is in a VM, it may alter the behaviour or become dormant to avoid detection. This makes it harder to analyze the true behaviour of malicious programs and develop effective countermeasures (Jithin and Chandran 2014; Lusky and Mendelson 2021).

2.2 Existing Approaches to Detect Virtualization Presence from within the Guest

Malicious software has developed various methods to detect whether it is running inside a virtualized environment, with the primary goal of avoiding detection during malware testing. These detection techniques use specific hardware characteristics, timing anomalies, and certain software properties that can reveal the presence of a VM or a sandbox. These techniques are often used to make it harder for security researchers to analyze and detect the malicious behaviour of the software.

One widely used approach is timing analysis, which involves measuring the time taken for specific operations. In a virtualized environment, the presence of the VMM can introduce delays due to its involvement in handling instructions. These delays, often referred to as “hypervisor-induced latency,” can lead to irregular timing patterns that would not typically occur on physical hardware. If the observed timing behaviour deviates significantly from what is expected on a real machine, it can serve as an indication that the software is running within a VM (Raffetseder, Kruegel, and Kirda 2007; Lusky and Mendelson 2021; Chen et al. 2016; Sun et al. 2011; Bulazel and Yener 2017).

Another technique is based on examining CPU performance counters, which track various hardware events such as cache hits, instruction counts, and branch predictions. These counters can reveal differences between the performance characteristics of a physical machine and a virtual environment. VMs can show abnormal or inconsistent performance patterns in these counters, due to their dependence on the VMM as the VMM manages resources differently compared to a physical system. Malware can use these differences to decide whether it is running inside a VM (Jithin and Chandran 2014; Lusky

and Mendelson 2021; Chen et al. 2016; Bulazel and Yener 2017).

Network packet timing is another area of focus for virtualization detection. Virtual environments can introduce slight delays in network packet transmission due to the overhead of the virtualization layer, which handles network communication differently from physical hardware. Malware can analyze the round trip times of network packets and compare these values with expected benchmarks for physical machines. If the timing deviates from those, it can indicate the presence of a virtualized network stack, allowing the malware to conclude that it is running in a VM (Bulazel and Yener 2017; Jithin and Chandran 2014; Raffetseder, Kruegel, and Kirda 2007).

In addition to above methods, malware may scan for specific indicators left by VMMs. These artifacts can include device drivers, registry keys, or files that are unique to the virtualization environment. The existence of such indicators can serve as a clear sign that the system is running within a VM. Malware may also examine system level behaviours, such as the CPUID instruction, which can return specific values that signal the presence of a VMM. These unique VMM behaviours, along with certain I/O port interactions, can further support the detection of a VM (Lusky and Mendelson 2021; Pék, Buttyán, and Bencsáth 2013; Sun et al. 2011).

Furthermore, malware can assess the resource availability on the system. VMs often have limited resources compared to physical hardware, especially when running in environments with resource allocation constraints. If a system displays unusual patterns, such as unexpected resource limitations or mismatches in available hardware resources, this can signal that the environment is virtualized. The guest system may have access to fewer resources than a physical machine would provide, which is an indicator of virtualization (Lusky and Mendelson 2021; Zhang et al. 2021).

Lastly, there has been increasing interest in the use of machine learning techniques for detecting virtualization. Supervised learning approaches can be used to train models on various features such as performance metrics, CPU behaviour, and hardware characteristics. By using these features, machine learning models can differentiate between virtual and non virtual environments with greater accuracy, often outperforming traditional methods. These models can be trained to identify subtle differences between physical and virtual systems, enhancing the ability to detect virtualization presence from within the guest (Lusky and Mendelson 2021; Yokoyama et al. 2016).

2.3 Research Gap and Potential Common Parameters to Detect Virtualization

All existing methods for detecting virtualization presence require privileged access to the system, either kernel privileges or root user access. Such techniques are typically out of reach for non privileged user space processes, hence the detection capability can be masked by restricting the required privileges. However, previous studies have not focused on mechanisms to detect the underlying virtualization platform without relying on any special privileges. This limitation has motivated the focus of this project, to explore the potential of detecting virtualization presence from within the non privileged user space, where access to system level information is restricted.

In the search for parameters that are commonly available in Linux systems and do not require elevated privileges, it was observed that one of the key indicators of virtualization presence could be in the behaviour of the random number generation process of the system. It was identified that random number generation exhibits distinct differences in virtualized environments compared to physical hardware (Kumari, Alimomeni, and Safavi-Naini 2015; Cieslarová 2018; Everspaugh et al. 2014).

On VMs, specialized tools such as additional rng-tools (RedHat 2018; QEMU 2016; Intel 2018) and daemons like `haveged` (Wuertz and Hladky 2025) are often used to enhance the quality and availability of entropy, particularly during operations that require high levels of randomness, such as cryptographic key generation. These tools aim to avoid the reduced entropy states that are often experienced in virtualized environments, ensuring that applications requiring strong randomness are not blocked by entropy starvation. However, despite these efforts to improve randomness, the underlying differences in the entropy sources between a physical machine and a virtualized system remain evident.

The behaviour of the LRNG in these environments serves as a potential technique for detecting virtualization. While physical machines typically have a more reliable and stable source of randomness from hardware based events, VMs rely on the virtualized environment for entropy generation, which may lead to variations in randomness. These variations are often a result of factors such as the limited access to hardware resources, the influence of VMM scheduling, and the need to use software based entropy sources. As a result, the performance of LRNG and entropy generation behaviours are often distinguishable between physical systems and VMs.

Hence, the LRNG provides a novel direction for detecting virtualization presence from within the non privileged user space. The observed differences in LRNG characteristics between physical and virtual environments may serve as indicators of virtualization, enabling non privileged applications to decide whether they are running in a VM. This technique is especially valuable in scenarios where more invasive detection methods are not feasible or desirable.

2.4 Behaviour of LRNG in Virtual Environments

Using the LRNG in VMs has been a challenging task, primarily because virtual environments often lack the diverse range of hardware events that are used to gather entropy. On physical machines, the LRNG collects entropy from sources such as device interrupts, hardware timings, and user interactions. However, in VMs, the virtualization layer abstracts much of the underlying hardware, limiting the availability and diversity of these entropy sources. This limitation has resulted in lower entropy availability within VMs, making it more difficult to generate secure and high quality random numbers during critical system operations such as cryptographic key generation (Kumari, Alimomeni, and Safavi-Naini 2015).

Furthermore, an additional security concern arises with the use of VM snapshots. When a snapshot is taken, the complete state of the VM, including the internal state of the Random Number Generator (RNG), can be captured. If a snapshot is restored and reused without properly reinitializing the LRNG state, it can lead to the generation of predictable random numbers. This compromises the security of cryptographic operations and other processes that depend on high quality randomness, as attackers could potentially predict future outputs based on the captured state (Everspaugh et al. 2014). These issues highlight the unique challenges associated with ensuring secure random number generation in virtualized environments.

Although several studies have explored the general behaviour and security implications of the LRNG in VMs (Kumari, Alimomeni, and Safavi-Naini 2015; Everspaugh et al. 2014), the specific use of LRNG behaviour as a technique for detecting the presence of virtualization has not been considered. The inherent differences in entropy collection and randomness generation between physical systems and virtual environments present a potential method for distinguishing between them from within the VM user space.

Additionally, while significant changes have been introduced to the LRNG in recent years to enhance its performance and security, particularly in newer Linux kernel versions, these developments have not been thoroughly examined with respect to their implications for virtualization detection. This gap presents an opportunity for further investigation into the relationship between LRNG behaviour and virtual environment characteristics.

2.5 Randomness

Randomness plays a vital role in modern computer security, especially in generating cryptographic keys, initialization vectors, session tokens, and password salts. The criticality of high quality randomness lies in its ability to provide unpredictability. This unpredictability is fundamental to resisting attacks that attempt to guess or replicate security parameters. A key security principle emphasizes that the strength of a cryptographic system should not depend on the secrecy of the algorithms used but solely on the secrecy of the key (Shannon 1949). Consequently, the security of cryptographic operations fundamentally relies on the strength of the randomness used to generate these keys.

2.5.1 Random Number Generation

There are two primary approaches for random number generation, True Random Number Generator (TRNG)s and Pseudo Random Number Generator (PRNG)s.

True random number generators derive their randomness from naturally unpredictable physical phenomena such as electronic noise, radioactive decay, and atmospheric noise. Because these sources originate from physical processes, TRNGs typically offer high entropy outputs suitable for critical security applications. Real world examples of TRNGs include Cloudflare’s “Wall of Entropy,” where lava lamps are used to generate randomness through video capture (Dillon 2017) and Random.org’s atmospheric noise based random number service (Random.org 2025).

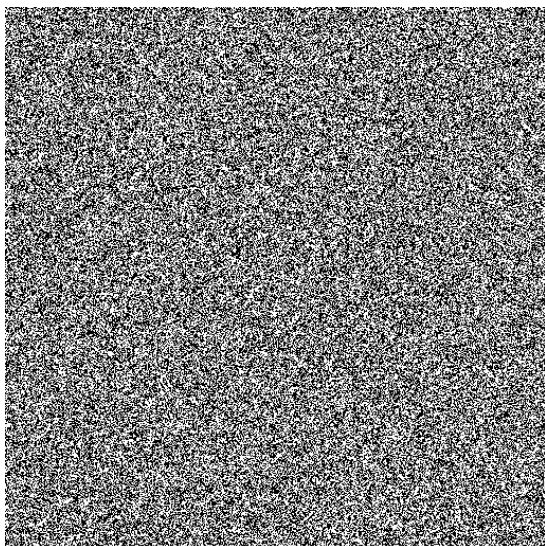
In contrast, PRNGs generate sequences of numbers through deterministic algorithms based on an initial seed value. Although computationally efficient, PRNGs inherently suffer from predictability if the seed is known. One basic PRNG model is the Linear Congruential Generator (LCG), defined by the recursive relation,

$$X_{n+1} = (aX_n + c) \mod m$$

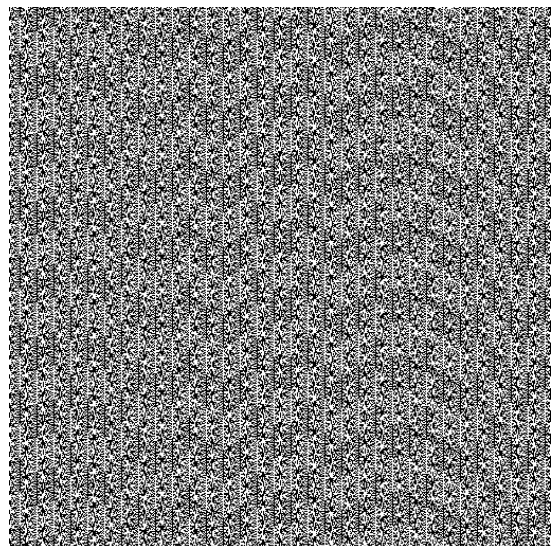
where X represents the sequence of pseudo random values, m is the modulus ($m > 0$), a is the multiplier ($0 < a < m$), c is the increment ($0 \leq c < m$), and X_0 is the seed or starting value ($0 \leq X_0 < m$).

Beyond the LCG, a wide range of mathematical techniques exists to generate pseudo random numbers. A particularly important subset is the Cryptographically Secure Pseudo Random Number Generator (CSPRNG), designed such that even if some internal state becomes known, predicting future or past outputs remains computationally infeasible (Kelsey et al. 1998). Examples include algorithms like Fortuna and the output of secure functions such as `/dev/urandom` when properly seeded.

To illustrate the difference between TRNGs and PRNGs visually, Figure 2.2 shows bitmap images generated from sequences produced by a typical PHP PRNG (Figure 2.2b) compared to those from true random sources (Figure 2.2a) (Allen 2016).



(a) Random.org TRNG



(b) PHP rand() function

Figure 2.2: Comparison of bitmap images from PHP PRNG and true random sources

The difference between the pseudo random and true random bitmaps is evident in their visual patterns. Pseudorandom bitmaps, generated by deterministic algorithms, tend to repeat patterns due to the periodicity of the algorithm. In contrast, true random bitmaps, derived from unpredictable physical processes, show no repetition or regularity, resulting in a more chaotic and irregular appearance.

2.5.2 Measures of Randomness

Randomness can be measured by assessing the entropy, which represents the level of unpredictability within a data source. Entropy quantifies the degree of disorder or uncertainty, providing a metric to evaluate the quality of random sequences. The Shannon entropy $H(X)$ of a discrete random variable X is defined as,

$$H(X) = - \sum_{x \in A} p(x) \log_2 p(x)$$

where $p(x)$ is the probability of each unique outcome in the sequence (Shannon 1949). Higher entropy values indicate greater unpredictability. A random number generated with higher entropy means that number is highly unpredictable (Shannon 1949; Kumari, Alimomeni, and Safavi-Naini 2015; Cieslarová 2018). In security-sensitive applications, a related measure, min-entropy is often considered.

$$H_{\infty}(X) = - \log_2(\max p(x_i))$$

The min-entropy of a discrete random variable X represents a lower bound on its overall entropy, expressed in bits. It is commonly used as a worst-case indicator of the uncertainty in observing X (Cover and Thomas 1991).

Despite these mathematical definitions, it remains practically difficult to quantify randomness precisely. Various methods exist to evaluate the quality of randomness, with the NIST test suite being one of the most widely recognized standards. This suite offers a comprehensive set of statistical tests to analyze random bit sequences, assessing factors like bit frequency, patterns, and sequence complexity. By applying these tests, one can validate whether a random sequence meets the stringent requirements necessary for cryptographic and other security sensitive applications (Bassham et al. 2010).

2.6 Linux Random Number Generator

The Linux Random Number Generator (LRNG) is responsible for generating cryptographically secure random numbers in Linux systems. It gathers entropy from various unpredictable sources such as keyboard, mouse movements, disk I/O, and network activity. The LRNG is composed of several components that handle entropy collection, entropy extraction, and entropy expansion, which are then used to produce random numbers.

In earlier versions of the LRNG, a distinct structure existed with two main entropy pools, a blocking pool (`/dev/random`) and a non blocking pool (`/dev/urandom`). The `/dev/random` pool would block the process requesting random data until enough entropy was available, ensuring high quality randomness. In contrast, the `/dev/urandom` pool, designed to provide randomness even with limited entropy, would not block and would reuse entropy from the pool even in low entropy situations. At the core of the earlier design was a large entropy pool, initially set to 4096 bytes, which was used to accumulate entropy from various sources. The system was also designed to allow random number generation via `get_random_bytes()`.

However, over time, the LRNG has undergone a series of important updates aimed at enhancing security and simplifying its architecture. The entropy pool size, has been reduced from 4096 bytes to 256 bytes, reflecting a shift towards more efficient memory usage and more predictable behaviour in entropy management. Additionally, the core hash function has been updated, moving away from older methods such as SHA-1 and Linear Feedback Shift Registers (LFSRs). Notably, the LRNG now uses BLAKE2s for both its hash and pseudorandom function (PRF) modes, ensuring better security and performance (Donenfeld 2022; Cieslarová 2018).

A major turning point in the evolution of the LRNG has been the transition from the old blocking and non blocking pool system to a more streamlined architecture. Both `/dev/random` and `/dev/urandom` now utilize the `getrandom()` system call. This unification of interfaces means that once the LRNG is initialized, both interfaces effectively behave the same, offering better predictability and less complexity in entropy management (Donenfeld 2022; Torvalds 2024; *drivers/char/random.c* 2024).

Additionally, the entropy collection mechanism has been significantly modified. Newer sources such as RDSEED/RDRAND hardware generators, and bootloader seeds are now incorporated with traditional sources. These additions aim to strengthen the randomness and overall security of the system (Donenfeld 2022).

Historically, the development of the LRNG has been overseen by Theodore Ts'o, who was instrumental in the design and early development of the system. Over time, the LRNG became increasingly complex, and the need for modernization became apparent. In recent years, Jason Donenfeld has taken over the primary development efforts, focusing on simplifying and securing the architecture while retaining backward compatibility.

Under Donenfeld’s leadership, several important changes have been implemented, including the replacement of older cryptographic methods, such as SHA-1 and Linear Feedback Shift Registers (LFSRs), with modern, cryptographically secure primitives like BLAKE2s and ChaCha20 (Donenfeld 2022; *drivers/char/random.c* 2024). These updates have not only improved the security and performance of the LRNG but also streamlined entropy collection and expansion strategies. The current design reflects a shift towards a more efficient and secure random number generation system for Linux based systems, addressing both historical complexity and the demands of contemporary cryptographic standards.

2.7 Identification of Research Gaps and Questions

The review of the existing literature revealed that while the LRNG has undergone significant updates in its architecture and internal workings, certain areas remain less explored. Notably, the impact of virtualization on the behaviour of the LRNG has not been systematically studied, especially from the perspective of non privileged user space observations. This observation led to the formulation of the first research question, which aims to determine whether distinguishable patterns in LRNG behaviour can be used to detect the presence of virtualization in a single VM setup.

Furthermore, the literature did not address how the LRNG behaves in complex environments of multiple VMs, such as desktop, private cloud infrastructures, and public cloud platforms. This absence of comparative analysis across multiple VM environments motivated the second research question, which seeks to evaluate whether LRNG based virtualization detection remains feasible under different virtualization architectures.

Finally, the review highlighted that virtualization detection techniques are commonly used by malware to avoid analysis within testing environments. Therefore, understanding and mitigating LRNG based detection methods has practical significance for malware analysis and defense strategies. Based on this consideration, the third research question was formulated to explore how virtualization detection through LRNG behaviour can be obscured, considering possible mitigation techniques.

Thus, the research objectives and questions were systematically derived by identifying specific gaps in the current body of knowledge, ensuring that the study is positioned to contribute meaningfully to both the understanding of LRNG behaviour and its potential applications in virtualization detection.

3 Research Methodology

3.1 Experimental Design

3.1.1 Preliminary Experiments

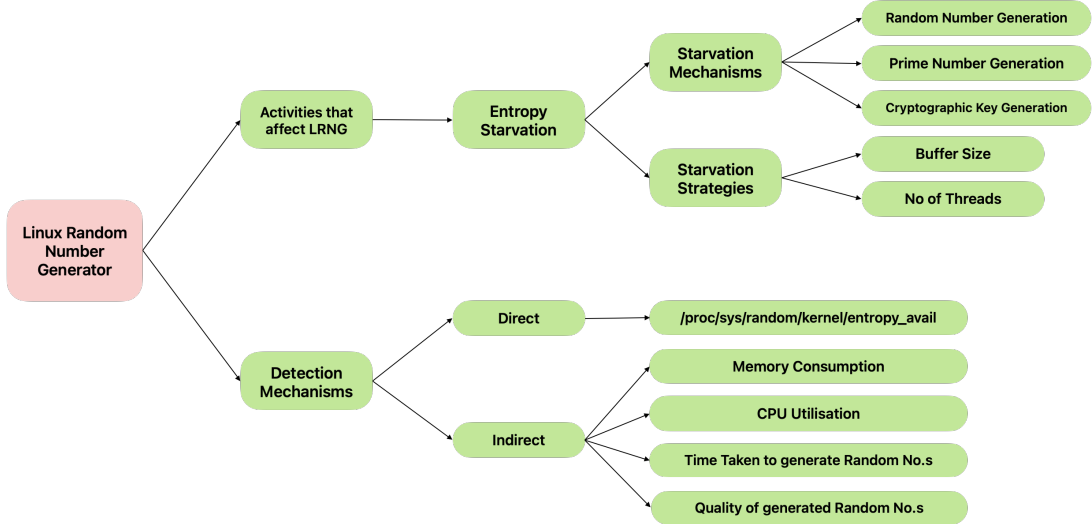


Figure 3.1: Overview of Experiments

The first focus towards the initial experiments was identifying the activities that affect and reflect the behaviour of the LRNG by exploring existing literature. Through this exploration, it was found that the level of entropy associated with the LRNG is a measure that is directly linked to hardware, and this measure can reflect the behavioural changes within the random number generation process. Based on this understanding, attention was given to identifying the types of activities that can cause changes in these entropy values. As a result,

- Generation of random numbers
- Generation of prime numbers and
- Generation of cryptographic keys

were identified as operations that can consume entropy. Based on existing studies, it was hypothesised that entropy related parameters in virtualized environments would be less rich compared to those in bare metal environments, which have closer access to hardware sources (Kumari, Alimomeni, and Safavi-Naini 2015). These insights were used

for the initial experimental design, where the preliminary set of experiments consisted of attempts to create an entropy starvation using the identified mechanisms.

Before moving on to the entropy starvation attempts, another important consideration was the variable strategies in depleting entropy. Entropy depletion can be carried out using various approaches, such as varying the buffer size and the number of threads involved in the operations. While defining these strategies, it was also necessary to determine how to detect whether changes in entropy were actually taking place. The straightforward method of detection is monitoring the entropy value through the `/proc/sys/kernel/random/entropy_avail` interface. However, it was observed that relying on this direct measure was insufficient in capturing behaviour of the LRNG in newer kernel versions. As a result, indirect indicators were also explored as explained in Section 3.1.1.4. These included memory consumption, CPU utilization, the time taken to generate random numbers, and the quality of the generated random numbers. These indirect measures were selected to investigate whether they could reflect underlying LRNG behaviour.

Programs were developed in C and bash script to attempt entropy starvation using the identified entropy consuming mechanisms. These starvation attempts were carried out using different strategies, which included varying the buffer sizes and the number of threads running in parallel as described in following topics. Initially, no observable changes in entropy behaviour were detected. The first round of experiments, conducted on Ubuntu 22.04, used the direct entropy availability measure from the `entropy_avail` interface. However, despite different approaches and configurations, no significant changes in entropy values were observed. According to existing literature this interface has used to indicate changes of entropy levels hence it was decided to attempt these experiments on an older kernel version. Ubuntu 14 was selected for this purpose due to its usage in literature for entropy indicating activities. The same set of experiments was conducted on Ubuntu 14 to observe which of the selected activities had the most noticeable impact on entropy related parameters.

Since the core motivation of this study is to identify whether there is a difference in behaviour of the LRNG between bare metal systems and VMs, the experiments were carried out separately on both environments. To ensure consistency and fairness in comparison, the VMs and bare metal systems were hosted on the same hardware configurations which

will be explained in Section 3.2.4. The initial set of experiments for this comparison was conducted using Ubuntu 14, as it was observed to clearly reflect changes in entropy behaviour. Ubuntu 14 was therefore used as a baseline to better understand which operations and activities have the most significant impact on the LRNG.

3.1.1.1 Attempts to create an entropy starvation through direct random number generation

To attempt entropy starvation through direct random number generation, several approaches were explored. Shell scripts were developed using loops that continuously read from `/dev/random` or `/dev/urandom` using commands such as `"dd if=/dev/random"` and `"head -c"`, with varying block sizes and byte counts ranging from 32 to 8192 (in powers of two). Fluctuations of entropy level through `entropy_avail` in `proc` directory were observed only on Ubuntu 14, but no any changes in OSs with newer kernel versions such as 6.8 on Ubuntu 24.04, 5.15 on Ubuntu 22.04, and 5.14 AlmaLinux 9.

Extending the starvation attempts to multiple threads, a C program was implemented to perform uncontrolled recursive forking to rapidly consume entropy. Though this method lead to the systems crashing due to resource exhaustion, it was an attempt to check whether an entropy fluctuation can be observed in newer kernel versions by stressing the entropy pool. This was later excluded from broader testing due to its potentially destructive impact on systems. A more controlled version using fork based loops was then attempted with progressively increasing the number of threads generating random numbers. However, noticeable entropy fluctuations were observed only in Ubuntu 14 older kernel version.

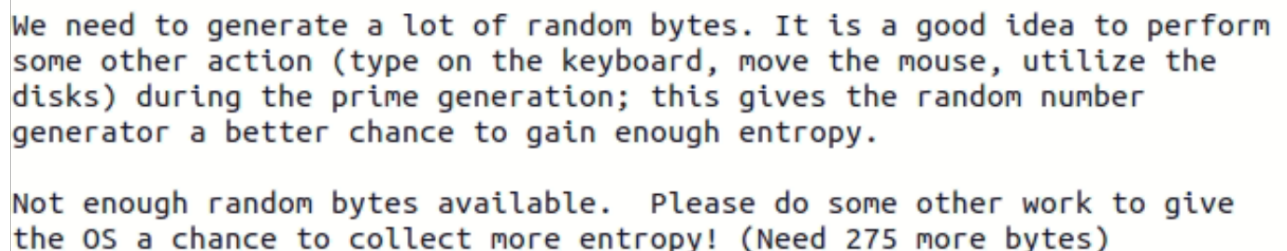
3.1.1.2 Attempts to create an entropy starvation through prime number generation

Following the limited success of direct random number generation in starving entropy, prime number generation was explored as another entropy consuming method. Prime number generation consumes a significant amount of random data, providing a potential to induce an entropy starvation. Shell scripts were implemented to continuously generate large prime numbers using `"openssl prime -generate -bits"`, both in single threaded and controlled forked executions, repeating the task until a shift in entropy levels was observed. Despite its theoretical potential, these experiments also failed to significantly

deplete entropy in newer Linux kernels, with entropy starvation again observed only on older kernel versions.

3.1.1.3 Attempts to create an entropy starvation through cryptographic key generation

Cryptographic key generation was investigated as a further method to create entropy starvation, building on earlier experiments involving direct random number reads and prime number generation. This approach was selected due to the inherently high randomness requirements of cryptographic processes. In particular, RSA and ECC key generation rely heavily on strong entropy sources to ensure secure key creation. Shell scripts were developed to repeatedly generate RSA and ECC keys using both single threaded and controlled forked methods, using tools such as "openssl genrsa" and "openssl ecparam". Additionally, GPG key generation was tested as it is known for its sensitivity to low entropy conditions (*GPG does not have enough entropy*, serverfault.com 2010). While RSA and ECC key generation did not cause significant entropy starvation, GPG key generation on older systems produced an immediate entropy starvation, invoking the error message “not enough random bytes available” (Figure 3.2).



```
We need to generate a lot of random bytes. It is a good idea to perform
some other action (type on the keyboard, move the mouse, utilize the
disks) during the prime generation; this gives the random number
generator a better chance to gain enough entropy.

Not enough random bytes available. Please do some other work to give
the OS a chance to collect more entropy! (Need 275 more bytes)
```

Figure 3.2: Entropy starvation during GPG key generation on Ubuntu 14

3.1.1.4 Attempts to identify indirect parameters reflecting entropy fluctuations

As previous attempts to induce entropy starvation through direct methods such as random number dumps, prime number generation, and cryptographic key generation showed limited impact on modern Linux systems, the focus shifted toward identifying indirect system parameters that might reflect entropy fluctuations in the input pool. The following experiments were designed to capture various metrics that could indicate entropy pool behaviour and depletion.

Initially a C program was developed to measure the time required to generate a specific amount of random data using the `getentropy` function (mapped to syscall `SYS_getrandom` 318 on Ubuntu 14). The goal was to determine if prolonged generation times indicated entropy depletion. Occasional spikes of time taken to generate random numbers were observed hence the monitoring process was extended to include system parameters like context switching and interrupts, which may also respond to changes in entropy availability. A shell script was developed to record these metrics alongside entropy depletion attempts.

The monitoring was later extended across several scenarios such as while the system was idle and during the execution of entropy consuming operations used in above starvation attempts. Each scenario was logged for 180 seconds, with data plotted and analyzed for patterns. Among all operations tested, only GPG key generation consistently exhibited notable entropy depletion, correlating with system parameter fluctuations (Figure 3.3).

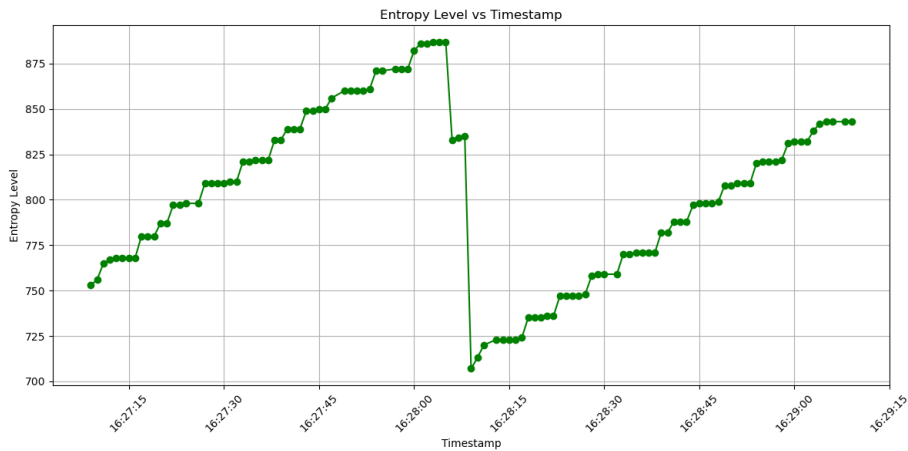


Figure 3.3: Entropy Level fluctuation while generating GPG Keys

To improve accuracy, later experiments introduced concurrent monitoring of system

parameters while executing entropy depletion techniques, capturing real-time interactions. Again, only GPG key generation presented observable depletion behaviour (Figure 3.4).

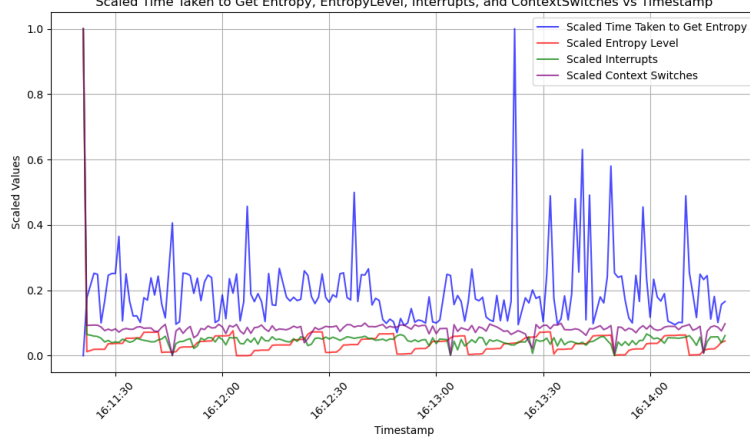


Figure 3.4: Entropy Level & System Parameter fluctuation while generating GPG Keys

As previous monitoring efforts showed some observable patterns of entropy depletion during GPG key generation, it was decided to further examine this process to understand its entropy consumption mechanisms. To achieve this, the GPG key generation sequence was traced, aiming to capture the system calls involved and any interactions with entropy sources. The command used for this tracing was `"strace -e trace=open,read,getrandom,ioctl gpg --batch --generate-key input.txt"`. This trace captured calls related to file opening, reading, entropy acquisition (`getrandom`), and device control operations (`ioctl`). While no direct entropy starvation mechanisms were identified, the trace offered deeper insight into the system call sequence associated with cryptographic key generation.

3.1.1.5 Analyzing the Quality of Random Numbers

As previous experimental approaches did not provide positive results for detecting virtualization, an analysis was conducted to compare the quality of random numbers generated in VMs versus bare metal systems. This approach used the NIST test suite for random bit generation, which statistically evaluates binary sequences through a range of tests designed to assess randomness. This comprehensive test set was executed to capture any potential inconsistencies in random number quality that might reflect virtualization effects. This is discussed in detail in the section 3.4.5.

3.1.2 Evaluating Preliminary Experiments

Based on the preliminary experiments and the comparative analysis of these activities on bare metal and virtual environments, the entropy starvation activities were ranked according to the level of observed impact they had on kernel level entropy indicators in Ubuntu 14. This ranking was derived from observations across both bare metal and virtual environments of Ubuntu 14 and helped identify the most influential operations.

Experiment	Buffer Size	Observable Impact
Generating random numbers	Less than 1024	Low
Generating random numbers	4096	High
Generating prime numbers	4096	Medium
Generating RSA Keys	4096	Low
Generating ECC Keys	-	Low
Generating GPG Keys	-	High

Table 3.1: Listing entropy starvation activities in the order of the fluctuation they made on older kernels

According to the evaluation, random number generation using the "dd" command with larger buffer sizes, as well as GPG key generation, were found to have the highest impact on the entropy pool. Among these, GPG key generation demonstrated a particularly significant level of quick entropy depletion, while other cryptographic operations such as RSA and ECC key generation did not show comparable levels of impact. However, considering that these experiments were to be extended across different kernel versions, it was important to select an entropy depleting activity with minimal dependency on library specific updates. Given that GPG related libraries have undergone significant changes across kernel and distribution versions, it was decided to proceed with direct random number generation using the "dd" command, which offered a more consistent and version independent mechanism for entropy starvation.

3.1.3 Experimental Approaches

After identifying that the generation of random numbers using the `dd` command had the most significant impact on the entropy pool, the subsequent experiments were designed based on this activity. In terms of detection mechanisms, as it was observed that direct entropy value measurement through the "`entropy_avail`" interface was not effective in recent kernel versions, indirect measurements were considered. Among these, memory and CPU utilization data showed limited effectiveness in capturing entropy related measures as standard memory and CPU monitoring tools were not capable of capturing the fine grained variations in these performance metrics. Therefore, the focus shifted towards two potentially promising indicators, the rate of random number generation and the quality of generated random numbers.

The rate of random number generation refers to the time taken to generate a specified amount of random bytes. This serves as an indicator of how efficiently the LRNG architecture is able to supply random numbers under varying system conditions in environments with fluctuating entropy availability.

The quality of the random numbers are evaluated using standard test suites such as the NIST Statistical Test Suite. The quality reflects the strength of the entropy source that underpins the LRNG. This is important because the LRNG is designed to use available entropy to produce random numbers that are as close as possible to true randomness. Consequently, quality measurements can provide insight into where the output lies on the spectrum between true and pseudo randomness (Refer Section 2.5.1 for details). Based on these two aspects, the rate and the quality of random number generation, next stage of experiments in this study was formulated.

All the experiments conducted in this study are designed to address three key research questions. The first question explores whether virtualization presence can be detected through LRNG behaviour in a single VM setup. For this, experiments on a single VM desktop setup were performed, focusing on rate and quality of random number generation under both high and low impact conditions.

The second research question investigates whether virtualization detection is possible in multiple VM setups, with experiments spanning across multiple VM desktop as well as private and public cloud platforms. Similar measurements of random number generation were taken, allowing for comparative analysis between different environments.

The third question examines methods to obscure virtualization detection if it exists, which involves rerunning all the experiments with additional random number generation tools and daemons activated. To investigate this, three experimental rounds were conducted, the first with only the `haveged` daemon active, the second with only `jitterentropy-rngd`, and the third with both tools active at the same time. These software based entropy generators were selected for their capacity to induce high frequency entropy in the system, thereby influencing the behaviour of the LRNG. In each round, random number generation experiments were repeated using the same procedure applied in Research Questions (RQ) 1. This ensured that comparisons remained consistent across configurations. The objective was to evaluate whether these entropy enhancing tools could reduce or obscure the virtualization detection previously observed between virtual and bare metal environments.

3.1.4 Establishing Baselines and Impact Levels

With the focus shifting toward measuring the rate and quality of random number generation, it was essential to first establish appropriate baselines. Several baseline measures were considered to ensure consistency and reliability in the experimental outcomes.

The first baseline involved recording the initially available entropy in a given environment before any experimental activity was conducted. This value served as a reference point for evaluating the impact of the entropy starvation attempts. Even though modern kernel versions may not reflect any changes in entropy values it was still considered as a foundational metric across all test environments.

In alignment with the core objective of the research, which is to compare entropy behaviour between bare metal and virtualized environments, all measurements taken in VM setups were analyzed relative to those obtained from corresponding bare metal systems. The bare metal environments functioned as the control, allowing for a comparative assessment of how entropy related parameters behave in non virtualized systems.

Furthermore, entropy measurements obtained from older Linux versions, such as Ubuntu 14, were also considered as a foundation. These environments, where entropy fluctuations were more visibly observable, provided additional insight into how LRNG behaviour evolves across system versions and configurations.

3.1.4.1 Hardware Limitations and Relative Comparisons

An additional challenge encountered during the experimental design was the inability to make direct comparisons between bare metal machines and VMs due to hardware performance mismatches. Despite hosting both environments on the same physical hardware configurations, resource allocation limitations in virtualized systems created inconsistencies in performance characteristics. As a result, direct comparisons of raw metrics were found to be invalid. For example, the absolute time taken to generate 2MB of random numbers on bare metal cannot be directly compared to the absolute time taken to generate 2MB of random numbers on a VM because it is obvious that the VM has less resources.

To overcome this, all comparisons were made based on differences rather than absolute values. For example, instead of comparing the time taken to generate 2MB of random numbers on bare metal versus on a VM, the experimental design focused on internal delta values within each environment. A simple example would be measuring the time difference between generating 2MB and 4MB of random numbers in the same system, and then comparing that difference across environments. This relative approach helped normalize variations in hardware performance and ensured more accurate interpretation of entropy related behaviours.

3.1.4.2 Impact Levels and Experimental Conditions

To refine the experimental setup with relative comparisons as mentioned above, two impact levels were defined based on the level of background entropy depletion expected during measurements.

Activities with a **lower impact** score were designed to operate under minimal entropy depletion conditions. These experiments primarily focused only on data collection without any background entropy starvation, ensuring that the entropy pool remained relatively stable.

In contrast, activities with a **higher impact** score were carried out under higher entropy depletion conditions. In these experiments, while the main data collection was in progress, an additional background script was executed to deplete entropy in parallel.

3.2 Experimental Setup

3.2.1 Environments used for Experiments

To ensure consistent and comprehensive results regarding the behaviour of the LRNG in virtualized environments, experiments were conducted across multiple deployment settings.

- Desktop Environment
- Private Cloud Environment - University cloud infrastructure
- Public Cloud Environment - Amazon Web Services (AWS) and Google Cloud Platform (GCP)

3.2.2 Virtual Machine Monitors (VMMs) used for Experiments

The experiments also included a range of widely used VMMs to include differences across virtualization technologies and to strengthen the generalizability of the findings.

- Desktop Environment : Virtual Box (Version 7.0.20 r163906), VMware® Workstation 17 Pro (Version 17.6.1 build-24319023), QEMU (Powered by libvirt Version 4.0.0)
- Private Cloud Environment : VMWare ESXi (Version 7.0.3)
- Public Cloud Environment : AWS - Nitro (Custom KVM based VMM by AWS), Google Cloud - Compute Engine (Custom KVM based VMM by Google)

3.2.3 Operating Systems used for Experiments

To represent a broad range of Linux distributions and kernel versions, the experiments used OSs from major Linux families, capturing a wide spectrum of the LRNG's behaviour across versions.

- Debian Based : Ubuntu Server 22.04 LTS (Kernel 5.15) and 24.04 LTS (Kernel 6.8) (Ubuntu 14 OS with Kernel 4.4 is used as a reference point to the older version of LRNG)
- RedHat Based : AlmaLinux 9.4 (Kernel 5.14)

The reason behind selecting these distributions was primarily driven by the differences in their kernel versions. Both Ubuntu Server 22.04 and AlmaLinux 9.4 utilize Linux Kernel 5 versions. Ubuntu Server 24.04 includes the more recent Linux Kernel 6.8 however there is no corresponding RedHat distribution to this kernel version. Therefore, to maintain consistency for comparison purposes, Ubuntu 22.04 and AlmaLinux 9.4 were used as the common ground between the two distribution families, while Ubuntu 24.04 was added to observe newer kernel behaviour within the Debian based lineage.

Although Ubuntu Server 22.04 and AlmaLinux 9.4 use slightly different kernel versions, this minor difference was negligible for the purpose of the experiment. The update from kernel 5.14 to 5.15 did not introduce any changes related to the LRNG, and no significant impact on the entropy pool or RNG functionality was found. As a result, both kernel versions were considered functionally equivalent for the study.

It is important to note that AlmaLinux 9.4 was not included in the public cloud experiments. This exclusion was due to availability and performance constraints. AlmaLinux was not offered within the free-tier limits on Amazon Web Services (AWS), and although it was technically accessible on Google Cloud Platform (GCP), the instance performance was significantly degraded, making it impractical for executing the experiments reliably. Therefore, only Ubuntu Server 22.04 and 24.04 were used for public cloud data collection.

3.2.4 Hardware Specifications used for Experiments

The physical machines used for both the bare metal and hosting desktop VMs were equipped with an Intel Core i5-3570 CPU at 3.40 GHz, 8 GB of DDR3 RAM at 1600 MT/s, and a 500 GB SATA Hitachi HDS72105 SATA hard disk drive operating at 7200 RPM. Each VM on this machine was allocated 2 GB of RAM, 20 GB of HDD, and 2 virtual CPUs. These configurations applied to all three VMMs used in the desktop setup.

In the desktop environment, experiments involving multiple VMs were conducted with 3 VMs running simultaneously under the same VMM. This number was selected as it is the maximum feasible configuration given the resource constraints of the host machine.

In private cloud, each VM was provided with 2 GB RAM, 15 GB HDD, and 2 virtual CPUs. In the public cloud environments there were certain limitations in resource allocation due to restrictions of free service offerings. Therefore, each VM instance on AWS and Google Cloud was limited to 1 GB RAM, 20 GB HDD, and 1 virtual CPU.

To ensure a clear and structured evaluation, each instance is assigned a unique label based on the RQ, environment, VMM or hardware platform, and the OS (Table 3.2).

RQ	Environment	Bare Metal / VMM	OS	Label
RQ1	Desktop	Bare Metal	Ubuntu 24.04	dsk-host-u24
			Ubuntu 22.04	dsk-host-u22
			AlmaLinux 9.4	dsk-host-al9
		VirtualBox	Ubuntu 24.04	dsk-vbox-u24
			Ubuntu 22.04	dsk-vbox-u22
			AlmaLinux 9.4	dsk-vbox-al9
		VMware	Ubuntu 24.04	dsk-vmw-u24
			Ubuntu 22.04	dsk-vmw-u22
			AlmaLinux 9.4	dsk-vmw-al9
		QEMU	Ubuntu 24.04	dsk-qemu-u24
			Ubuntu 22.04	dsk-qemu-u22
			AlmaLinux 9.4	dsk-qemu-al9
RQ2	Desktop	VirtualBox	Ubuntu 24.04	mdsk-vbox-u24
			Ubuntu 22.04	mdsk-vbox-u22
			AlmaLinux 9.4	mdsk-vbox-al9
		VMware	Ubuntu 24.04	mdsk-vmw-u24
			Ubuntu 22.04	mdsk-vmw-u22
			AlmaLinux 9.4	mdsk-vmw-al9
		QEMU	Ubuntu 24.04	mdsk-qemu-u24
			Ubuntu 22.04	mdsk-qemu-u22
			AlmaLinux 9.4	mdsk-qemu-al9
	Private Cloud	VMware ESXi	Ubuntu 24.04	pvt-vmw-u24
			Ubuntu 22.04	pvt-vmw-u22
			AlmaLinux 9.4	pvt-vmw-al9
	Public Cloud	AWS Nitro	Ubuntu 24.04	pub-aws-u24
			Ubuntu 22.04	pub-aws-u22
		Google Cloud	Ubuntu 24.04	pub-ggl-u24
			Ubuntu 22.04	pub-ggl-u22

RQ	Environment	Bare Metal / VMM	OS	Label
RQ3	Desktop	VirtualBox	Ubuntu 24.04	odsk-vbox-u24
			Ubuntu 22.04	odsk-vbox-u22
			AlmaLinux 9.4	odsk-vbox-al9
		VMware	Ubuntu 24.04	odsk-vmw-u24
			Ubuntu 22.04	odsk-vmw-u22
			AlmaLinux 9.4	odsk-vmw-al9
		QEMU	Ubuntu 24.04	odsk-qemu-u24
			Ubuntu 22.04	odsk-qemu-u22
			AlmaLinux 9.4	odsk-qemu-al9

Table 3.2: Summary of Experimental Configurations across Research Questions

3.3 Data Collection

3.3.1 System Preparation for Testing

Each experimental system was configured to operate under minimal usage conditions to ensure consistency and reduce noise in the collected data. This was achieved by disabling or removing unnecessary background services and processes, allowing only essential components to remain active. The objective of this configuration was to create a controlled and interference free environment, minimizing the impact of unrelated activities on the behaviour of the LRNG.

However, in cloud based VMs, this level of control was limited due to dependencies on background services enforced by the cloud platform’s virtualization infrastructure. Certain system services could not be disabled entirely, as doing so would result in termination or unresponsiveness of the VMs.

3.3.2 Script Implementation and Interface Selection

Implementation of data collection scripts was guided by the current behaviour of the LRNG interfaces, as discussed in the Section 2.6. Traditionally, the difference between `"/dev/random"` and `"/dev/urandom"` was their blocking behaviour and the resource pool, where `"/dev/random"` would use a blocking pool which blocks when entropy was insuf-

ficient and `"/dev/urandom"` would use a non blocking pool. However, as noted in the literature, this distinction is no longer maintained in modern Linux kernels, since both interfaces use the same entropy pool and internally rely on the `"getrandom()"` system call with its flags.

To maintain backward compatibility and ensure alignment with prior studies, data was collected from both `"/dev/random"` and `"/dev/urandom"` interfaces. This approach was used consistently across experiments to reflect traditional usage patterns while also capturing any subtle variations that might still exist across different environments.

However, for background entropy depletion in high impact scenarios, the background depletion program was configured to use the `getrandom()` system call with the `"GRND_RANDOM"` flag. This configuration enforces blocking behaviour to reliably induce entropy depletion and observe corresponding system responses.

Furthermore, buffer sizes and sample sizes for data collection were carefully selected based on the defined baselines and experimental objectives, informed by preliminary testing as described in Section 3.1

3.3.3 Selection of Buffer Sizes

For both rate of random number generation and quality of random number generation experiments, it was essential to identify buffer sizes that could reveal differences between virtualized and bare metal environments. To address this, three levels of buffer sizes were selected,

- 2 MB
- 4 MB
- 6 MB

These buffer levels were chosen to evaluate how the amount of random data generated under different impact levels as defined earlier, could reflect the performance and entropy characteristics of the LRNG. The variation in buffer size allowed for assessment of how the system behaves under increasing demands on the entropy pool.

3.3.4 Sample Size and Repetition for Generalization

Initially, data was collected by generating 100 samples for each buffer size level and impact level. The amount 100 data samples was selected to ensure compliance with optimal testing requirements for NIST test suit used in assessing the quality of random numbers.

- 100 samples of 2 MB under each impact level (high and low)
- 100 samples of 4 MB under each impact level (high and low)
- 100 samples of 6 MB under each impact level (high and low)

However, upon early analysis, it was noticed that a single set of 100 samples per level might still be susceptible to various system states or occasional outliers. As a result, to ensure statistical robustness and generalizability, the each experiment was repeated ten times, resulting in 1000 random number samples per buffer level per execution. Each of these experiments was conducted under both,

- Low impact conditions - where no additional entropy depleting processes were running while data collection and
- High impact conditions - where concurrent background processes were running for entropy depletion while data collection

This resulted in a comprehensive dataset across environments, buffer levels, and impact levels, allowing for reliable comparative and statistical analysis. The Table 3.3 summarizes the buffer sizes and corresponding impact levels used in the data collection phase for both the `/dev/random` and `/dev/urandom` interfaces.

3.3.5 Initial Execution and Observed Network Effects

The initial execution of the data collection scripts was performed under minimal system load, but with network access enabled. In cloud environments, the initial data collection was conducted via remote SSH connections. However, upon analyzing the collected data, a significant anomaly was identified in one of the graphs as shown in Figure 3.5. During the execution of the data collection script SSH login was disconnected, which correlated with the noticeable spike in the time taken to generate random numbers.

This observation revealed that network related interrupts also could significantly influence the performance of LRNG related operations. To address this influence of network services, the data collection process was re-executed in all environments, including desktop, private cloud, and public cloud. In cloud environments, this required a more careful approach due to their reliance on remote access.

Interface	Buffer Size	Impact Level
/dev/random	2MB	High
		Low
	4MB	High
		Low
	6MB	High
		Low
/dev/urandom	2MB	High
		Low
	4MB	High
		Low
	6MB	High
		Low

Table 3.3: Summary of Interfaces, Buffer Sizes and Impact Levels of Experiments

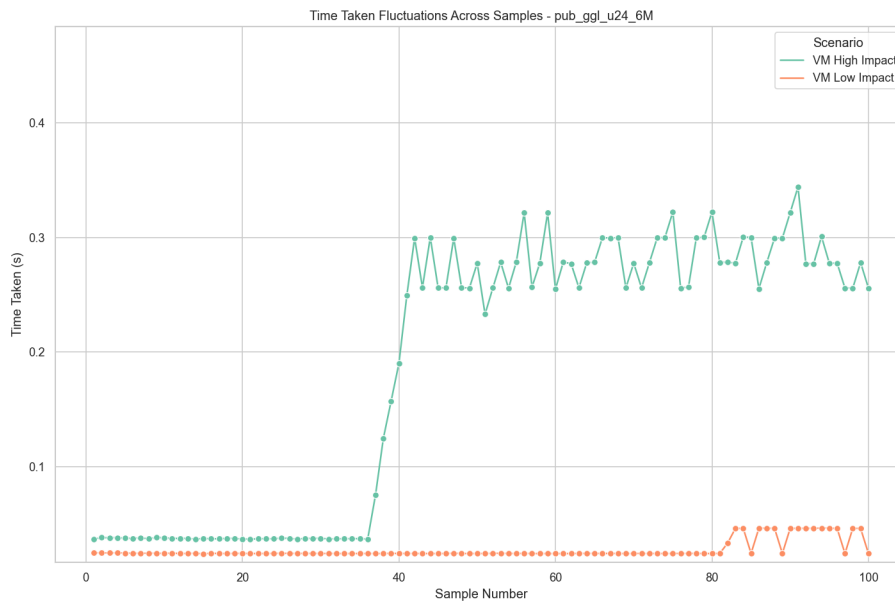


Figure 3.5: Time taken to generate 6MB random numbers during SSH disconnection

The solution involved scheduling the data collection scripts using cron jobs, temporarily disabling SSH and network access before script execution and re-enabling SSH services once data collection was complete. This approach ensured that no external network events could interfere with the timing or entropy consumption behaviour of the system during data collection.

3.3.6 Data Collection with Entropy Enhancing Tools

To investigate whether entropy enhancing tools influence the behaviour of the LRNG in virtualized environments, RQ3 experiments introduced three controlled configurations involving entropy enhancing daemon services. This strategy was selected upon the hypothesis that increasing entropy richness could possibly hide the virtualization presence detection through the behaviour of LRNG. Data was collected under the following isolated and combined scenarios,

- Only `haveged` service active
- Only `jitterentropy-rngd` service active
- Both services active concurrently

It is important to note that the virtualized environments selected for these tests were limited to those in which clear evidence of virtualization presence had been previously observed in baseline measurements under RQ1. This constraint was necessary to ensure that the analysis focused on conditions where virtualization detection had already been positively established through LRNG behaviour.

3.4 Data Analysis Methods

3.4.1 Data Visualization

The first method used for analyzing the data was visualization. This approach allowed for a preliminary examination of the collected data and its trends. Specifically, for the rate of random number generation experiments, the data samples were plotted on graphs with the time taken to generate random numbers on the y-axis and the sample number on the x-axis. This visual representation enabled a direct comparison between bare metal

and virtual environments. Key features observed from these plots included the patterns in distribution of data points, peaks in data and variance across different sample sets.

3.4.2 Basic Statistics

To extend the findings from the visual analysis, basic statistical methods were used. These included measures such as the mean, variance, standard deviation and the coefficient of variation. Each of these was selected for a specific purpose based on the experimental findings.

1. **Mean** is used for the identification of average time taken to generate random numbers, particularly to quantify the average performance under high impact and low impact scenarios. Differences in mean values were critical for determining whether certain conditions resulted in longer generation times across environments, as highlighted in Observation 1 in the Section 4.

$$\text{Mean} = \frac{\text{Sum of all values}}{\text{Total number of values}}$$

$$\mu = \frac{\sum_{i=1}^n x_i}{n}$$

2. **Variance** is used to measure how much the values in the dataset deviate from the mean. This was relevant to Observation 3 described in Section 4, where the degree of dispersion between high impact and low impact scenarios was found to vary across environments and OSs.

$$\text{Variance} = \frac{\text{Sum of squared differences from the mean}}{\text{Total number of values} - 1}$$

$$\sigma^2 = \frac{\sum_{i=1}^n (x_i - \mu)^2}{n - 1}$$

3. **Standard Deviation**, the square root of variance is used to represents the average amount by which the values differ from the mean.

$$\text{Std.Dev} = \sqrt{\text{Variance}}$$

$$\sigma = \sqrt{\sigma^2}$$

4. **Coefficient of Variation (CV)** is calculated to allow a normalized comparison of variability between datasets with differing mean values. This was useful in Observation 3 to compare the relative spread of high and low impact distributions across environments where absolute values differed significantly.

$$CV = \frac{\text{Standard Deviation}}{\text{Mean}}$$

$$CV = \frac{\sigma}{\mu}$$

3.4.3 Error Metrics

However, basic statistics were found to be insufficient for capturing certain variations and scatter observed in the data. Specifically, some graphs showed significant fluctuations and a spread of values across the axes. To capture this more detailed spread and scatter, additional error metrics were introduced

1. **Sum of Squared Errors (SSE)** measures how much individual values differ from their respective row-wise mean across multiple files. The errors (differences from the mean) are squared before summing them to ensure all differences are positive. This helped summarize the total error present in the sample, which was also useful in Observation 3 described in Section 4.

$$\text{Total SSE} = \sum (\text{each row's sum of squared errors})$$

$$\text{SSE} = \sum_{i=1}^n \sum_{j=1}^m (x_{ij} - \mu_i)^2$$

2. **Mean Squared Error (MSE)** is found by dividing the total sum of squared errors by the total number of samples, provides an average measure of error per data point.

$$\text{MSE} = \frac{\text{Total SSE}}{1000}$$

$$\text{MSE} = \frac{\text{SSE}}{1000}$$

3.4.4 Peak Detection Metrics

One of the most important findings from the visual analysis was the presence of early peaks in the time taken to generate random numbers, particularly in virtual environments.

These peaks were absent in bare metal environments and provided a key insight into the behaviour of the LRNG under virtualization. Upon careful visual inspection, it was observed that these peaks always occur within first 8-10 samples and the distribution of time taken values get stable within first 12 samples. Therefore, it was decided to apply several peak detection statistical methods to first 12 samples to quantify and analyse the presence of these early peaks.

1. **Peak-to-Mean Ratio (PMR)** is used to assess the relative height of the early peak compared to the average behaviour. A higher PMR suggests that the peak is much larger than the average behaviour.

$$PMR = \frac{\text{Max of first n samples}}{\text{Overall mean}}$$

$$PMR = \frac{\max(x_1, x_2, \dots, x_n)}{\mu}$$

2. **Z-Score for Early Peak** measures how many standard deviations the highest value among the first n samples is above the overall mean. A high Z-score indicates that the peak is significantly different from the rest of the data.

$$Z_{\text{early peak}} = \frac{\text{Max of first n samples} - \text{Overall mean}}{\text{Standard deviation}}$$

$$Z = \frac{\max(x_1, x_2, \dots, x_n) - \mu}{\sigma}$$

3. **Early Mean Ratio (EMR)** helps in assessing whether the early samples are generally higher than the rest of the dataset.

$$EMR = \frac{\text{Mean of first n samples}}{\text{Overall mean}}$$

$$EMR = \frac{\frac{1}{n} \sum_{i=1}^n x_i}{\mu}$$

4. **Percentage Drop After Peak** measures how much the data drops after the highest early peak. A high drop percentage suggests a strong initial peak followed by stabilization. This provided the clearest evidence for the presence of early peaks.

$$\text{Drop} = \left(\frac{\text{Max of first n samples} - \text{Mean of next n samples}}{\text{Max of first n samples}} \right) \times 100\%$$

$$\text{Drop} = \left(\frac{\max(x_1, x_2, \dots, x_n) - \frac{1}{n} \sum_{i=n+1}^{100} x_i}{\max(x_1, x_2, \dots, x_n)} \right) \times 100$$

3.4.5 Assessing the Quality of Random Numbers

To assess the quality of the generated random numbers, the NIST Test Suite was employed and the following tests were used,

1. Frequency Test, evaluates the proportion of 0s and 1s in the data, checking whether the number of 1s and 0s are approximately equal. A large deviation from equality suggests non randomness.
2. Block Frequency Test, divides the data into blocks and checks whether the frequency of 1s within each block is approximately equal. This helps identify any periodic patterns or biases in the data.
3. Cumulative Sums Test, examines the cumulative sum of the sequence to detect any long-term trends or deviations from randomness. A consistent upward or downward trend suggests non-random behaviour.
4. Runs Test, analyzes the sequence for consecutive occurrences of the same value (runs). It checks whether the number of runs is consistent with what would be expected for a random sequence.
5. Longest Run Test, looks at the longest consecutive sequence of 0s or 1s and compares it to the expected values for a random sequence. A sequence that contains unusually long runs suggests non-randomness.
6. Rank Test, evaluates the rank of sub-sequences to determine whether the sequence behaves as expected for a random sequence. Non-randomness is indicated if the ranks exhibit patterns or clustering.
7. Fourier Transform Test (FFT Test), performs a Fourier transform on the sequence to detect any periodic components. If the sequence has periodicity or patterns, they will appear in the frequency domain.
8. Overlapping Template Matching Test, checks whether overlapping patterns (templates) within the data occur more frequently than expected in a random sequence, which may suggest predictability.

9. Approximate Entropy Test, evaluates the complexity of the data sequence by comparing the likelihood of similar patterns occurring in the data. A high level of regularity or predictability indicates non-randomness.
10. Serial Test, examines the relationships between pairs or larger sets of consecutive bits, ensuring that the sequence does not contain any discernible patterns across multiple bit positions.
11. Linear Complexity Test, assesses the linear complexity of the sequence, which measures the length of the shortest linear feedback shift register (LFSR) that can reproduce the sequence. A low complexity value suggests that the sequence is predictable and thus non-random.

Each of these tests outputs a p-value, which indicates the likelihood that the sequence is random. If the p-value is greater than 0.01, the sample is considered to be random. If the p-value is 0.01 or lower, the sample is deemed non-random.

In this study, it was assumed that if a sample fails any of these quality tests, it cannot be considered fully random. This assumption was based on the notion that failure of any test implies the presence of non-random characteristics in the sequence. To evaluate the quality of random number generation across different environments, the percentage of samples that passed all the tests was calculated. Due to the extensive time taken to generate NIST test result (24+ hour per each set) only 100 data samples were subjected to tests at each buffer size (2MB, 4MB, 6MB samples). The percentage of passing samples for each environment was then compared across bare metal and virtual environments, allowing for the identification of potential differences in the randomness quality of the generated numbers.

It is also important to note that the notion of assessing randomness quality through statistical tests is inherently limited. Randomness, by nature, is not easily quantifiable, and even true random number generators are not expected to consistently achieve a 100% pass rate across all NIST tests. This limitation is acknowledged within the research community, and accurate evaluation of randomness remains a challenge. (Kenny and Mosurski 2005)

3.5 Summary of the Chapter

This chapter outlined the methodology used to investigate the behaviour of the LRNG in virtualized environments. The experimental design focused on preliminary experiments, experimental approaches, and establishing baselines and impact levels. Baselines were determined using initial entropy measures, and impact levels were categorized to compare random number generation in bare metal versus virtualized environments. The experimental setup involved testing across desktop, private cloud, and public cloud environments, using various VMMs like VirtualBox, VMware, and QEMU, and different Linux distributions such as Ubuntu and AlmaLinux.

For data collection, experiments were conducted with minimal system usage, selecting buffer sizes of 2MB, 4MB, and 6MB. Data was collected under high and low-impact conditions, and the data collection was repeated to ensure reliability.

Finally, data analysis used visualizations to compare random number generation rates across environments, and statistical methods were applied to quantify the visual observations. The quality of random numbers was assessed using the NIST Test Suite, with results compared between bare metal and virtual environments. This methodology combines controlled experiments and rigorous analysis to assess how virtualization presence can be detected from the user space using entropy and random number generation parameters.

4 Results Evaluation and Discussion

4.1 Results of Research Question 1

How can a program running in the user space of a Linux VM detect the underlying virtualization platform using Linux Random Number Generator and related parameters?

The first research question, which investigates whether the presence of virtualization can be detected from the user space of a VM setup using the LRNG and related parameters, is addressed through experiments conducted in the desktop environment. These experiments involved running a single VM at a time as the guest OS on the host machine and collecting data as specified in the Section 3.3.

4.1.1 Rate of Random Number Generation

4.1.1.1 Observation 1 - Difference of time distribution between high and low impact scenarios

Distribution of time taken to generate a given amount of random data under high impact and low impact scenarios were observed to have noticeable differences in bare metal and virtual environments. In some OSs, the average time taken under high impact condition was higher than that of low impact condition in bare metal while the time taken under low impact condition was higher in VMs. In some OSs, these observations were reversed. Following are the detailed findings across different OSs.

Ubuntu 24.04 with kernel version 6.8

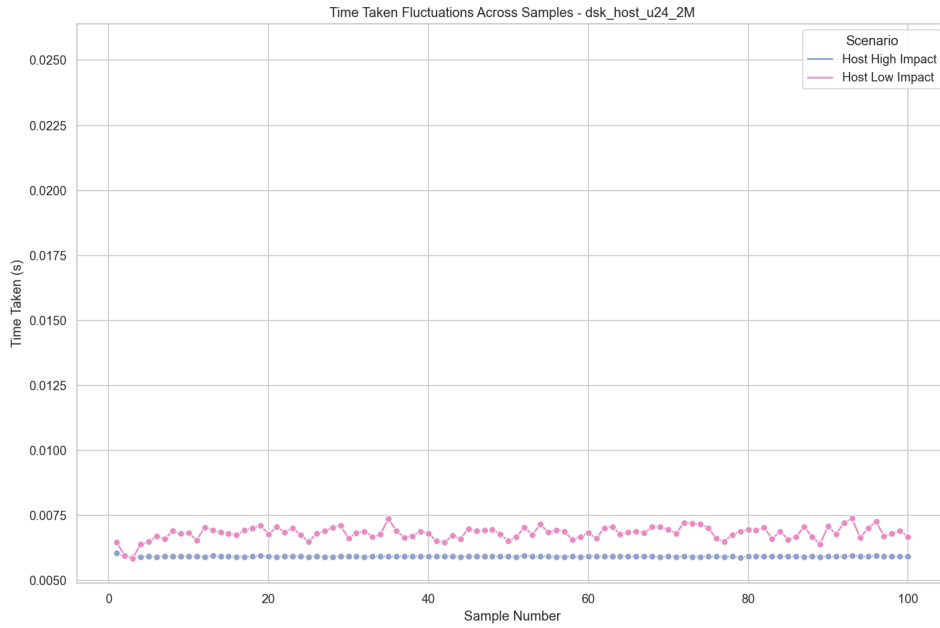
As depicted in the Figure 4.1a, time taken to generate 2MB random numbers under the low impact scenario (pink) is greater than that under the high impact scenario (purple) in the bare metal environment. However in virtual environments, the time taken to generate 2MB random numbers under the low impact scenario (orange) is lower than that under the high impact scenario (green) (Figure 4.1b, 4.1c and 4.1d)

This inversion of behaviour between environments is also reflected through descriptive statistics. The difference between the mean of the distributions (high impact minus low impact) were taken and the difference is found to be negative in Ubuntu 24.04 bare metal environments while it is positive in Ubuntu 24.04 VMs. (Table 4.1)

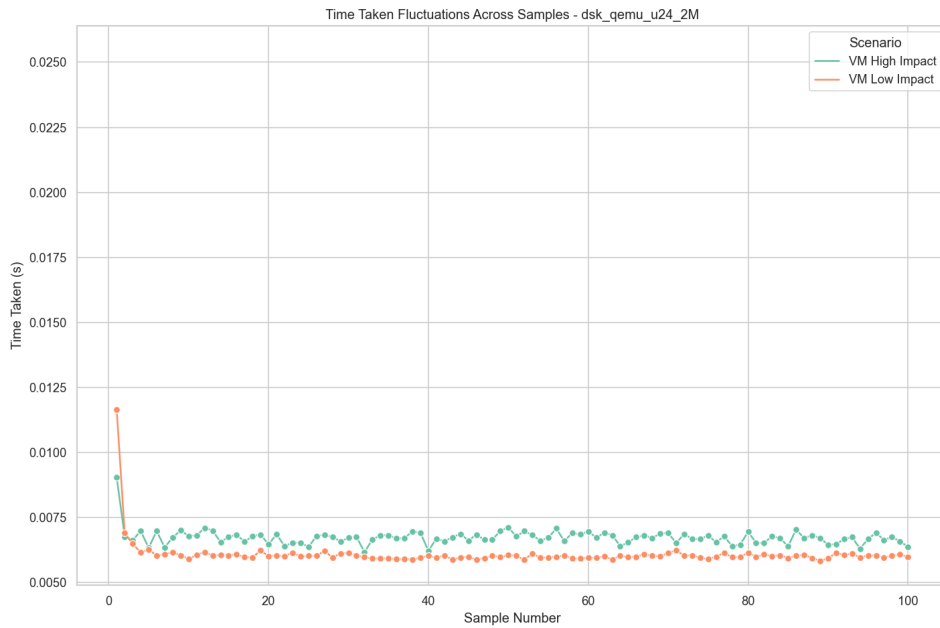
These patterns were consistent across both `/dev/random` and `/dev/urandom` interfaces under 2MB buffer size. However, this observation was not uniform nor consistent with buffer sizes of 4MB and 6MB. The detailed statistical measures for these experiments are provided in the Appendix (Table A.1 and A.2).

Ubuntu 22.04 with kernel version 5.15

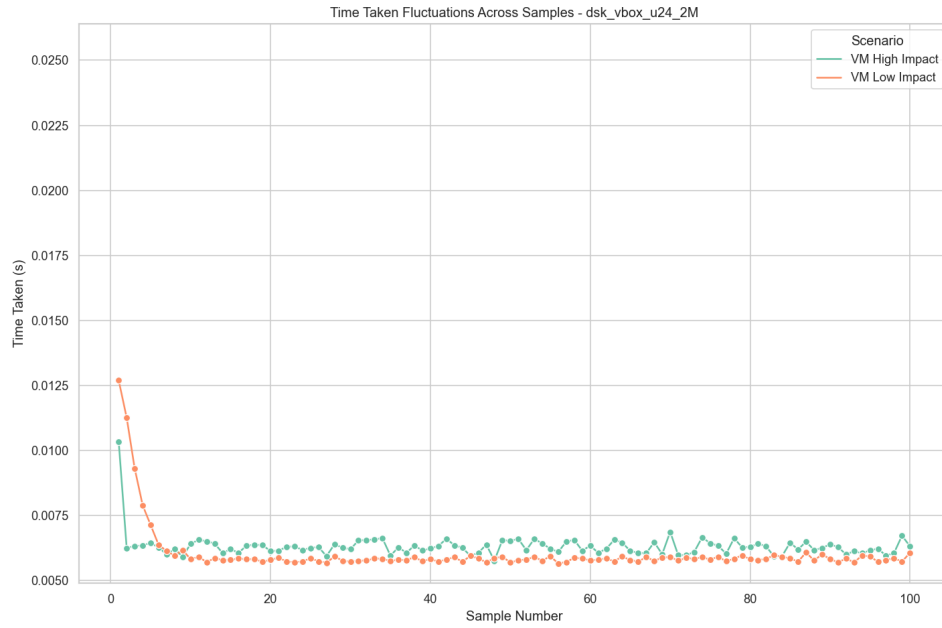
In Ubuntu 22.04, the observed pattern is inverted compared to Ubuntu 24.04. The



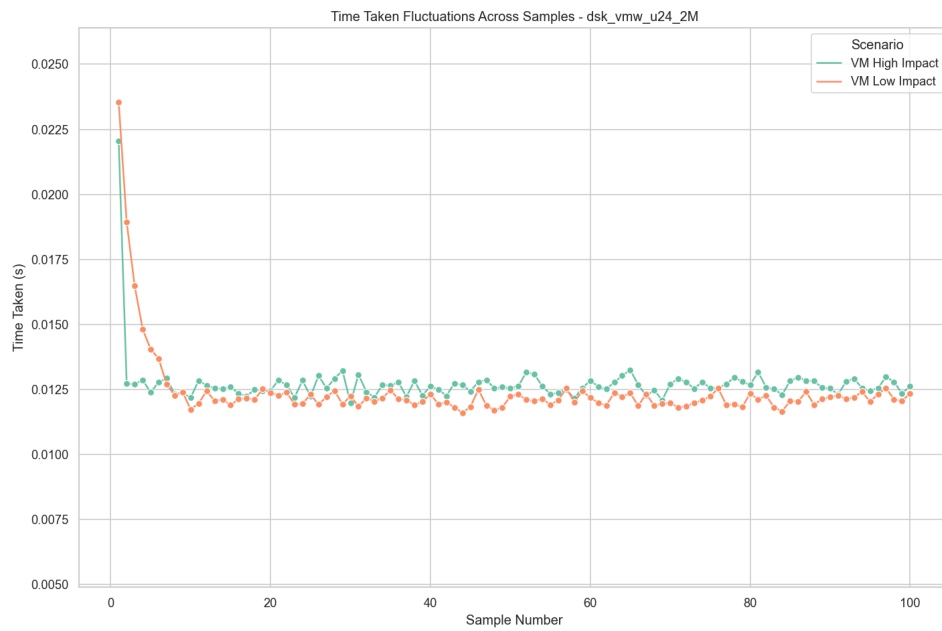
(a) Ubuntu 24.04 on Bare Metal



(b) Ubuntu 24.04 VM on QEMU

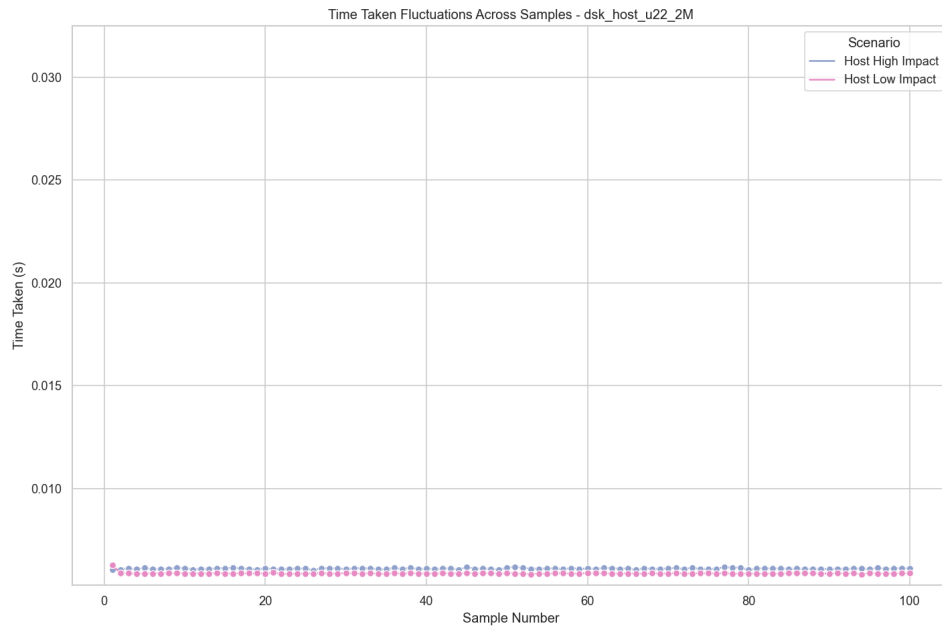


(c) Ubuntu 24.04 VM on Virtual Box

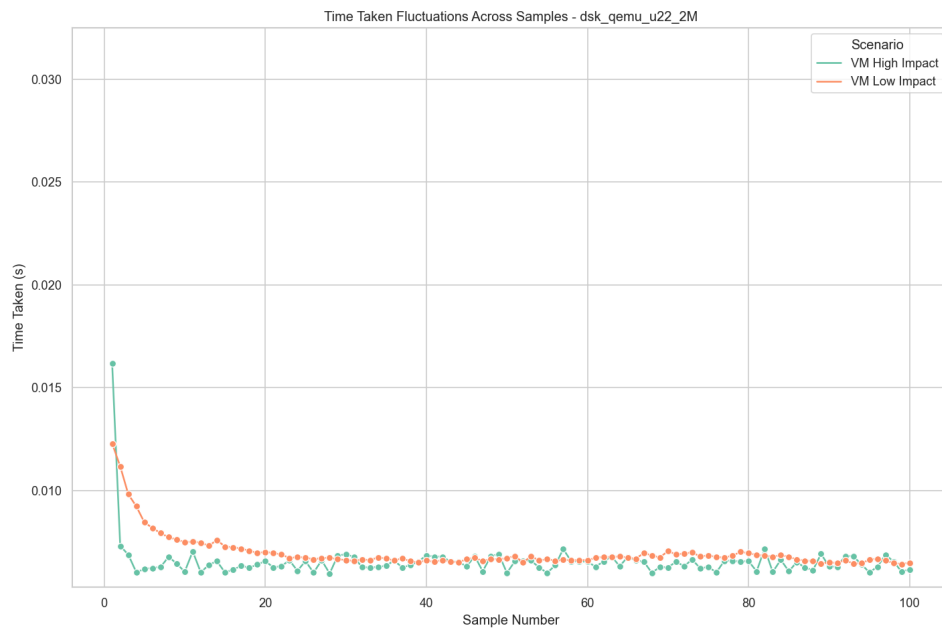


(d) Ubuntu 24.04 VM on VMWare

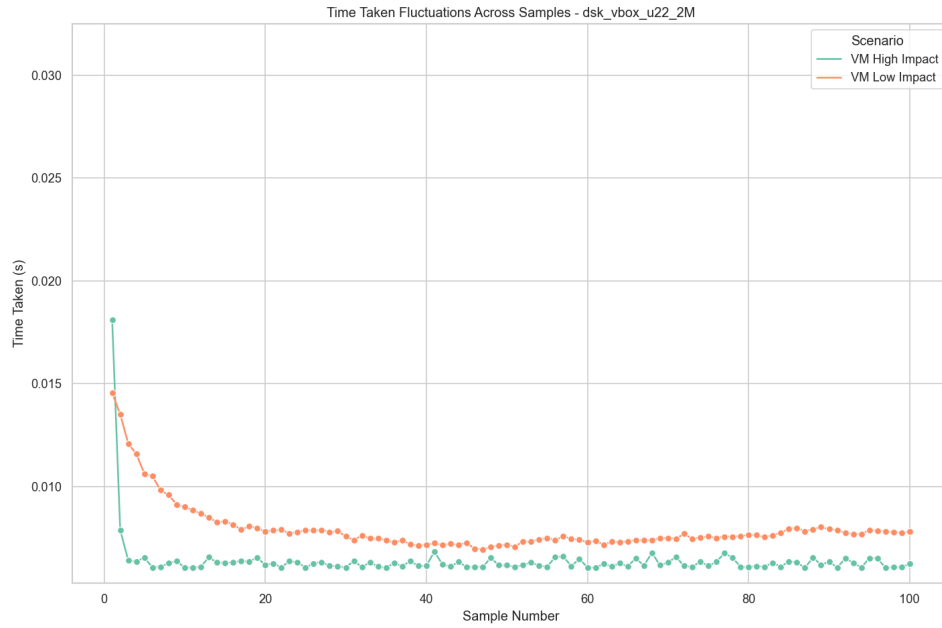
Figure 4.1: Distribution of time taken to generate random numbers under high and low impact scenarios on Ubuntu 24.04 bare metal and VM environments



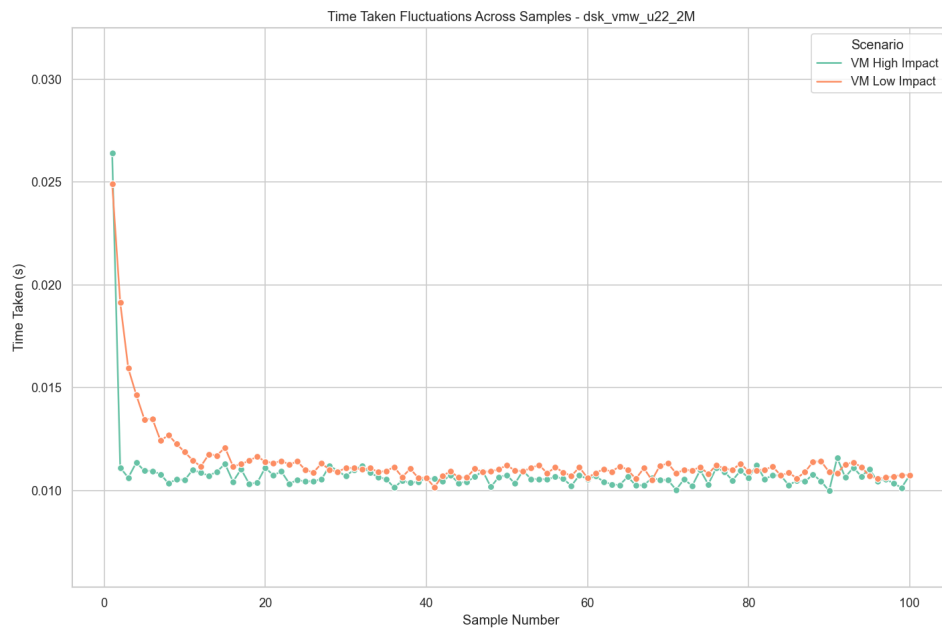
(a) Ubuntu 22.04 on Bare Metal



(b) Ubuntu 22.04 VM on QEMU



(c) Ubuntu 22.04 VM on Virtual Box



(d) Ubuntu 22.04 VM on VMWare

Figure 4.2: Distribution of time taken to generate random numbers under high and low impact scenarios on Ubuntu 22.04 bare metal and VM environments

Interface	Size	Location	Impact	Mean	Mean Difference (High - Low)
/dev/random	2MB	dsk-host-u24	high	0.00592832	-0.00089834
			low	0.00682667	
		dsk-qemu-u24	high	0.00673412	0.00064556
			low	0.00608856	
		dsk-vbox-u24	high	0.00631766	0.00029177
			low	0.00602588	
		dsk-vmw-u24	high	0.01272027	0.00031262
			low	0.01240765	

Table 4.1: Mean value differences of time taken on Ubuntu 24.04 environments

time taken to generate 2MB of random numbers under the high impact scenario (purple) is higher than that under the low impact scenario (pink) in the bare metal environment (Figure 4.2a). While the opposite behaviour was observed in virtual environments, the time taken to generate 2MB random numbers under the high impact scenario (green) is lower than that under the low impact scenario (orange) (Figure 4.2b, 4.2d and 4.2c)

This contrast is further confirmed through statistical analysis. The difference in mean values (high-impact minus low-impact) was positive in bare metal environments and negative in VMs (Table 4.2), directly opposing the findings from Ubuntu 24.04 (Table 4.1).

Similar behaviour was observed for 4MB and 6MB buffer sizes across both interfaces. Detailed statistical comparisons are included in the Appendix (Table A.3 and A.4).

AlmaLinux 9.4 with kernel version 5.14

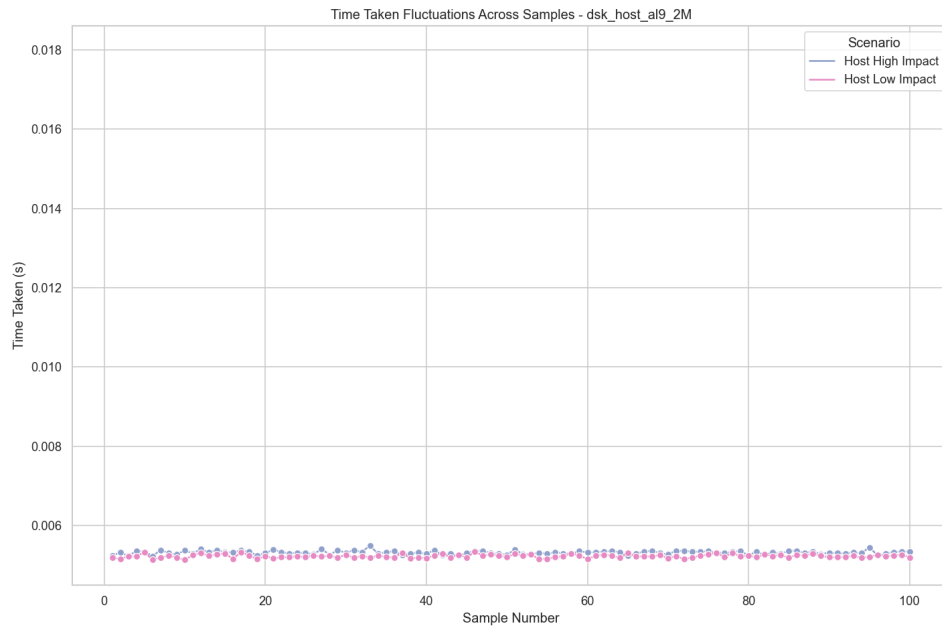
In the case of AlmaLinux 9.4, no consistent or distinguishable pattern was found in the time distributions under high and low impact scenarios when comparing virtualized and bare metal environments (Figure 4.3). The difference in mean values between the two distributions did not show a notable difference across environments (Table 4.3), indicating the absence of a detectable relationship similar to those seen in the Debian based systems.

Interface	Size	Location	Impact	Mean	Mean Difference
/dev/random	2MB	dsk-host-u22	high	0.00610495	0.00023012
			low	0.00587483	
		dsk-qemu-u22	high	0.00654526	-0.00043266
			low	0.00697791	
		dsk-vbox-u22	high	0.00639046	-0.00155774
			low	0.00794820	
		dsk-vmw-u22	high	0.01080001	-0.00063328
			low	0.01143329	

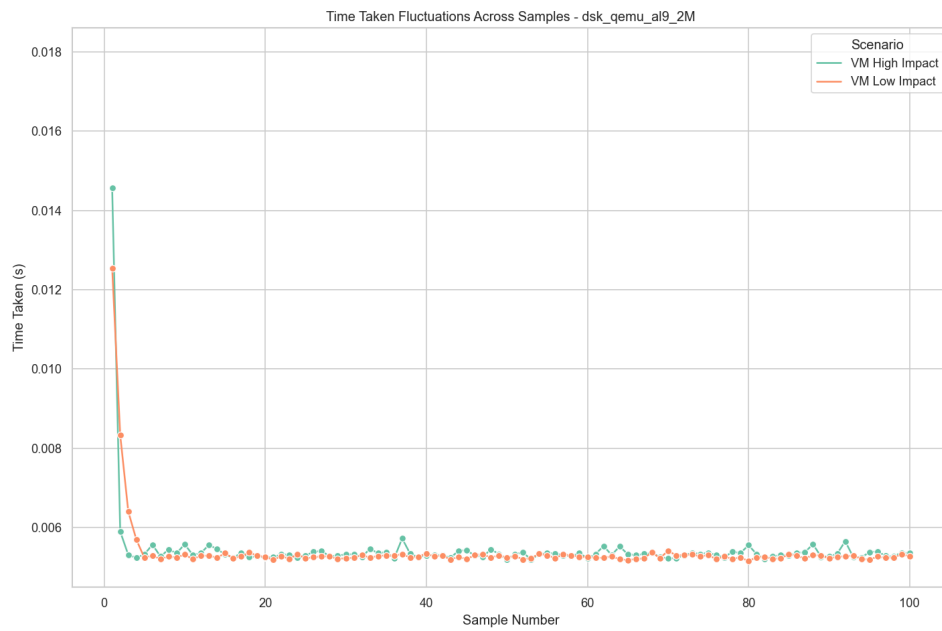
Table 4.2: Mean value differences of time taken on Ubuntu 22.04 environments

Interface	Size	Location	Impact	Mean	Mean Difference
/dev/random	2MB	dsk-host-al9	high	0.00532411	0.00009273
			low	0.00523138	
		dsk-qemu-al9	high	0.00543994	0.00005401
			low	0.00538594	
		dsk-vbox-al9	high	0.00571684	-0.00004430
			low	0.00576114	
		dsk-vmw-al9	high	0.00538035	0.00005976
			low	0.00532059	

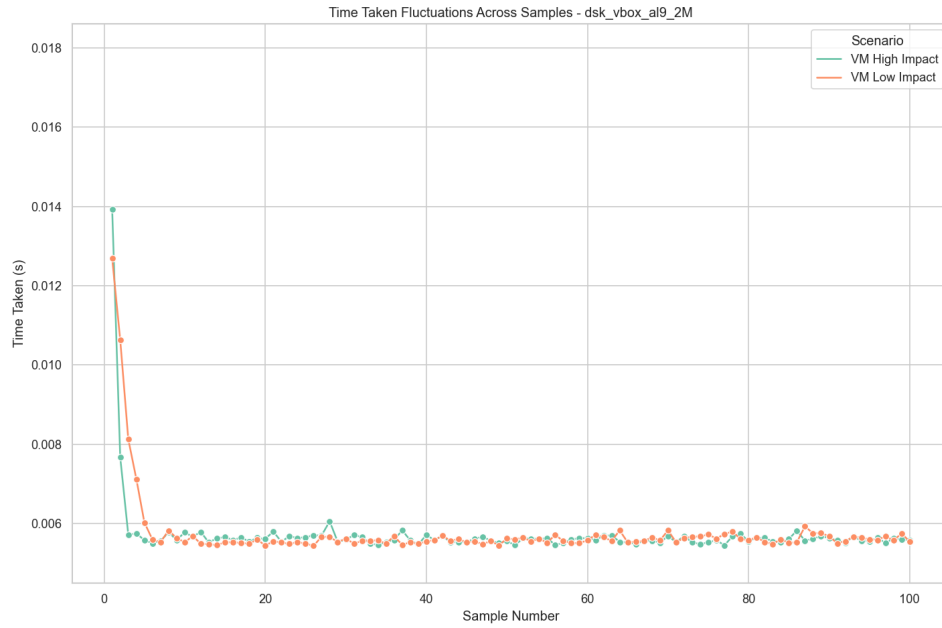
Table 4.3: Mean value differences of time taken on AlmaLinux 9.4 environments



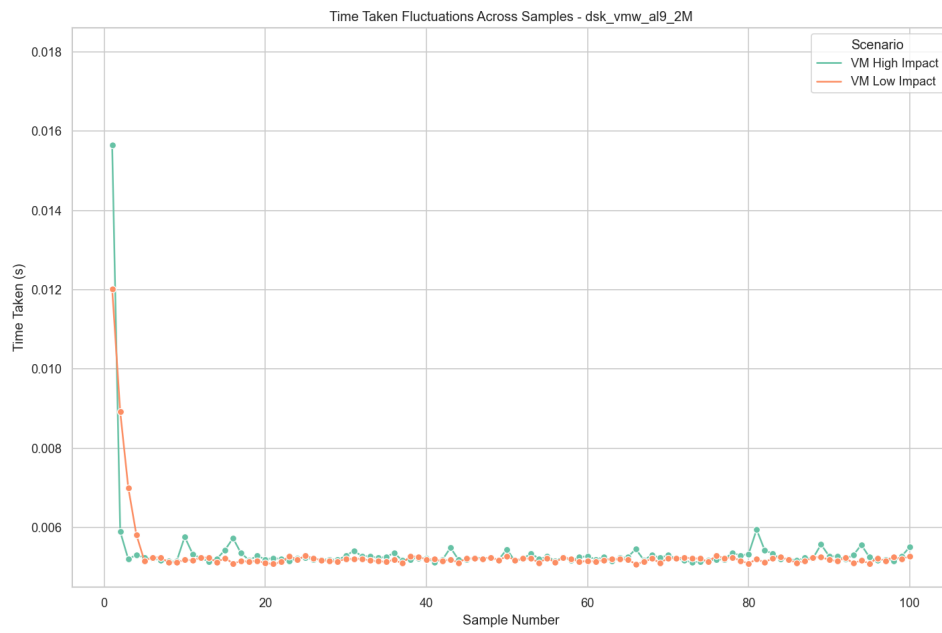
(a) AlmaLinux 9.4 on Bare Metal



(b) AlmaLinux 9.4 VM on QEMU



(c) AlmaLinux 9.4 VM on Virtual Box

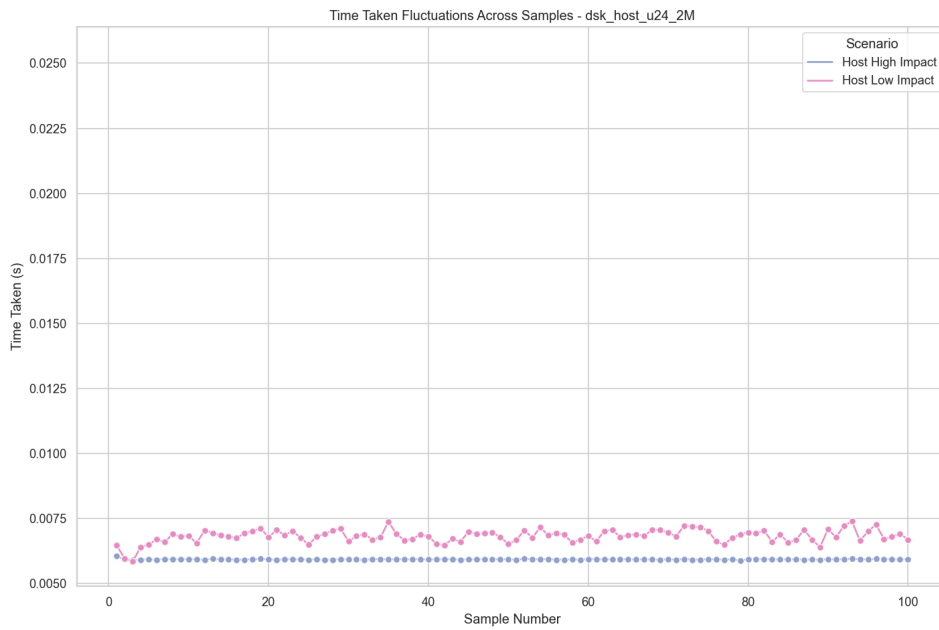


(d) AlmaLinux 9.4 VM on VMWare

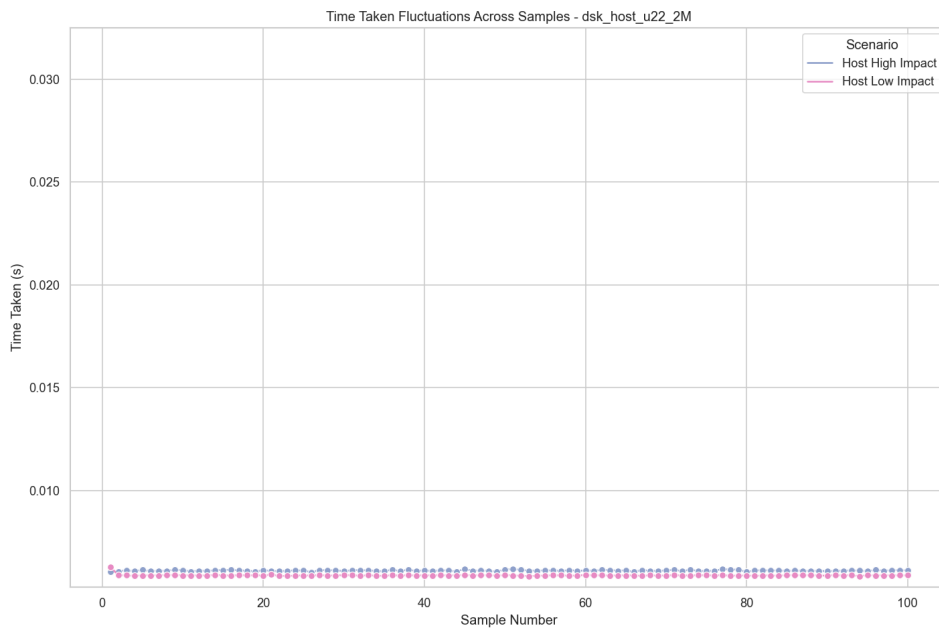
Figure 4.3: Distribution of time taken to generate random numbers under high and low impact scenarios on AlmaLinux 9.4 bare metal and VM environments

4.1.1.2 Observation 2 - Presence of an early peak in virtual environments

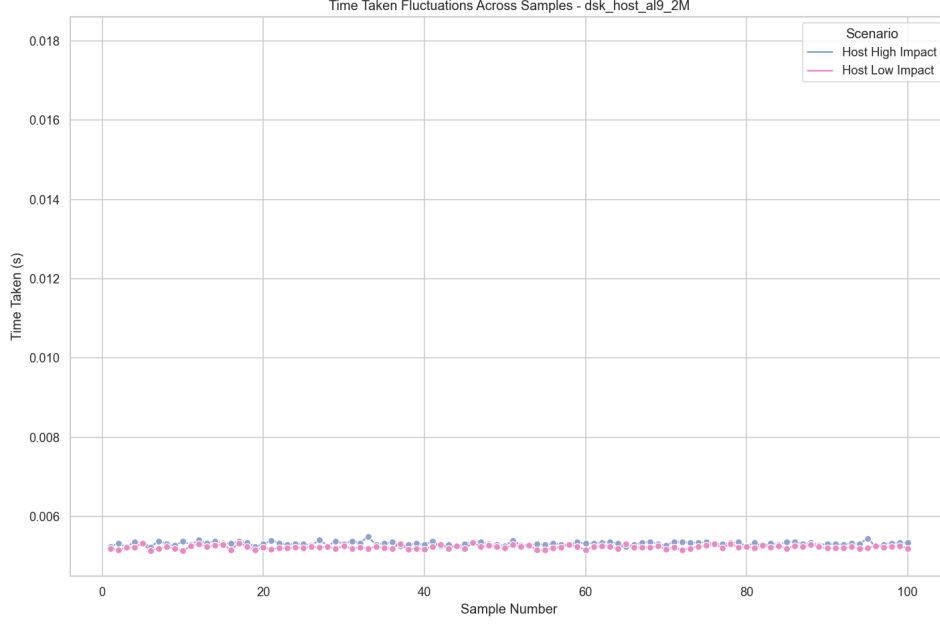
A key distinguishing behaviour observed throughout the study was the presence of an early peak in the distributions of time taken to generate random data in virtual environments (higher amount of time) as shown in the Figures 4.4d, 4.4e and 4.4f, which was not present in bare metal environments as shown in Figures 4.4a, 4.4b and 4.4c. This behaviour was consistent across all OSs used, all VMMs used, both high and low impact scenarios and all buffer sizes (2MB, 4MB and 6MB).



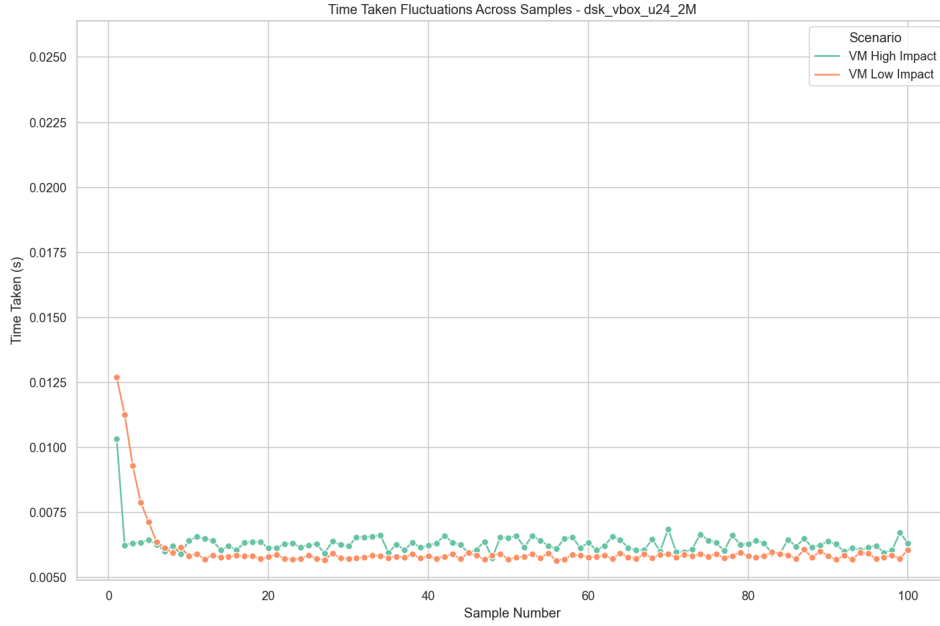
(a) Time distribution with no early peaks, Ubuntu 24.04 on Bare Metal



(b) Time distribution with no early peaks, Ubuntu 22.04 on Bare Metal

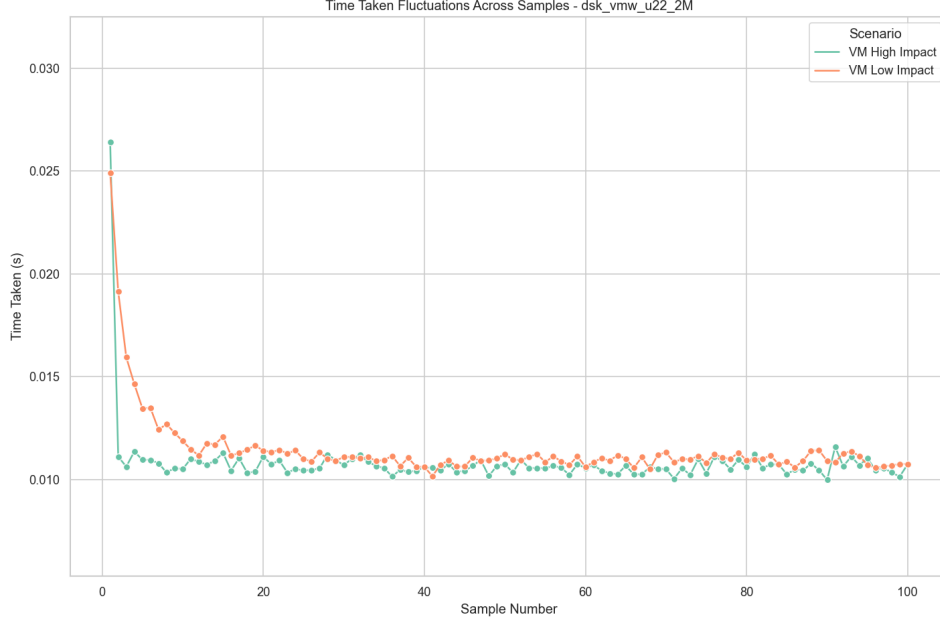


(c) Time distribution with no early peaks, AlmaLinux 9.4 on Bare Metal

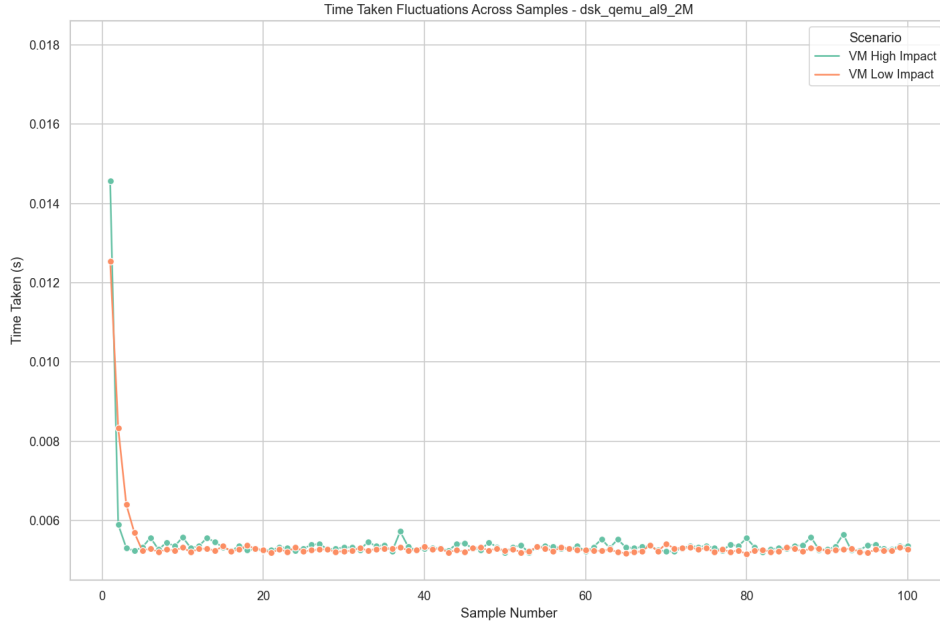


(d) Time distribution with a distinct early peak, Ubuntu 24.04 VM on VirtualBox

On closer visual examination, it was found that the early peak is within the first 8 to 10 samples in every VM and the distribution of time taken values becomes stable within first 12 samples, indicating a potential detection of virtualization. Furthermore, a pattern emerged when comparing high and low impact scenarios within a VM as well. In high impact scenarios, the peak was significantly steep, and the distribution stabilized rapidly within the first 2 to 3 samples. In contrast, low impact scenarios displayed a more gradual



(e) Time distribution with a distinct early peak, Ubuntu 22.04 VM on VMware



(f) Time distribution with a distinct early peak, AlmaLinux 9.4 VM on QEMU

Figure 4.4: Presence of early peaks in the distributions of virtual environments

decline from the peak, with the distribution stabilizing within the first 8 to 10 samples. However, this distinction was not explored further, as the primary focus remained on the presence or absence of an early peak rather than its slope or stabilization behaviour. Notably, such peaks were entirely absent in bare metal environments, highlighting the

relevance of early peak detection as a potential indicator of virtualization presence.

To capture this four statistical methods were tested, Peak-to-Mean Ratio, Z-Score for Early Peak, Early Mean Ratio and Percentage Drop After Peak (Refer Section 3.4 for details). The Percentage Drop After Peak is most effective in capturing this anomaly, as it consistently reflected the observation across all virtualized configurations (Table 4.4). Detailed results of peak detection metrics are provided in the Appendix (Section A).

Interface	Size	Location	Impact	Drop After Peak
/dev/random	2MB	dsk-host-u24	high	2.1088 %
			low	2.0196 %
		dsk-vbox-u24	high	39.6653 %
			low	54.3855 %
		dsk-vmw-u24	high	43.2342 %
			low	48.3606 %
/dev/random	4MB	dsk-host-u22	high	0.2707 %
			low	5.0286 %
		dsk-qemu-u22	high	61.7050 %
			low	41.1557 %
		dsk-vbox-u22	high	64.1066 %
			low	45.3790 %
/dev/random	6MB	dsk-host-al9	high	0.4856 %
			low	0.3203 %
		dsk-qemu-al9	high	57.1515 %
			low	52.6063 %
		dsk-vmw-al9	high	60.5417 %
			low	56.3494 %

Table 4.4: Drop after the peak percentages across multiple OSs, VMMs, and buffer sizes

Graphical Representation Note: For both observation 1 and 2 each y-axis value was averaged over ten repeated measurements due to the volume of data and to minimize visual clutter. The resulting graphs thus represent averaged values across 10 repeated runs.

4.1.1.3 Observation 3 - Differences in the dispersion across impact levels

The focus was placed on the dispersion of time measurements when 1000 samples were collected and plotted. Unlike the previous observations, all individual data points were plotted without averaging to reveal detailed patterns of variability.

Ubuntu 24.04 with kernel version 6.8

In the Ubuntu 24.04 experiments, a clear distinction was observed in bare metal environments between high and low impact conditions. Specifically, low impact scenarios showed wider dispersion in the time taken to generate random numbers (Figure 4.5b), while high impact scenarios displayed less dispersed compact distributions (Figure 4.5a). In contrast, Ubuntu 24.04 virtual environments demonstrated fairly dispersed distributions across both high and low impact conditions (Figures 4.5c and 4.5d), indicating no meaningful distinction in dispersion with the impact level.

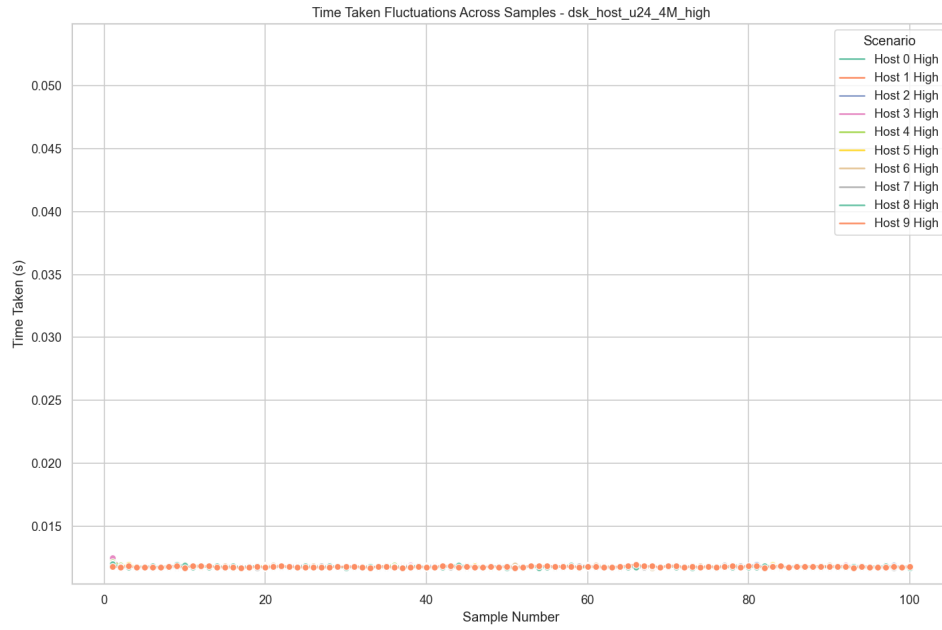
To capture and quantify this observation, statistical measures were computed including the variance, standard deviation and coefficient of variation. The ratio of each metric under low impact to high impact scenarios was calculated to capture the variations. For example, the variance under low impact was divided by the variance under high impact to determine how many times more variability was present in the low impact case. These ratios are indicators of relative dispersion, a higher ratio indicates greater variability under low impact conditions.

Interface	Size	Location	Variance Ratio (Low/High)	SSE Ratio (Low/High)
/dev/random	2MB	dsk-host-u24	385.3518	423.8643
		dsk-qemu-u24	0.9139	0.3065
		dsk-vbox-u24	1.5502	0.3195
		dsk-vmw-u24	1.4062	0.6751

Table 4.5: Variance ratio and SSE ratio of time taken to generate 2MB random numbers on Ubuntu 24.04

In Ubuntu 24.04 bare metal environments, the variance ratio was found to be relatively higher, while in virtual environments the ratio was lower reflecting uniform dispersion across both low impact and high impact scenarios (Table 4.5).

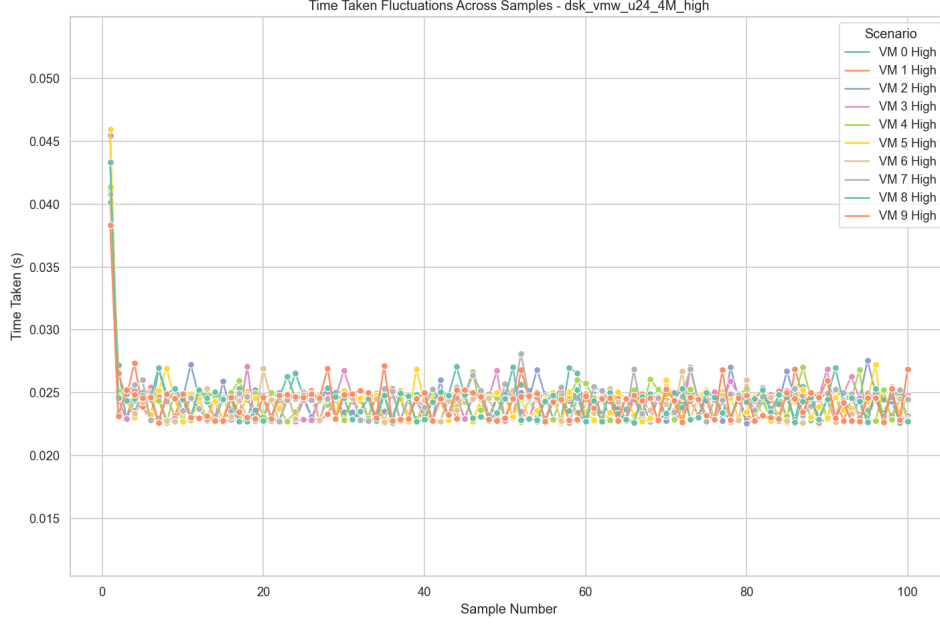
Further analysis included the computation of sum of squared error and mean squared errors metrics to identify more insights on the dispersion. The ratio of SSE between low and high impact scenarios was similarly computed. In bare-metal environments, this ratio was notably higher while in virtual environments, these values remained low and uniform. This difference emerges as another clue to detect virtualization presence in Ubuntu 24.04 systems (Table 4.5).



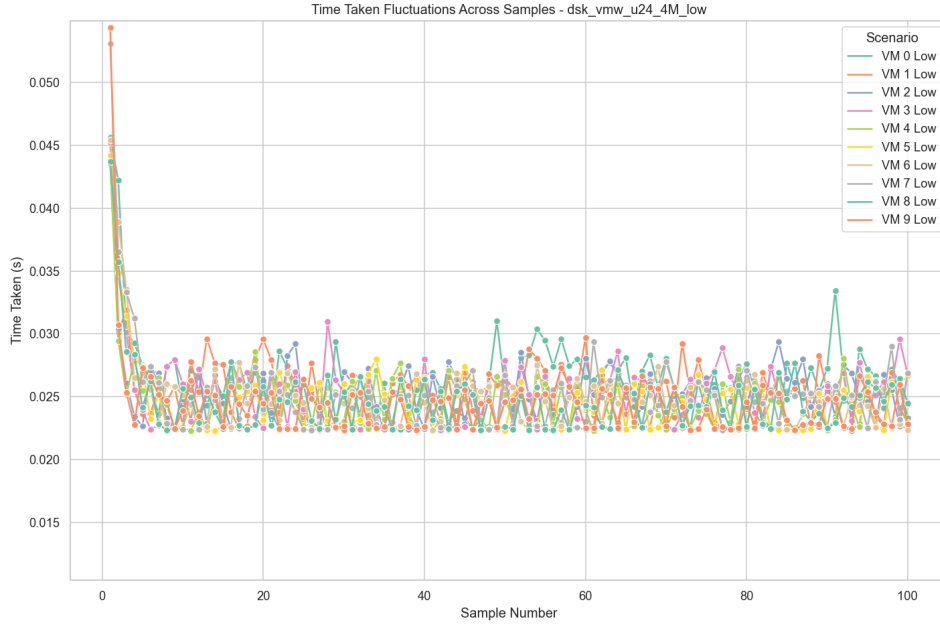
(a) Ubuntu 24.04 on Bare metal (High Impact)



(b) Ubuntu 24.04 on Bare metal (Low Impact)



(c) Ubuntu 24.04 on VMWare (High Impact)



(d) Ubuntu 24.04 on VMWare (Low Impact)

Figure 4.5: Dispersion of time taken to generate random numbers under high and low impact scenarios of Ubuntu 24.04 bare metal and VM environments

Ubuntu 22.04 with kernel version 5.15

In Ubuntu 22.04, the pattern of dispersion observed in Ubuntu 24.04 was not present. Instead, a slightly different behaviour was noted. In bare metal environments, both high impact and low impact distributions were relatively compact and less dispersed as shown in Figures 4.6a and 4.6b. And in virtual environments, both high impact and low impact

distributions were more dispersed than bare metal environments (Figure 4.6). However, the amount of dispersion did not differ distinctly between the high impact and low impact conditions.

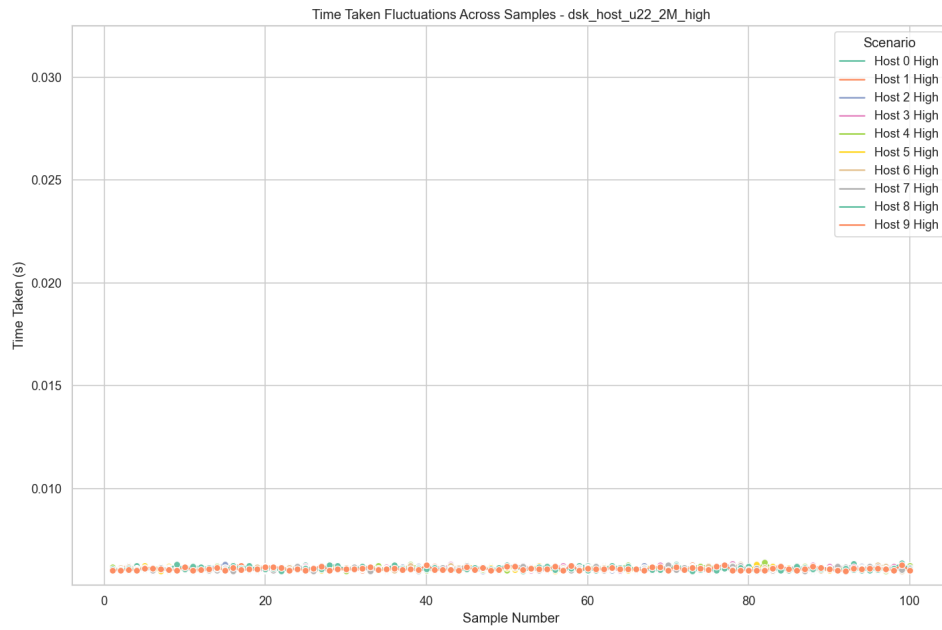
Given the limitations in performing a comparison between bare metal and VMs without a difference across impact and low impact, the analysis focused on within VM comparisons.

Upon visual inspection, a hint of difference was identified in the shapes of distribution of high impact and low impact scenarios of virtual systems. In low impact, the time values were distributed in a more irregular and dynamic pattern as shown in Figure 4.6d, while the high impact distribution followed a relatively linear fluctuation patterns as shown in Figure 4.6c. However, attempts to statistically capture this pattern through measures used in Ubuntu 24.04 (variance ratios, standard deviation, SSE) did not reveal any positive results.

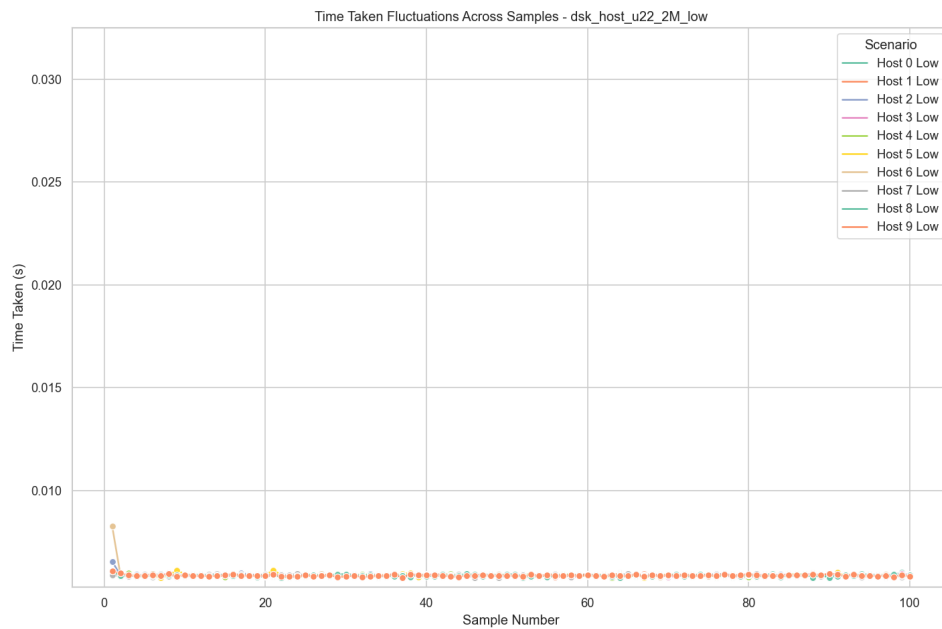
It was observed that the dispersion of low impact scenarios of virtual environments is caused by variations of skewness in the distribution of multiple rounds of the experiment (Figure 4.6d). Therefore, the skewness of each 100 sample set was calculated. Then, the mean skewness across the 10 rounds was derived for each impact condition. The ratio of low impact to high impact mean skewness for bare metal and virtual systems revealed a distinct difference. In Ubuntu 22.04 bare metal, the skewness ratio was consistently greater than 1.75 under every buffer size and interface, while the skewness ratio of virtual environments was always less than 1 (Table 4.6). This indicates a potential virtualization measure using the distribution of data in Ubuntu 22.04 systems.

AlmaLinux 9.4 with kernel version 5.14

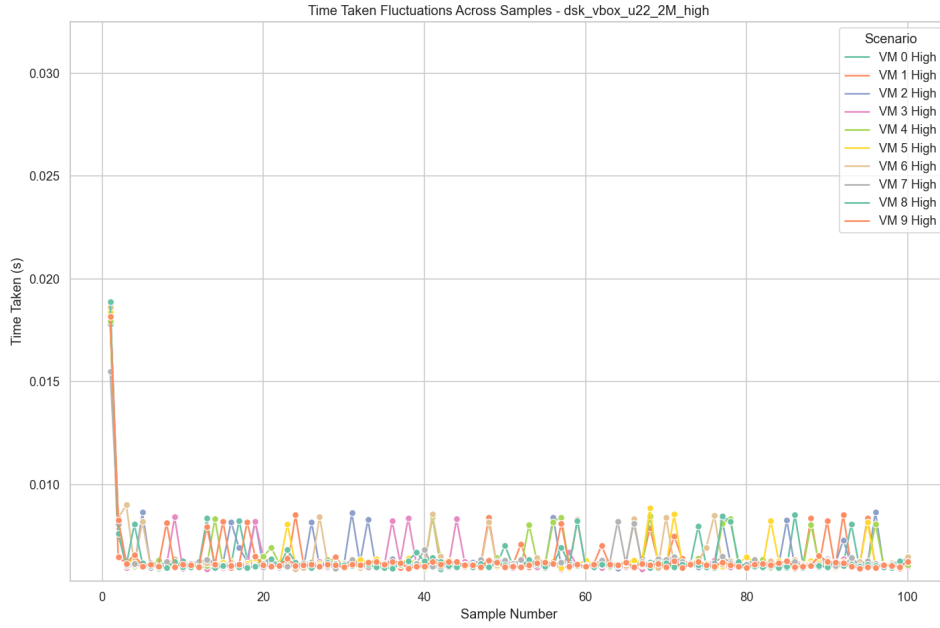
In the AlmaLinux 9.4 environment, none of the patterns observed in Ubuntu 24.04 or Ubuntu 22.04 were detected. Both bare metal and virtualized setups showed uniform dispersion patterns across impact levels. Statistical metrics and visual graphs did not indicate any significant or consistent differences that could be used to distinguish between environments.



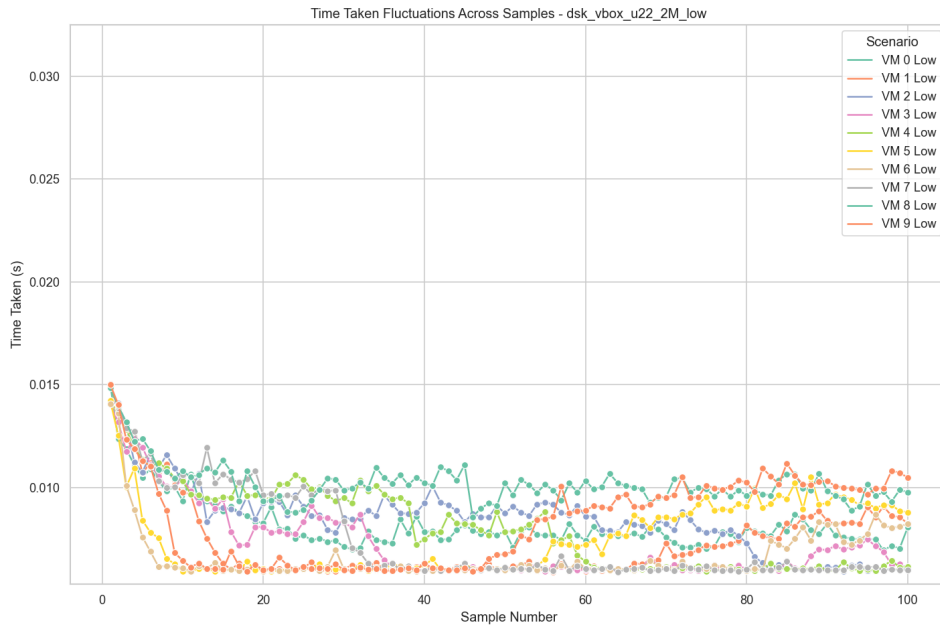
(a) Ubuntu 22.04 on Bare metal (High Impact)



(b) Ubuntu 22.04 on Bare metal (Low Impact)



(c) Ubuntu 22.04 on Virtual Box (High Impact)



(d) Ubuntu 22.04 on Virtual Box (Low Impact)

Figure 4.6: Dispersion of time taken to generate random numbers under high and low impact scenarios of Ubuntu 22.04 bare metal and VM environments

Interface	Size	Location	Impact	Mean Skewness	Mean Skew Ratio
/dev/random	2MB	dsk-host-u22	high	0.6593	3.5241
			low	2.3235	
		dsk-qemu-u22	high	4.3103	0.5941
			low	2.5607	
		dsk-vbox-u22	high	7.3972	0.2185
			low	1.6166	
		dsk-vmw-u22	high	6.4568	0.5898
			low	3.8085	
/dev/random	4MB	dsk-host-u22	high	1.1321	3.2841
			low	3.7179	
		dsk-qemu-u22	high	4.5032	0.4587
			low	2.0656	
		dsk-vbox-u22	high	7.3436	0.2753
			low	2.0217	
		dsk-vmw-u22	high	4.8114	0.7987
			low	3.8429	
/dev/random	6MB	dsk-host-u22	high	1.1470	2.3844
			low	2.7348	
		dsk-qemu-u22	high	3.2112	0.6176
			low	1.9833	
		dsk-vbox-u22	high	6.9596	0.2218
			low	1.5437	
		dsk-vmw-u22	high	3.8257	0.7152
			low	2.7363	

Table 4.6: Mean skewness ratio of time taken on Ubuntu 22.04 from /dev/random interface

4.1.2 Quality of Random Number Generation

The quality of generated random numbers was examined to investigate whether differences in entropy conditions between bare metal and virtual environments could be detected through output randomness quality. The evaluation was carried out using the NIST Statistical Test Suite, as outlined in the Section 3.4.5.

For each test instance, the percentage of samples that passed all the selected NIST tests was calculated and compared across bare metal and virtual environments under varying OSs and impact levels. Table 4.7 shows the percentage of samples that passed all NIST tests on Ubuntu 22.04 `/dev/random` interface across multiple VMMs and buffer sizes. However, this analysis did not reveal any consistent or notable patterns differentiating the VM and bare metal environments. Pass percentages for all experimental instances are given in the Appendix (Table B.1 and B.2).

Interface	OS	Size	Impact	Host	VBox	VMW	QEMU
<code>/dev/random</code>	Ubuntu 22.04	2MB	high	86	84	83	86
			low	84	88	86	88
		4MB	high	84	85	83	84
			low	85	81	85	86
		6MB	high	82	78	80	81
			low	85	82	87	88

Table 4.7: Percentage of samples passing all NIST tests on Ubuntu 22.04 `/dev/random` interface

The underlying hypothesis motivating this experiment was based on the assumption that VMs, typically having reduced access to entropy sources, might exhibit lower quality randomness compared to bare metal systems. However, the results did not support this hypothesis. Both environments demonstrated similar pass rates across the NIST tests, indicating no difference in quality attributable to virtualization.

To further investigate the presence of any correlations with the rate of random number generation, the early peak observed in Observation 2 of the rate based experiments was considered (Section 4.1.1.2). An additional analysis was conducted to examine whether the quality of randomness differed during the peak period.

As identified earlier, the early peak in time distribution consistently occurred within the first 12 samples in all virtual environments. To test for a possible correlation, the NIST quality assessment was repeated by separating the first 12 samples (early peak window) from the remainder of the data. The percentage of samples passing all core NIST tests was computed separately for the first 12 samples (during the early peak) and for the remaining samples after the peak. The Table 4.8 shows the percentage of samples that passed all NIST tests on Ubuntu 24.04 /dev/urandom interface split by the peak presence in rate of generation. Results for all experimental instances are given in the Appendix (Table B.3 and B.4).

Size	Impact	Host		VMWare	
		First12 (%)	Last88 (%)	First12 (%)	Last88 (%)
2MB	high	75.00	87.50	83.33	82.95
	low	66.67	87.50	75.00	87.50
4MB	high	91.67	79.55	91.67	78.41
	low	91.67	86.36	100.00	81.82
6MB	high	83.33	90.91	66.67	88.64
	low	83.33	85.23	75.00	81.82

Table 4.8: Percentage of samples passing all NIST tests on Ubuntu 24.04 /dev/urandom interface split by peak presence in rate of generation

The objective of this analysis was to determine whether the initially slower random number generation observed in VMs correlated with any observable variation in randomness quality. However, this did not reveal any significant difference of quality of randomness between the early peak window and the rest of the data, leading to the conclusion that the early peak phenomenon does not reflect any degradation or improvement in random number quality.

4.1.3 Evaluation of RQ1 Results

How can a program running in the user space of a Linux VM detect the underlying virtualization platform using Linux Random Number Generator and related parameters?

The findings under RQ1 suggest that virtualization creates a measurable impact on the behaviour of the LRNG, observable from user space. Detection of a virtualized environment was based on three primary observations,

1. Differences in the time distribution between high and low impact scenarios across bare metal and virtual systems
2. The presence of an early peak in time measurements in virtual environments
3. Differences in the dispersion of time measures across impact levels

Each observation contributes a distinct layer of evidence toward identifying virtual environments. A positive result under any single observation indicates potential virtualization, while the presence of multiple positive indicators offers stronger confirmation. However, it must be noted that detection parameters could not be generalized across different OS distributions and kernel versions, likely due to continual updates in the underlying random number generation algorithms and the Linux distributions workarounds on top of it.

Ubuntu 24.04 with kernel version 6.8

- Observation 1 - The time taken for high impact random number generation took longer than low impact in VMs while the time taken for high impact is lower than low impact in bare metal systems, depicting a clear inversion of results compared to VMs. This observation was quantified through the difference of mean values (mean of high impact minus mean of low impact), a positive mean gap indicates a virtual environment, while a negative value indicates a bare metal system. This pattern held consistently only with the 2MB buffer size. In 4MB and 6MB buffer sizes irregular behaviour was noticed, making 2MB the most reliable configuration for detection.

- Observation 2 - Virtual environments exhibited a consistent early peak in the first 10 - 12 samples, followed by stabilization. This behaviour was not observed in bare metal systems. The behaviour was captured using the percentage drop after peak metric. To determine a detection threshold, the minimum drop after peak among all Ubuntu 24.04 VMs was selected, from which a percentage buffer of 5% was subtracted. This percentage buffer of 5% is applied to account for minor variations and measurement noise, ensuring the classification threshold remains robust across similar instances. Accordingly, if the drop after peak of both high impact and low impact distributions exceeds the computed threshold 24.85%, the system can be classified as a VM.
- Observation 3 - Dispersion of time taken patterns revealed a distinction where in VMs, both high and low impact scenarios showed wide dispersion, while bare metal systems exhibited a clear separation. In bare metal, high impact values were tightly grouped, and low impact values showed more dispersion. This difference was quantified using the SSE ratio and the variance ratio. The detection threshold was computed by adding the percentage buffer to the maximum ratio observed in a VM environment. Accordingly, the systems with SSE ratio values below 7.3 and/or variance ratio below 2.95 are classified as virtual.

Ubuntu 22.04 with kernel version 5.15

- Observation 1 - A similar inversion of time taken pattern as Ubuntu 24.04 was noted but in reverse. In VMs, low impact data collection took longer than high impact, whereas in bare metal systems, the high impact scenario took time greater than low impact. This inversion was also measured using the difference of mean values. A negative mean gap suggests a VM and a positive gap indicates a bare metal system.
- Observation 2 - Early peaks were also present in Ubuntu 22.04 VMs. As with Ubuntu 24.04, the drop after peak value was used to capture this, and a similar method was applied to compute the detection threshold. A system showing a drop after peak of both high impact and low impact distributions greater than the calculated threshold, 37.54% is classified as a VM.

- Observation 3 - Ubuntu 22.04 VMs displayed dynamic dispersion patterns in low impact scenarios, with inconsistent skewness across experimental rounds. In contrast, high impact values were more linear. Bare metal systems showed compact distributions across both impact levels. This pattern was captured using the mean skewness of each group and the skewness ratio (low impact to high impact). A threshold was determined using the maximum VM skewness ratio plus the percentage buffer. Accordingly, the ratios below the threshold, 0.91 indicating virtualization.

AlmaLinux 9.4 with kernel version 5.14

- Observation 1 - No inversion pattern was observed. The numerical values for bare metal and VM environments were nearly indistinguishable, making this observation ineffective for virtualization detection on this OS.
- Observation 2 - Early peaks were evident in VM systems. The drop after peak threshold was computed similarly to the previous OSs. AlmaLinux 9.4 systems with both high impact and low impact distributions exceeding this threshold of 38.92% can be classified as virtualized.
- Observation 3 - No observable dispersion patterns were found. Both bare metal and virtual environments displayed uniform behaviour across impact levels, making this observation unsuitable for virtualization detection in AlmaLinux 9.4. As such, only Observation 2 can be relied upon for identifying virtual systems under this OS.

It is important to note that among the three observations, the only behaviour consistently present across all OSs was the early peak phenomenon captured in Observation 2. However, despite its presence, the corresponding metric values varied significantly across OS distributions, preventing the derivation of a generalized threshold applicable across platforms.

With regard to the quality of random numbers, as evaluated through the NIST statistical test suite, no consistent or distinguishable patterns were identified that could reliably differentiate virtual from bare metal environments. As detailed in the Section 4.1.2, both environments exhibited irregular and overlapping outcomes in test pass percentages and correlation patterns. While these findings suggest that quality based analysis may not support virtualization detection under the current methodology, further targeted investigations are required to explore its potential.

4.1.3.1 Summary of VM Detection Thresholds (Single VM Setup)

- Ubuntu 24
 - Mean of high impact - Mean of low impact > 0
 - Drop after peak percentage of both high and low impact levels $> 24.85\%$
 - SSE of low impact / SSE of high impact < 7.3
 - Variance of low impact / Variance of high impact < 2.95
- Ubuntu 22
 - Mean of high impact - Mean of low impact < 0
 - Drop after peak percentage of both high and low impact levels $> 37.54\%$
 - Mean skewness of low impact / Mean skewness of high impact < 0.91
- AlmaLinux 9
 - Drop after peak percentage of both high and low impact levels $> 38.92\%$

4.2 Results of Research Question 2

Can this detection capability be extended to an environment with multiple guest VMs on a single host?

The second research question expands the investigation to detect virtualization under multiple VMs running concurrently. This includes multi VM setups on desktop environments, the private cloud infrastructure of UCSC, and public cloud platforms (Amazon Web Services and Google Cloud Platform). The results are structured in alignment with the observations framework established in the analysis of Research Question 1.

Before presenting the results, it is necessary to acknowledge certain limitations encountered in configuring and executing experiments on cloud platforms. As described in the data collection (Section 3.3.1), each experimental instance was configured to operate under minimal usage conditions to minimize the impact of unrelated activities on the behaviour of the LRNG. This was done by disabling unnecessary background services and processes. However, on private and public cloud environments, it was not feasible to remove certain background services provided by the underlying VMM, as that risked breaking the VM functionality. As a result, some services had to remain active, unlike in the desktop based experiments.

Additionally, due to these limitations and the hardware differences, direct comparisons between cloud based VMs and bare metal systems were not accurate. Instead, consistent with the approach in Research Question 1, comparative analysis was performed using the difference between high impact and low impact scenarios within each environment. This difference-based methodology allowed relative patterns to be observed while mitigating the influence of environmental inconsistencies.

It is also important to note that in the case of Google Cloud Platform (GCP), a recurring anomaly was observed during data collection. Specifically, time measurements for random number generation would occasionally increase drastically and remain at elevated for the remainder of the data collection duration. System logs were examined to identify any anomalies or interferences at the OS level, but no evidence was found. It is suspected that certain Google Compute Engine background activities or VMM level controls, which are inaccessible from the VM, may have influenced these outcomes. Therefore, while GCP data is included in the results, its values should be interpreted with caution considering these anomalies.

4.2.1 Rate of Random Number Generation

4.2.1.1 Observation 1 - Difference of time distribution between high and low impact scenarios

Similar to Section 4.1.1.1, this section analyzes the distribution of time taken to generate random data under low and high impact scenarios.

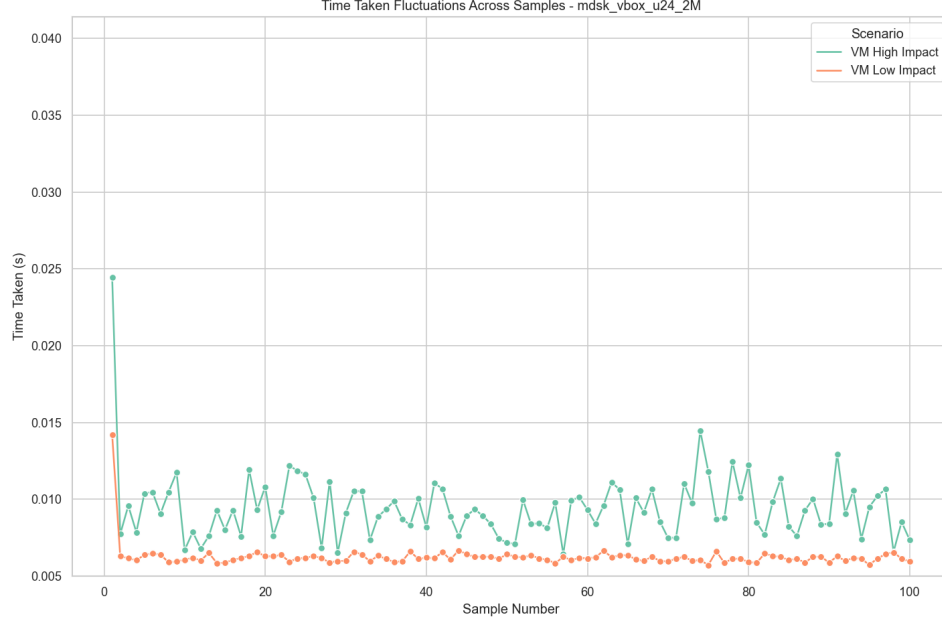
Ubuntu 24.04 with kernel version 6.8

In desktop based multi VM environments, Ubuntu 24.04 displayed a clear pattern where the mean of time taken for random number generation under high impact scenarios was consistently higher than that of low impact scenarios within all VMs (Figure 4.7a and 4.7b). This pattern was reversed in bare metal, where time taken for low impact scenarios is higher than that of high impact scenarios (Figure 4.1a). This remained consistent across both `/dev/random` and `/dev/urandom` interfaces and all buffer sizes.

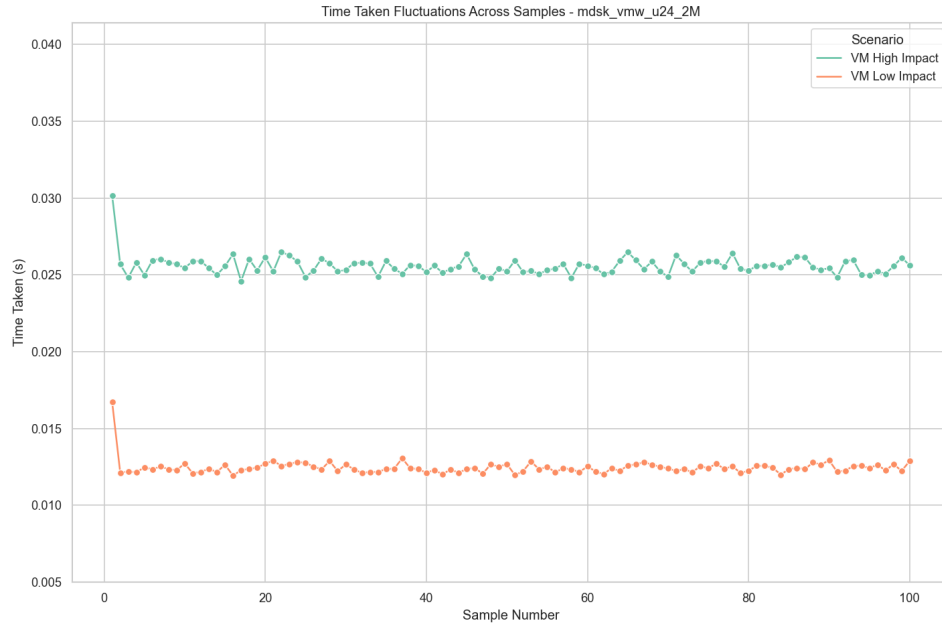
Similar findings were observed in the Ubuntu 24.04 configurations within private and public cloud environments as well. Table 4.9 shows the mean values and the difference between mean values of high impact and low impact scenarios across a selected set of multiple environments and conditions. Statistics relevant to all test instances are provided in the Appendix (Desktop : Table C.1 and C.2, Cloud : Table C.3 and C.4).

Interface	Size	Location	Impact	Mean	Mean Difference
/dev/random	2MB	dsk-host-u24	high	0.00592833	-0.00089835
			low	0.00682668	
		mdsk-qemu-u24	high	0.00718998	0.00090029
			low	0.00628969	
		mdsk-vbox-u24	high	0.00943000	0.00317012
			low	0.00625987	
		mdsk-vmw-u24	high	0.02559374	0.01313418
			low	0.01245956	

Table 4.9: Mean value differences of time taken on Ubuntu 24.04 bare metal and multi-VM environments



(a) Ubuntu 24.04 on Virtual Box (Multi-VM)



(b) Ubuntu 24.04 VM on VMWare (Multi-VM)

Figure 4.7: Distribution of time taken to generate random numbers on Ubuntu 24.04 multi-VMs

Ubuntu 22.04 with kernel version 5.15 and AlmaLinux 9.4 with kernel version 5.14

In both Ubuntu 22.04 and AlmaLinux 9.4, the inverted pattern observed in Ubuntu 24.04 was not consistently replicated. However, despite the absence of a clear trend, the gap between the mean timings for high and low impact scenarios remained higher in ma-

jority of virtual environments compared to bare metal systems in the private and public cloud setups. (Table 4.10 and 4.11). The absence of public cloud data for AlmaLinux 9.4 configuration limits broader comparison.

Interface	Size	Location	Impact	Mean	Mean Difference
/dev/random	2MB	dsk-host-u22	high	0.00610495	0.00023012
			low	0.00587483	
		mdsk-qemu-u22	high	0.00648120	-0.00040160
			low	0.00688280	
		mdsk-vbox-u22	high	0.00728570	0.00075000
			low	0.00653570	
		mdsk-vmw-u22	high	0.01471226	0.00380516
			low	0.01090710	

Table 4.10: Mean value differences of time taken on Ubuntu 22.04 bare metal and multi-VM environments

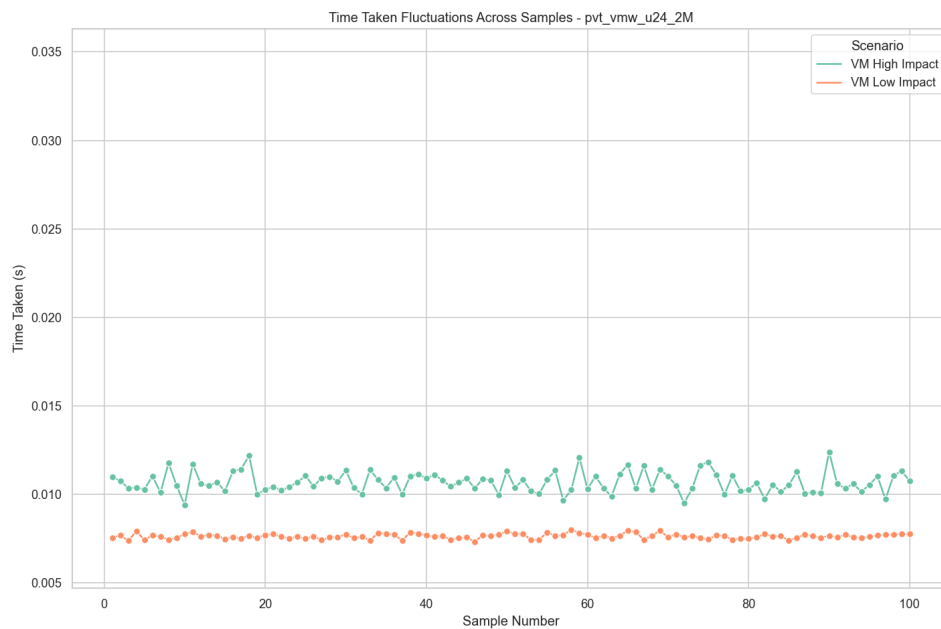


Figure 4.8: Distribution of time taken to generate random numbers on Ubuntu 24.04 cloud environments

Interface	Size	Location	Impact	Mean	Mean Difference
/dev/random	2MB	dsk-host-al9	high	0.00532411	0.00009273
			low	0.00523138	
		mdsk-qemu-al9	high	0.00584062	0.00041021
			low	0.00543041	
		mdsk-vbox-al9	high	0.00654017	0.00075510
			low	0.00578507	
		mdsk-vmw-al9	high	0.00715523	0.00176680
			low	0.00538843	

Table 4.11: Mean value differences of time taken on AlmaLinux 9.4 bare metal and multi-VM environments

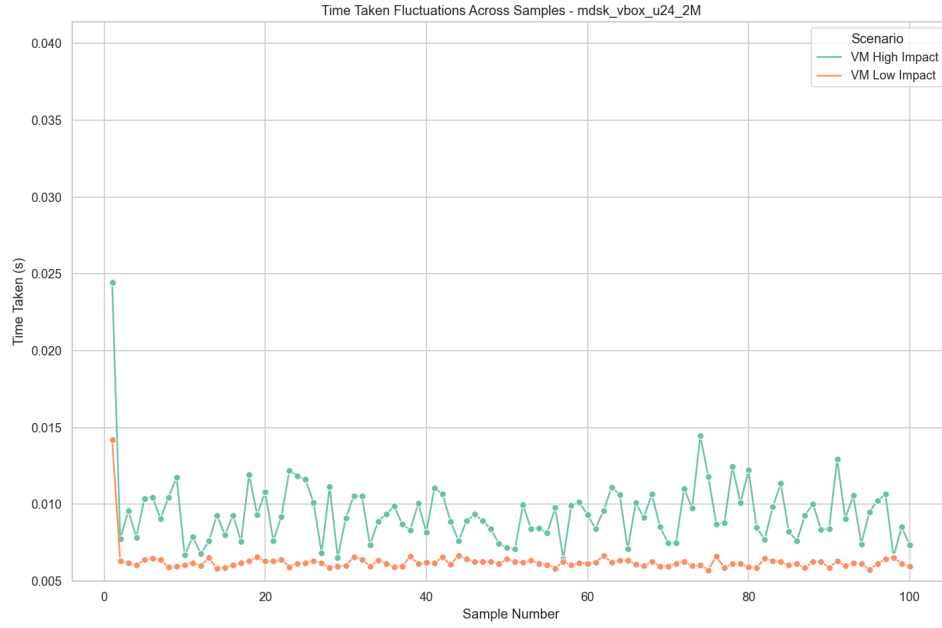
4.2.1.2 Observation 2 - Presence of an early peak in virtual environments

This examines the presence of early peaks in the time distribution across samples.

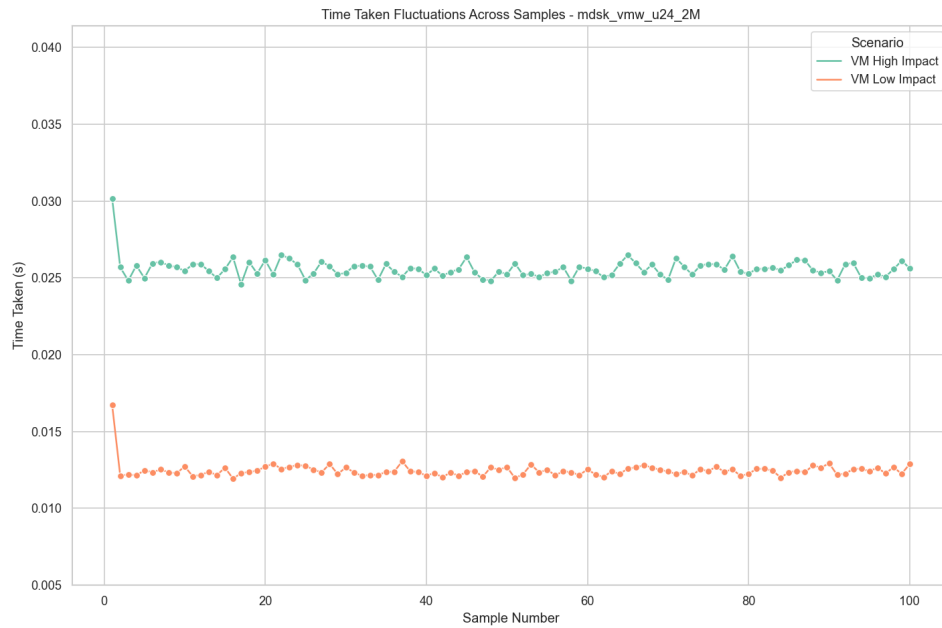
Ubuntu 24.04 with kernel version 6.8 and Ubuntu 22.04 with kernel version 5.15

In desktop multiple VM environments, early peaks were clearly observable in both Ubuntu 24.04 and Ubuntu 22.04 under all impact levels, interfaces and buffer sizes. The majority of the peaks occurred within the first 1-2 samples creating a steep peak and stabilizing the distribution quickly (Figure 4.2.1.2). Notably, in earlier experiments under Research Question 1, peak behaviour was extended across the first 8-10 samples in low impact scenarios (Figure 4.4e). However, in multiple VM context, the early peaks have become significantly steeper in both impact levels. To enable meaningful comparison, the same statistical techniques used earlier in RQ1 were applied here, with a focus narrowed to the first three samples, capturing the newly observed steep drop after the peak in time taken distribution patterns (Table 4.12 and 4.13). This early peak behaviour was not observed in the corresponding bare metal systems offering a possible distinguishing characteristic.

In contrast, VMs on private and public cloud platforms running both Ubuntu 24.04 and Ubuntu 22.04 did not display any early peaks as depicted in Figure 4.2.1.2.



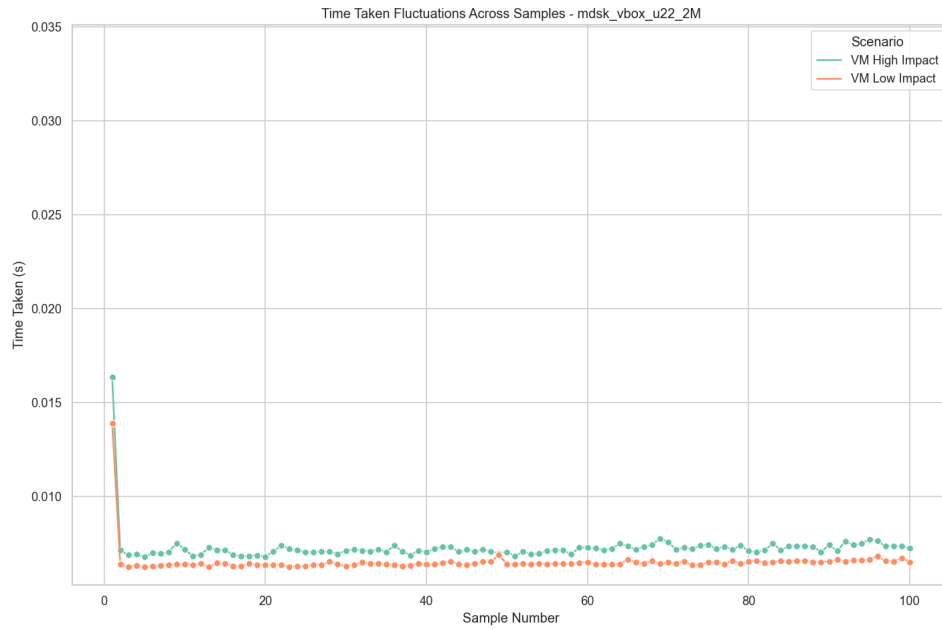
(a) Time distribution with a distinct early peak, Ubuntu 24.04 VM on VirtualBox (Multi-VM)



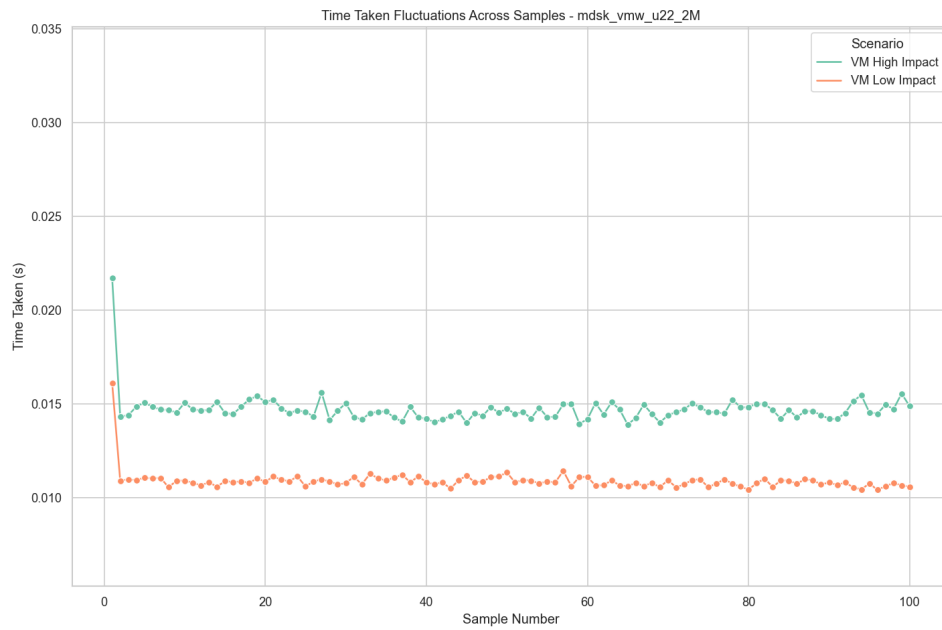
(b) Time distribution with a distinct early peak, Ubuntu 24.04 VM on VMware (Multi-VM)

AlmaLinux 9.4 with kernel version 5.14

Across both multiple VM environments, desktop and private cloud, consistent early peaks were not identified in AlmaLinux 9.4 systems as shown in Figure 4.2.1.2 which was also confirmed through the statistical measures as shown on Table 4.14.



(c) Time distribution with a distinct early peak, Ubuntu 22.04 VM on Virtual Box (Multi-VM)



(d) Time distribution with a distinct early peak, Ubuntu 22.04 VM on VMware (Multi-VM)

Figure 4.9: Presence of early peaks in the distributions of desktop virtual environments

Interface	Size	Location	Impact	Drop After Peak
/dev/random	2MB	dsk-host-u24	high	2.3671 %
			low	-1.1329 %
		mdsk-qemu-u24	high	53.1039 %
			low	44.1523 %
		mdsk-vbox-u24	high	60.8460 %
			low	55.6334 %

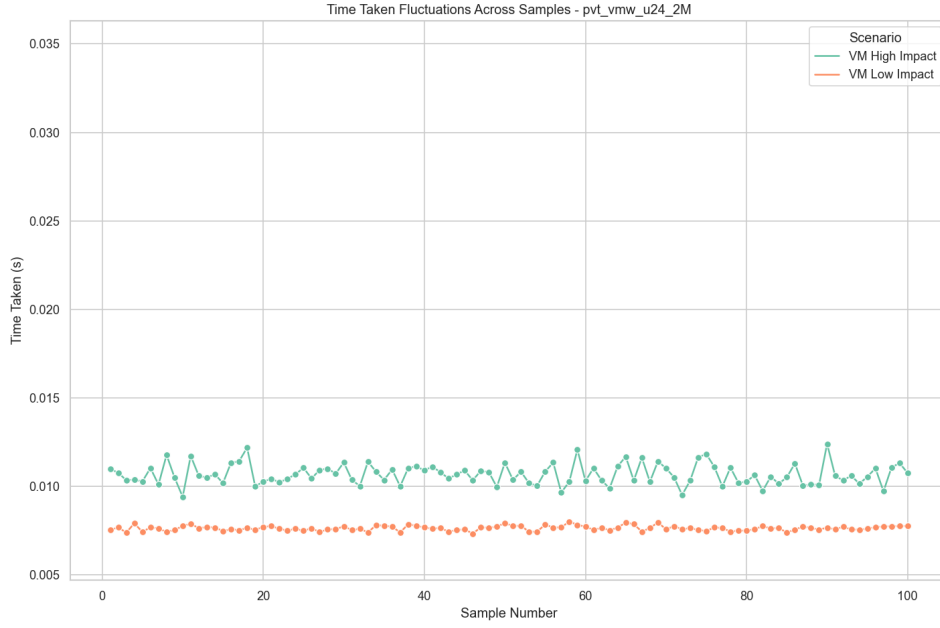
Table 4.12: Drop after the peak percentages on Ubuntu 24.04 Desktop multi-VMs

Interface	Size	Location	Impact	Drop After Peak
/dev/urandom	4MB	dsk-host-u22	high	0.1150 %
			low	6.4615 %
		mdsk-qemu-u22	high	62.4254 %
			low	40.8978 %
		mdsk-vmw-u22	high	25.9452 %
			low	29.8391 %

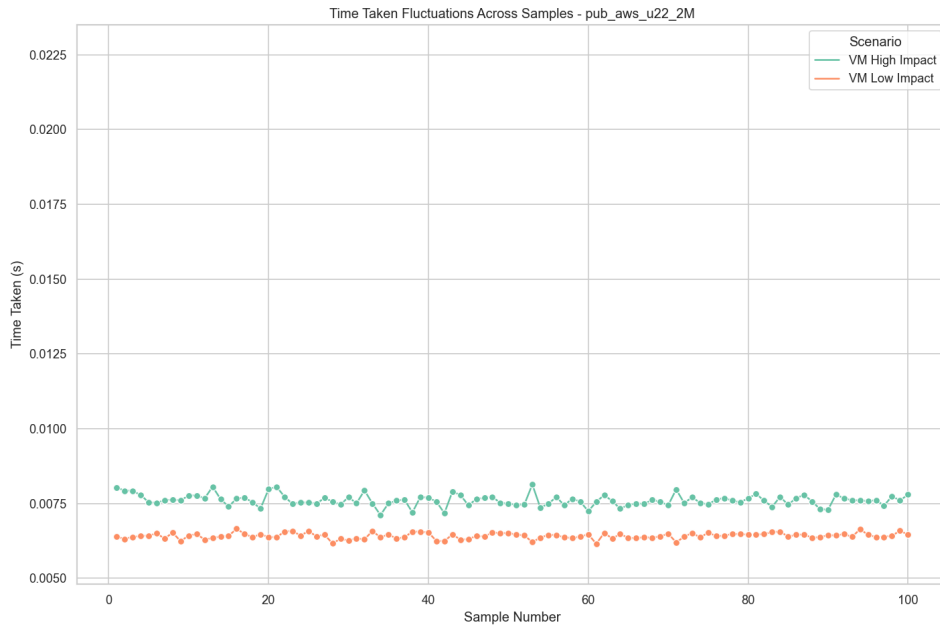
Table 4.13: Drop after the peak percentages on Ubuntu 22.04 Desktop multi-VMs

Interface	Size	Location	Impact	Drop After Peak
/dev/random	2MB	dsk-host-al9	high	1.3276 %
			low	0.1876 %
		mdsk-vbox-al9	high	0.8258 %
			low	-0.1151 %
		mdsk-vmw-al9	high	44.9819 %
			low	44.6974 %

Table 4.14: Drop after the peak percentages on AlmaLinux 9.4 Desktop multi-VM

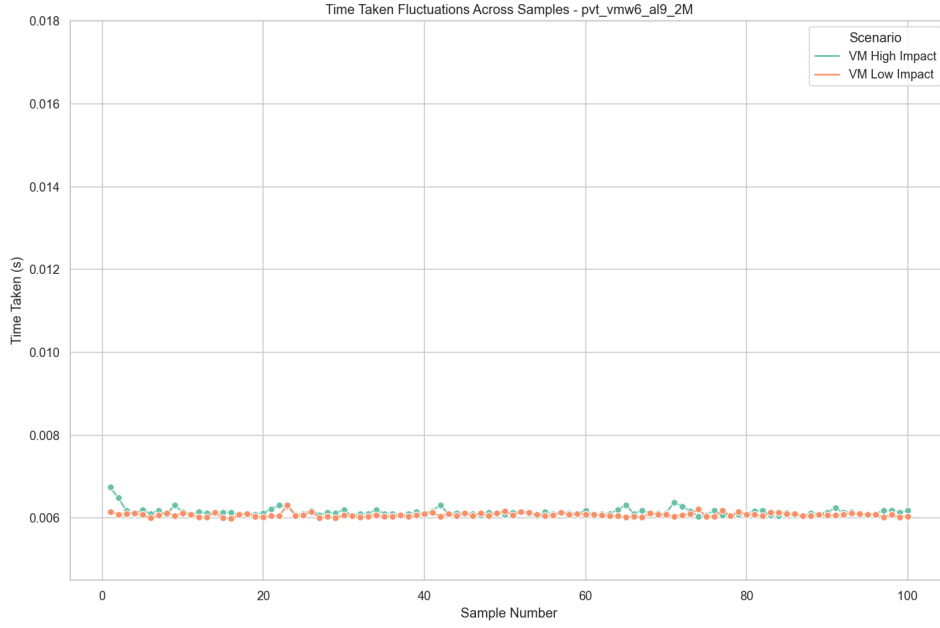


(a) Time distribution with no early peaks, Ubuntu 24.04 VM on Private Cloud



(b) Time distribution with no early peaks, Ubuntu 22.04 VM on Public Cloud (AWS)

Figure 4.10: Absence of early peaks on Ubuntu 24.04 and 22 in the distributions of private and public cloud environments



(a) Time distribution with no early peaks, AlmaLinux 9.4 VM on Private Cloud



(b) Time distribution with no early peaks, AlmaLinux 9.4 VM on Virtual Box (Multi-VM)

Figure 4.11: Absence of early peaks on AlmaLinux 9.4 in the distributions of desktop and private cloud environments

4.2.1.3 Observation 3 - Differences in the dispersion across impact levels

Ubuntu 24.04 with kernel version 6.8

In the desktop environment, Ubuntu 24.04 bare metal systems exhibited high dispersion under low impact scenarios (Figure 4.5b) and compact less dispersion under high impact scenarios (Figure 4.5a). In contrast, desktop VMs displayed a reversal of this pattern, with high impact scenarios being more dispersed than low impact ones (Figure 4.2.1.3). Private and public cloud virtual environments showed a substantially higher dispersion in high impact scenarios compared to low impact scenarios (Figure 4.2.1.3 and 4.2.1.3). These patterns were effectively statistically captured by using both SSE and variance ratios as well (Table 4.15).

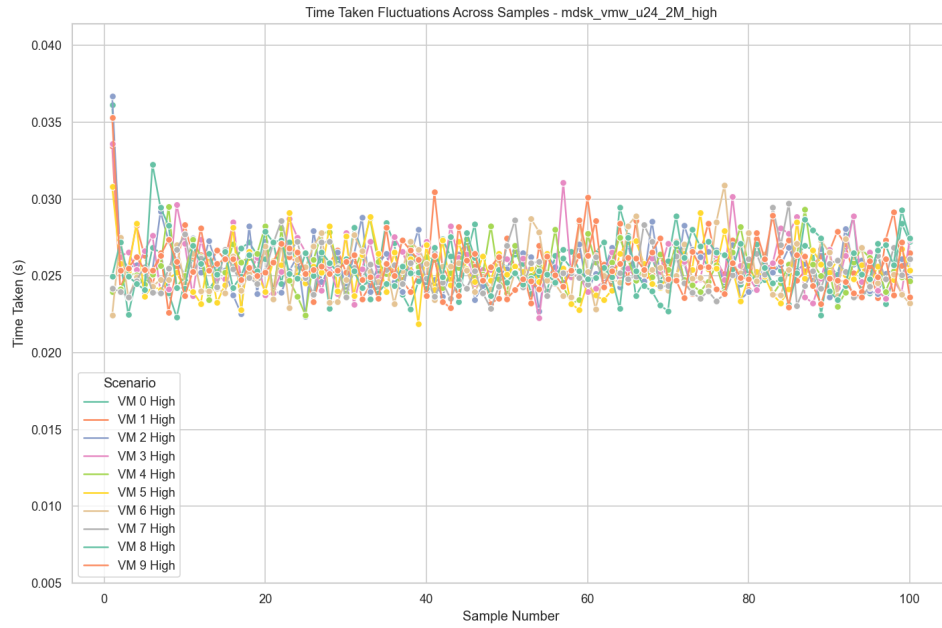
Interface	Size	Location	Variance Ratio	SSE Ratio
/dev/random	2MB	dsk-host-u24	385.3518	423.8643
		mdsk-qemu-u24	0.3085	0.1199
		mdsk-vbox-u24	0.0516	0.0296
		mdsk-vmw-u24	0.3862	0.3524
		pvt-vmw-u24	0.0657	0.0662
		pub-aws-u24	0.0387	0.0399
		pub-ggl-u24	0.0001	0.0001

Table 4.15: Variance ratio and SSE ratio of time taken to 2MB generate random numbers on Ubuntu 24.04 in multi-VM setups

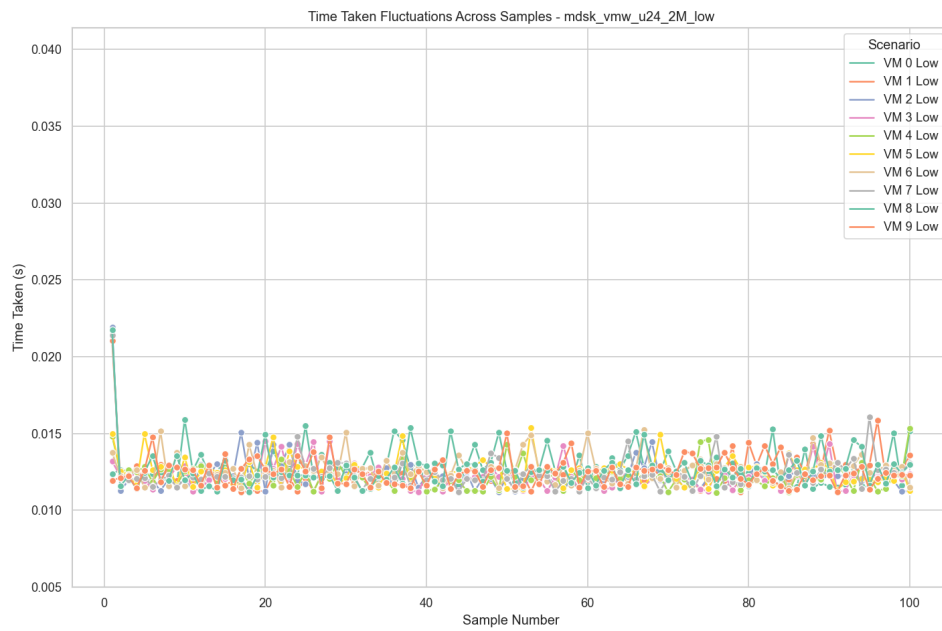
Ubuntu 22.04 with kernel version 5.15

In the desktop environment, Ubuntu 22.04 VMs were observed to be slightly more dispersed in high impact scenarios than low impact scenarios as shown in Figures 4.15a and 4.15b, while bare metal systems showed compact dispersion in both high and low impact conditions (Figure 4.6a and 4.6b). However, the SSE ratio fails to reflect this behaviour while it is captured more effectively by the variance ratio.

Furthermore, skewness analysis reveals a detectable pattern as shown in Table 4.17, though the ratios of VMs have increased compared to the single VM setup (Table 4.6).

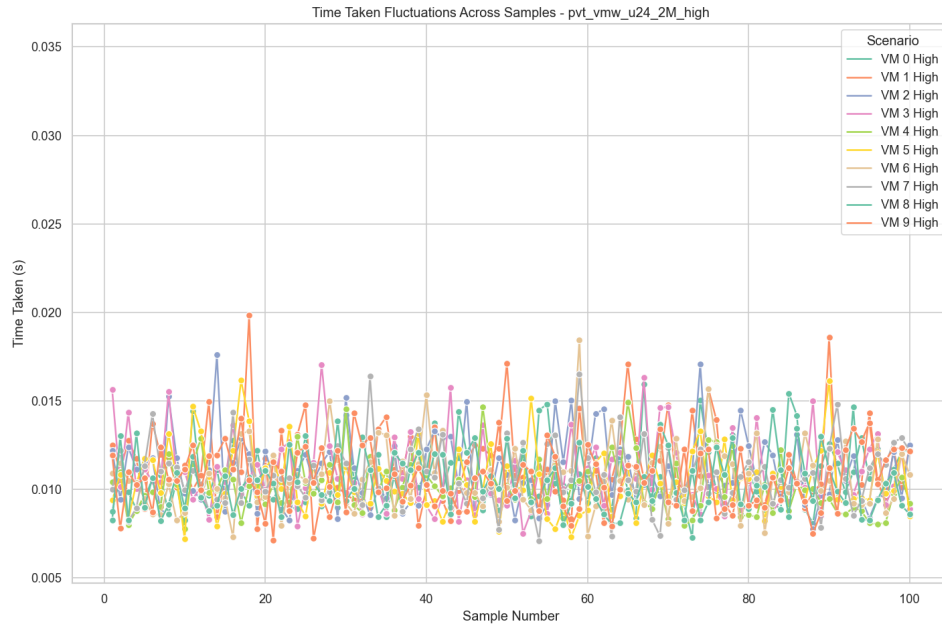


(a) Ubuntu 24.04 on VMWare (High Impact)

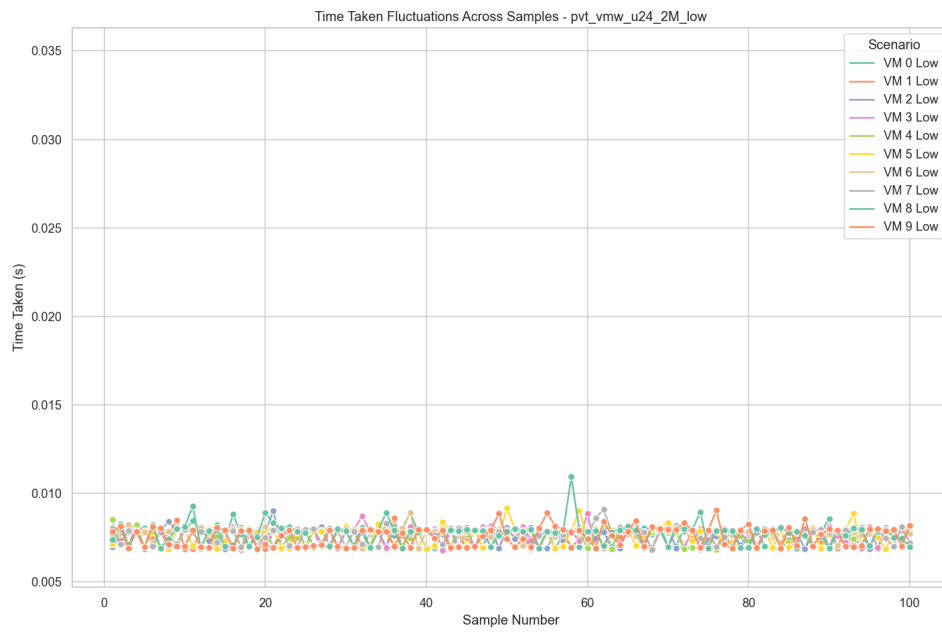


(b) Ubuntu 24.04 on VMWare (Low Impact)

Figure 4.12: Dispersion of time taken to generate random numbers on Ubuntu 24.04 multi-VM environments

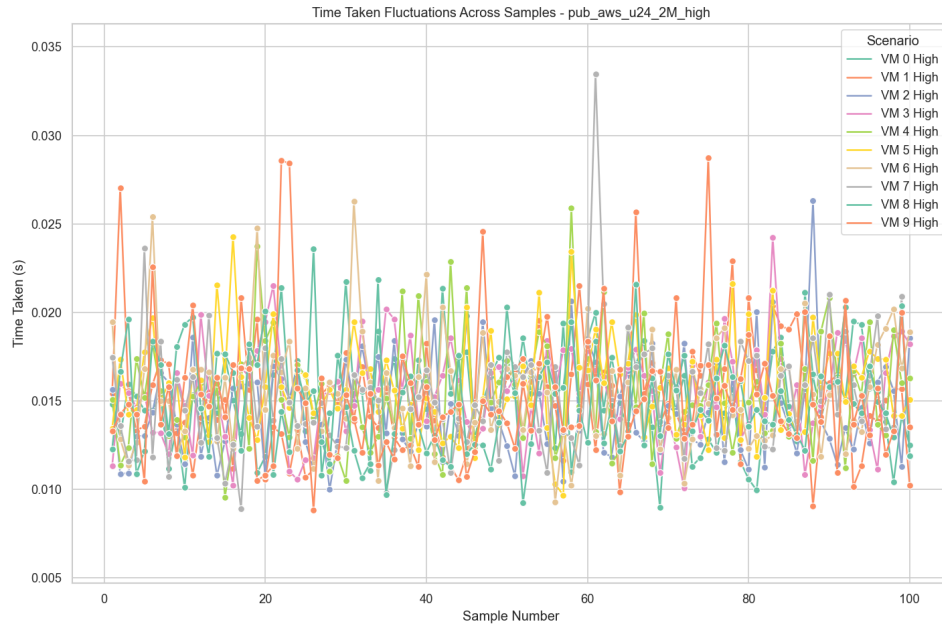


(a) Ubuntu 24.04 VM on Private Cloud (High Impact)

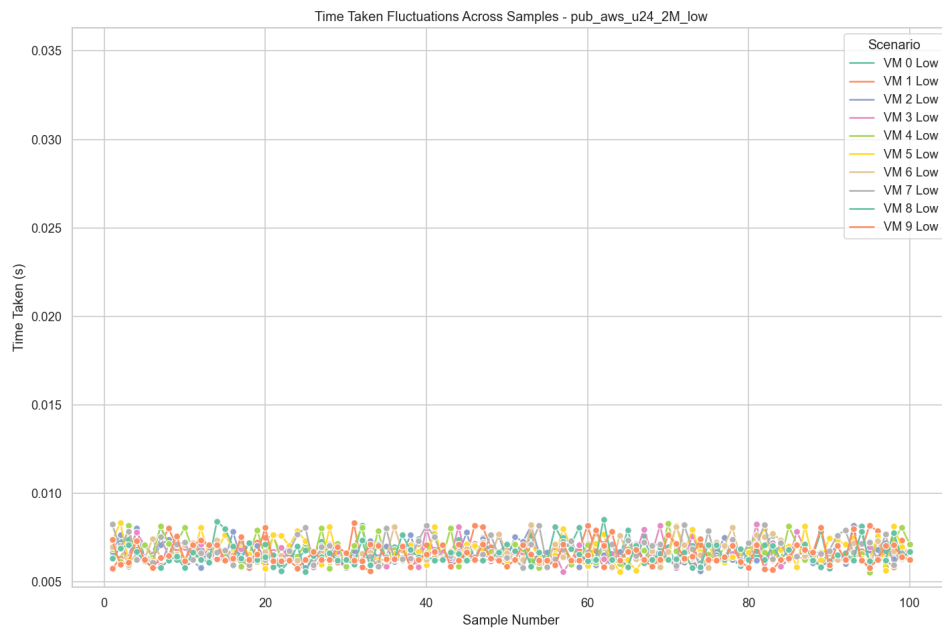


(b) Ubuntu 24.04 VM on Private Cloud (Low Impact)

Figure 4.13: Dispersion of time taken to generate random numbers on Ubuntu 24.04 private cloud environment



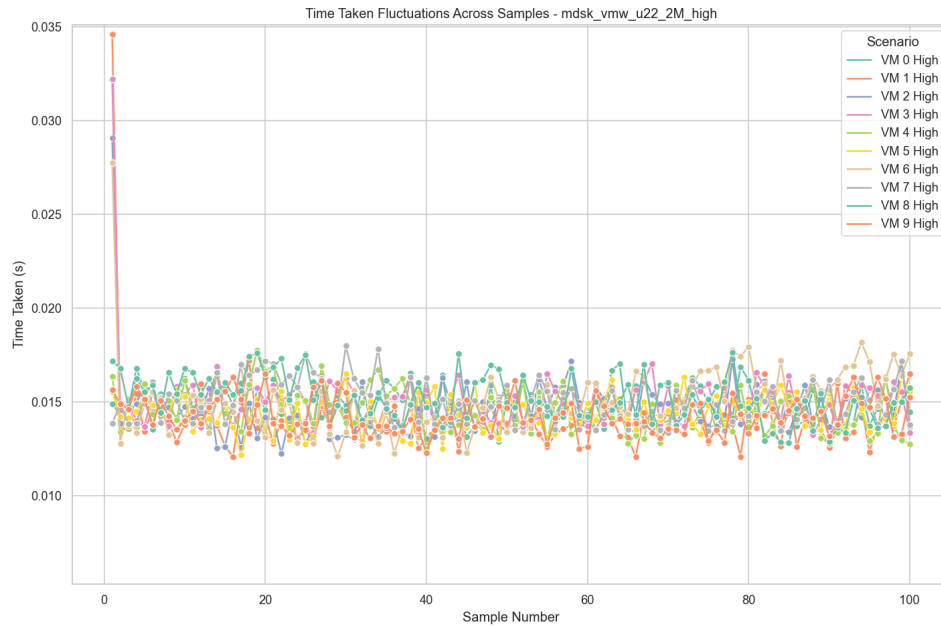
(a) Ubuntu 24.04 VM on Public Cloud (High Impact)



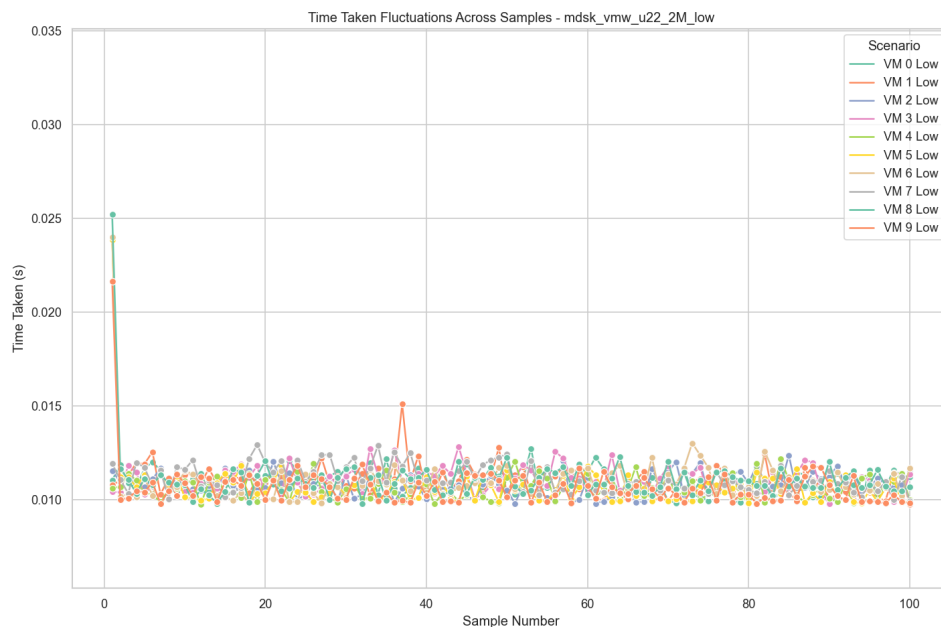
(b) Ubuntu 24.04 VM on Public Cloud (Low Impact)

Figure 4.14: Dispersion of time taken to generate random numbers under high and low impact scenarios of Ubuntu 24.04 on public cloud environment

In the private and public cloud environments, the variance ratio continued to be a valid detection parameter, capturing dispersion differences between VMs and bare metal systems in majority of the cases (Figure 4.16 and 4.17). However, this distinction was not observed for the `/dev/random` 6MB buffer size. Skewness, on the other hand, does not show a consistent separation across systems, limiting its utility in cloud contexts.

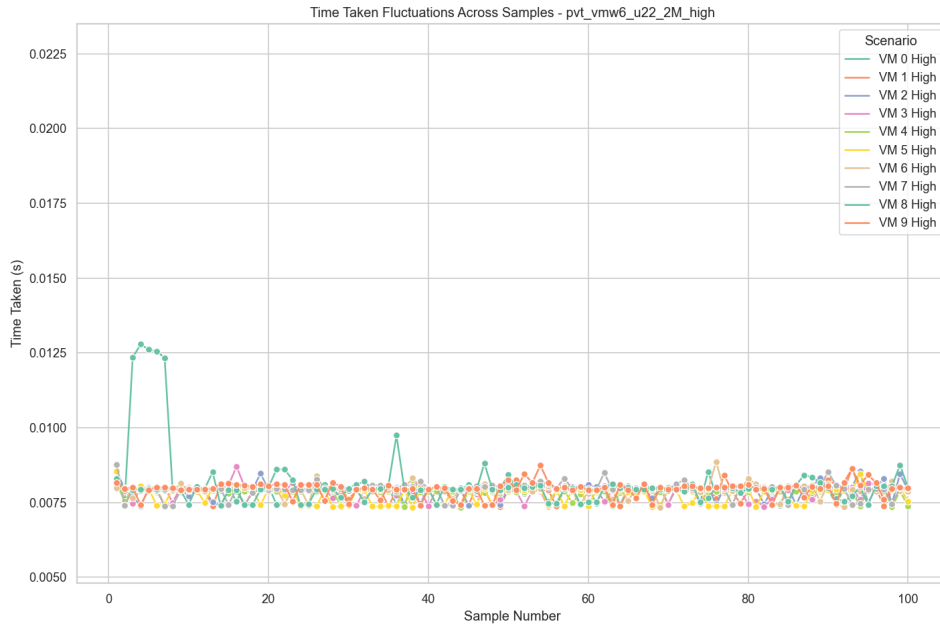


(a) Ubuntu 22.04 on VMWare (High Impact)

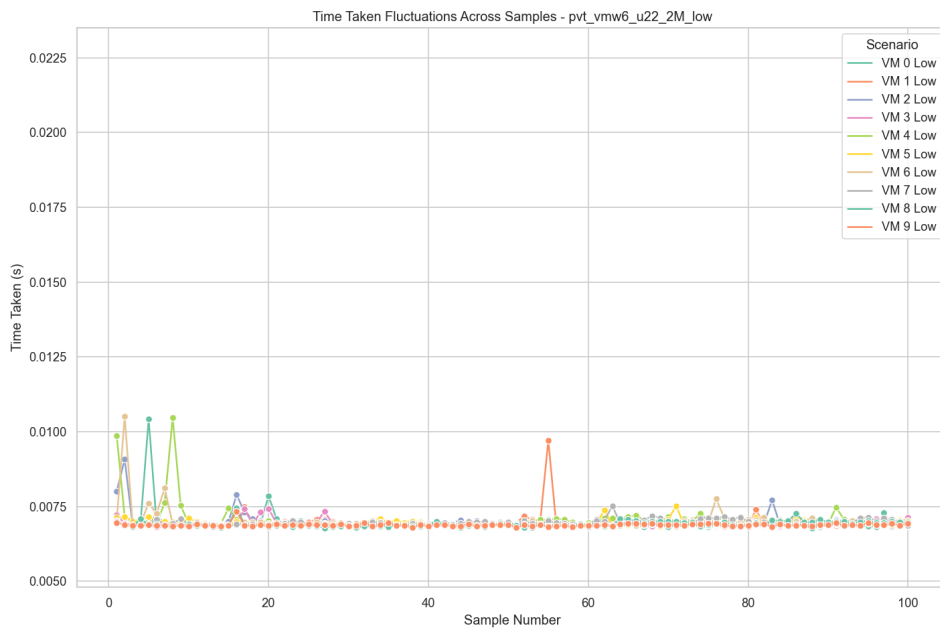


(b) Ubuntu 22.04 on VMWare (Low Impact)

Figure 4.15: Dispersion of time taken to generate random numbers on Ubuntu 22.04 on multi-VM environments

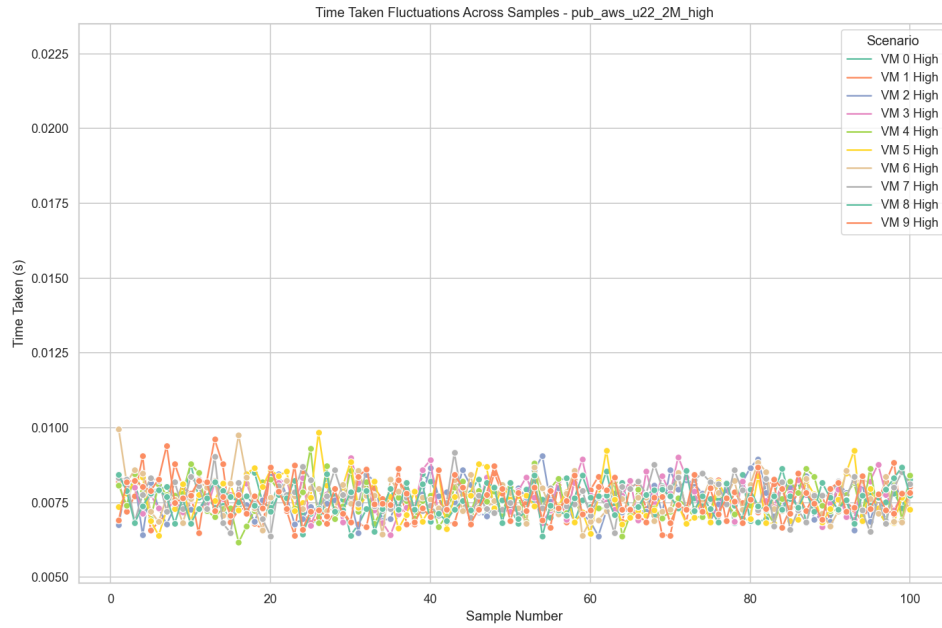


(a) Ubuntu 22.04 VM on Private Cloud (High Impact)



(b) Ubuntu 22.04 VM on Private Cloud (Low Impact)

Figure 4.16: Dispersion of time taken to generate random numbers on Ubuntu 22.04 private cloud environment



(a) Ubuntu 22.04 VM on Public Cloud (High Impact)



(b) Ubuntu 22.04 VM on Public Cloud (Low Impact)

Figure 4.17: Dispersion of time taken to generate random numbers on Ubuntu 22.04 public cloud environment

Interface	Size	Location	Variance Ratio	SSE Ratio
/dev/random	2MB	dsk-host-u22	1.3492	1.1981
		mdsk-qemu-u22	0.9714	1.2010
		mdsk-vbox-u22	0.4713	0.2216
		mdsk-vmw-u22	0.4712	0.4456
		pvt-vmw-u22	0.4592	0.4163
		pub-aws-u22	0.3486	0.3574
		pub-ggl-u22	0.6312	0.6733

Table 4.16: Variance ratio and SSE ratio of time taken to 2MB generate random numbers on Ubuntu 22.04 in multi-VM setups

Interface	Size	Location	Impact	Mean Skewness	Mean Skew Ratio
/dev/random	2MB	dsk-host-u22	high	0.6593	3.5241
			low	2.3235	
		mdsk-qemu-u22	high	5.9145	0.5887
			low	3.4818	
		mdsk-vbox-u22	high	5.7844	1.4042
			low	8.1226	
		mdsk-vmw-u22	high	2.3646	1.3981
			low	3.3058	

Table 4.17: Mean skewness ratio of time taken on Ubuntu 22.04 multi-VM

AlmaLinux 9.4 with kernel version 5.14

Across both desktop, and private cloud environments, AlmaLinux 9.4 did not display any notable patterns or trends of dispersion in high impact and low impact scenarios in VMs and bare metal systems.

4.2.2 Quality of Random Number Generation

The quality of generated random numbers was examined to determine whether virtualization influences randomness in multiple VM setups across desktop, private, and public cloud environments. The analysis followed the same procedure described in Section 3.4.5, using the NIST Statistical Test Suite to evaluate the quality from both virtual and bare metal systems.

For each instance, the percentage of samples that passed all selected NIST tests was calculated and compared across virtual and bare metal environments. However, consistent with the findings under RQ1, this evaluation did not reveal any consistent or meaningful patterns. Pass rates appeared irregular across environments and configurations, with no statistically significant advantage observed (Table 4.18).

Size	Impact	Host	VBox	VMW	QEMU	UCSC	AWS	GGLC
2MB	high	86	84	86	84	88	85	85
	low	84	88	87	83	82	83	85
4MB	high	84	81	85	73	83	85	82
	low	85	79	87	84	86	88	80
6MB	high	82	86	88	83	85	83	84
	low	85	79	77	85	79	81	80

Table 4.18: Percentage of samples that passed all NIST tests on Ubuntu 22.04 /dev/random interface in multi-VM desktop and cloud environments

To explore the possibility of a relationship between randomness quality and the rate based observations, particularly the early peak in timing noted in Observation 2, an additional analysis was conducted. This analysis was limited to the desktop environment, as neither the private nor public cloud environments demonstrated early peaks. The percentage of samples passing all tests was computed separately for the samples during the early peak and for the remaining samples after the peak. The results of this comparison

revealed no significant quality differences between the early peak window and the post peak data.

4.2.3 Evaluation of RQ2 Results

Can this detection capability be extended to an environment with multiple guest VMs on a single host?

This evaluation follows the same three observation framework established in Research Question 1, now extended to three distinct multiple VM environments, desktop, private cloud, and public cloud. As discussed prior to the results, the cloud environments presented several limitations, including restricted resource control, inconsistent kernel versions, and anomalies observed in the Google Cloud infrastructure. Furthermore, AlmaLinux was not consistently available across all platforms, further constraining uniform comparison.

Ubuntu 24.04 with kernel version 6.8

- Observation 1 - The inversion pattern observed in RQ1 was consistently replicated across all three multiple VM environments. The mean time taken for high impact scenarios was greater than that of low impact ones in virtual environments, while the reverse was evident in bare metal systems. This behaviour was statistically captured using the difference of mean values (high impact mean minus low impact mean), and a positive mean gap was indicative of a virtualized Ubuntu 24.04 system.
- Observation 2 - Early peaks were observed in the desktop multiple VM setup. These peaks were steeper than in RQ1, occurring within the first 1-2 samples. To statistically capture this, drop after peak metric was recalibrated to focus on the first 3 samples. Similar to RQ1, the thresholds were computed by subtracting the percentage buffer from the minimum drop percentage observed. Accordingly, a drop after peak values of both high impact and low impact distributions exceeding the threshold 13.70% suggests a VM. No such peaks were observed in the cloud based environments.
- Observation 3 - In virtual environments, high impact scenarios displayed higher dispersion than low impact, while bare metal showed compact distributions under

high impact and wider dispersion under low impact. This reversal behaviour was captured using SSE and variance ratios. Thresholds were calculated by adding the standard deviation to the maximum ratio observed among VMs. A system was classified as a VM if the SSE ratio is lower than 1.5 and/or variance ratio is lower than 1.25.

Ubuntu 22.04 with kernel version 5.15

- Observation 1 - No consistent inversion pattern was observed across the multiple VM environments. However, the gap between high and low impact timing measurements remained larger in the majority of virtual systems compared to bare metal environments, though not statistically conclusive for detection.
- Observation 2 - Early peaks were observed in the desktop multiple VM environment, consistent with Ubuntu 24.04, and were similarly captured via the drop after peak metric with a threshold of 19.39%. These peaks were not present in either private or public cloud setups.
- Observation 3 - In the desktop multiple VM environment, high impact scenarios displayed more dispersion than low impact ones, while bare metal systems showed compact distributions across both scenarios. This was statistically captured using variance ratio and skewness. Thresholds were defined accordingly, variance ratio below 1.1 and/or skewness ratio below the 1.73 indicate a VM.

AlmaLinux 9.4 with kernel version 5.14

- Observation 1 - No consistent or statistically distinguishable patterns were detected across any of the multiple VM environments. Although the average difference between high and low impact timings was higher in VM, this was not sufficient for classification.
- Observation 2 - Early peaks were not consistently present. Only one VM instance in the desktop environment showed early peaks, and no such behaviour was observed in cloud based instances.
- Observation 3 - No distinct patterns in dispersion were found.

Across the experiments, Ubuntu 24.04 and Ubuntu 22.04 demonstrated multiple observable patterns in multiple VM environments that enable VM detection using a combination of timing based strategies. AlmaLinux, however, remained mostly undetectable through the mechanisms applied. These results suggest the necessity for further research into Red Hat-based distributions to determine whether this limitation is due to fundamental differences in how the OS handles randomness and entropy.

As for the analysis of random number quality, results were consistent with those presented under RQ1. No clear patterns were found that could differentiate virtual and bare metal environments based on the quality of the random numbers generated. Although current evidence does not support quality based detection in these scenarios, further experimentation is required to determine its full potential.

4.2.3.1 Summary of Detection Thresholds for Multiple VM Setup

- Ubuntu 24
 - Mean of high impact - Mean of low impact > 0
 - Drop after peak percentage of both high and low impact levels $> 13.70\%$
 - SSE of low impact / SSE of high impact < 1.5
 - Variance of low impact / Variance of high impact < 1.25
- Ubuntu 22
 - Drop after peak percentage of both high and low impact levels $> 19.39\%$
 - Mean skewness of low impact / Mean skewness of high impact < 1.73
 - Variance of low impact / Variance of high impact < 1.1
- AlmaLinux 9
 - Not detectable

4.3 Results of Research Question 3

How to obscure the VM presence from user space programs running within the VM?

The third research question explores the potential for preventing virtualization detection based on LRNG related metrics. This was investigated only within the desktop VM environment, where consistent virtualization detection patterns had previously been observed. Furthermore, this phase was limited to experiments involving rate of random number generation due to the ineffectiveness of results from the quality of random number related experiments. The aim was to assess whether additional entropy sources or adjustments to system behaviour could obscure the distinguishable features that indicate virtualization presence.

In the rate based experiments, only the buffer size and random number interface demonstrating the most prominent detection capability were selected. The reason behind this choice was that any successful attempt to obscure virtualization indicators under the strongest detection conditions would implicitly extend to less sensitive configurations. Based on the results presented under Research Question 1, the configurations involving a 2MB buffer size and the `/dev/random` interface was identified as the most effective for virtualization detection. Accordingly, all experiments under this phase used this configuration.

To evaluate whether the distinguishable patterns could be effectively obscured, entropy enhancement daemons `haveged` (Wuertz and Hladky 2025) and `jitterentropy-rngd` (Müller 2022) were used, as described in the methodology Section 3.3.6.

Each observation in this phase is evaluated by comparing the statistical metrics from the entropy enhanced environment against the established threshold criteria derived from RQ1. These thresholds were identified as key indicators of virtualization presence for each OS. Accordingly, the effectiveness of entropy enhancement is determined based on whether the corresponding instances fall within or outside these threshold bounds.

4.3.1 Ubuntu 24.04 with kernel version 6.8

The experimental results for Ubuntu 24.04 reveal that the introduction of entropy enhancing daemons, `haveged`, `jitterentropy-rngd`, and their combination did not successfully obscure the virtualized nature of the instances. Despite the expected enhancement of

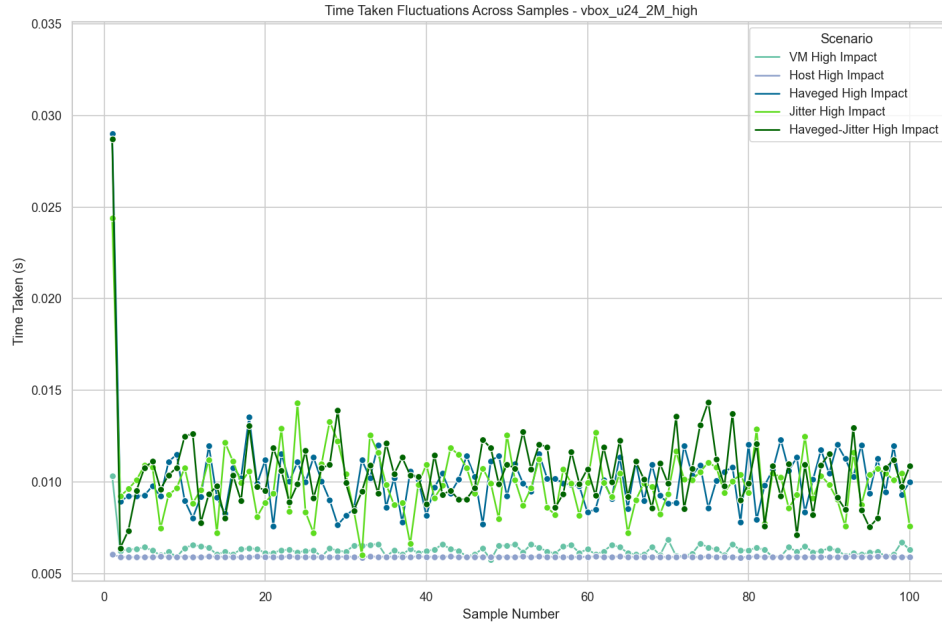
entropy availability due to these tools, key identified indicators of virtual presence such as the drop after peak, SSE ratio and variance ratio consistently fell within the thresholds previously identified for RQ 1 (Section 4.1.3.1). Tables 4.19, 4.20 and 4.21 contain the statistical measures obtained for each experiment under only **haveged** activated, only **jitterentropy-rngd** activated and both activated respectively. Specifically, the drop after peak values are greater than 24.85%, mean gaps are positive, while both the SSE ratio and variance ratio remained below their respective thresholds of 7.3 and 2.95, which were previously established as indicating virtualization presence in Ubuntu 24.04.

Furthermore, the time distributions depicted in Figures 4.18a, 4.18b, 4.19a and 4.19b indicate that random number generation with entropy enhancers have taken a higher time duration in both high impact and low impact scenarios when compared to both the typical VM behaviour observed in RQ1 and the corresponding bare metal measures.

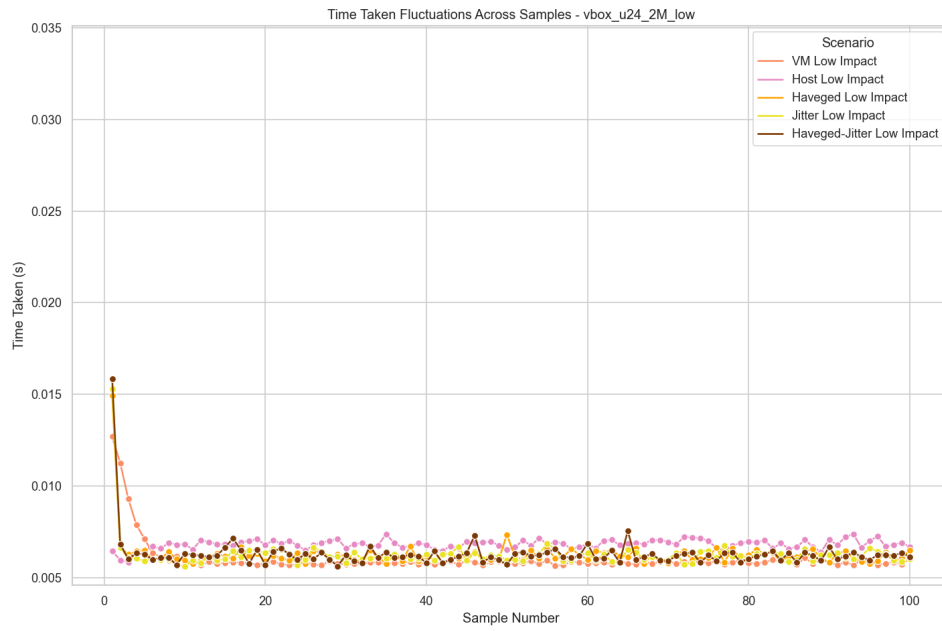
These results challenge the hypothesis that entropy enhancement could obscure virtual environments by enriching the LRNG’s input and thereby make it closer to bare metal systems. Instead, the results support the conclusion that identified virtualization detection characteristics in random number generation rates are not sufficiently obscured by simply enhancing the entropy.

Location	Impact	Drop After Peak	Mean Gap	SSE Ratio	Variance Ratio
odsk-qemu-u24	high	55.68	0.00022148	0.3460	1.7829
	low	53.56			
odsk-vbox-u24	high	63.97	0.00394860	0.0350	0.0599
	low	58.45			
odsk-vmw-u24	high	26.95	0.01288866	0.1469	0.3120
	low	43.69			

Table 4.19: Entropy Enhancement on Ubuntu 24.04 VMs only using Haveged

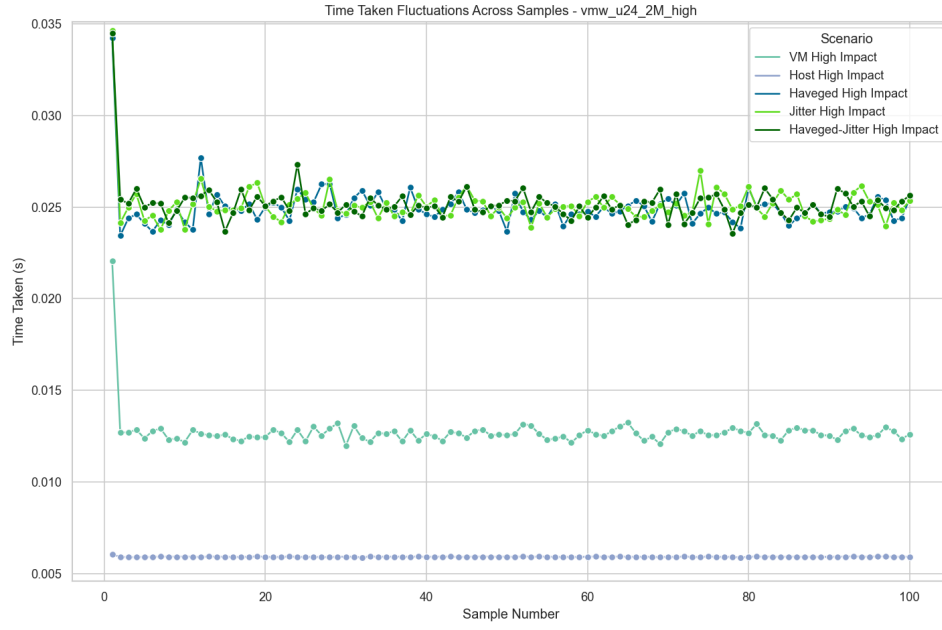


(a) Ubuntu 24.04 on Virtual Box - High Impact

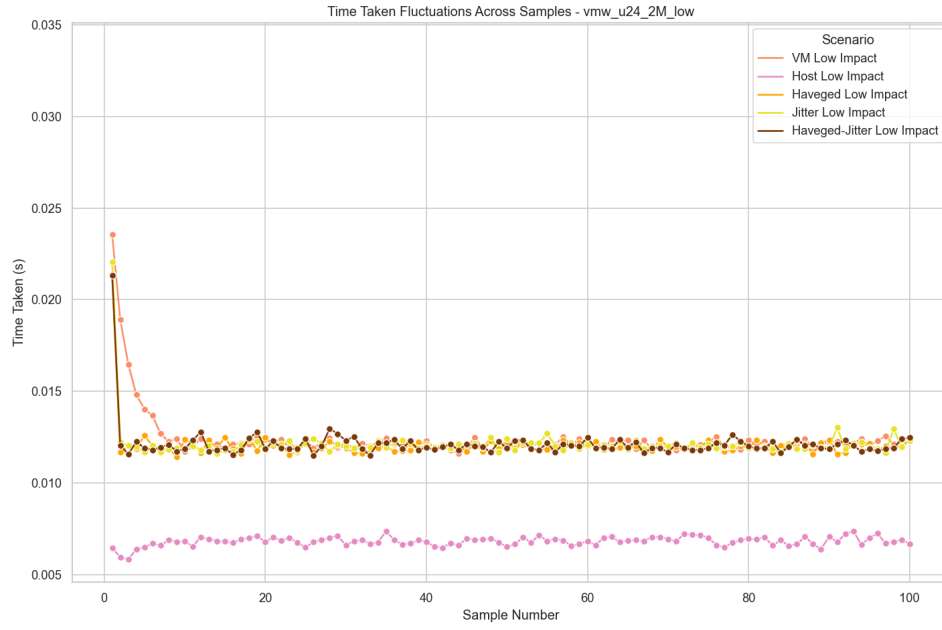


(b) Ubuntu 24.04 on Virtual Box - Low Impact

Figure 4.18: A comparison of random number generation rate curves of Ubuntu 24.04 on Virtual Box across bare metal, with and without entropy enhancing tools



(a) Ubuntu 24.04 on VMware - High Impact



(b) Ubuntu 24.04 on VMware - Low Impact

Figure 4.19: A comparison of random number generation rate curves of Ubuntu 24.04 on VMWare across bare metal, with and without entropy enhancing tools

Location	Impact	Drop After Peak	Mean Gap	SSE Ratio	Variance Ratio
odsk-qemu-u24	high	53.50	0.00019526	0.8330	1.9708
	low	52.40			
odsk-vbox-u24	high	57.51	0.00381568	0.0237	0.0544
	low	59.09			
odsk-vmw-u24	high	27.54	0.01297535	0.2113	0.3634
	low	45.64			

Table 4.20: Entropy Enhancement on Ubuntu 24.04 VMs only using Jitterentropy

Location	Impact	Drop After Peak	Mean Gap	SSE Ratio	Variance Ratio
odsk-qemu-u24	high	56.59	0.00045410	0.1449	1.4677
	low	57.23			
odsk-vbox-u24	high	65.12	0.00420792	0.0411	0.0685
	low	60.11			
odsk-vmw-u24	high	26.54	0.01304665	0.2226	0.3643
	low	43.80			

Table 4.21: Combined Entropy Enhancement on Ubuntu 24.04 VMs using Haveged and Jitterentropy

4.3.2 Ubuntu 22.04 with kernel version 5.15

The results observed for Ubuntu 22.04 similarly indicate that entropy enhancement techniques that were used were not effective in obscuring the virtualization detection of the systems. Across all three virtualization platforms the key classification indicator, drop after peak exceeded the threshold of 37.54% established for virtual environments (Tables 4.22, 4.23, and 4.24). This metric remained the most prominent and consistent distinction between virtualized and bare metal systems, and none of the enhanced configurations were able to bring this value below the virtual threshold.

However, some instances showed partial obfuscation with respect to other indicators such as mean gap values occasionally being positive or skewness ratios exceeding the

VM threshold. However, these measures cannot be considered independently without the most prominent detection method, drop after peak. The time distribution plots in Figures 4.20a through 4.21b further support these findings. The application of entropy enhancing tools appeared to amplify the patterns in timing behaviour associated with VMs, rather than aligning them with the bare metal patterns observed in RQ1.

Location	Impact	Drop After Peak	Mean Gap	Skewness Ratio
odsk-qemu-u22	high	65.14	-0.00040983	0.5872
	low	53.30		
odsk-vbox-u22	high	65.04	0.00068079	1.0715
	low	58.13		
odsk-vmw-u22	high	52.08	0.00366671	1.2075
	low	53.66		

Table 4.22: Entropy Enhancement on Ubuntu 22.04 VMs only using Haveged

Location	Impact	Drop After Peak	Mean Gap	Skewness Ratio
odsk-qemu-u22	high	67.07	-0.00181154	0.3358
	low	39.69		
odsk-vbox-u22	high	63.28	0.00064448	1.0593
	low	60.94		
odsk-vmw-u22	high	54.04	0.00355165	1.1555
	low	53.56		

Table 4.23: Entropy Enhancement on Ubuntu 22.04 VMs only using Jitterentropy

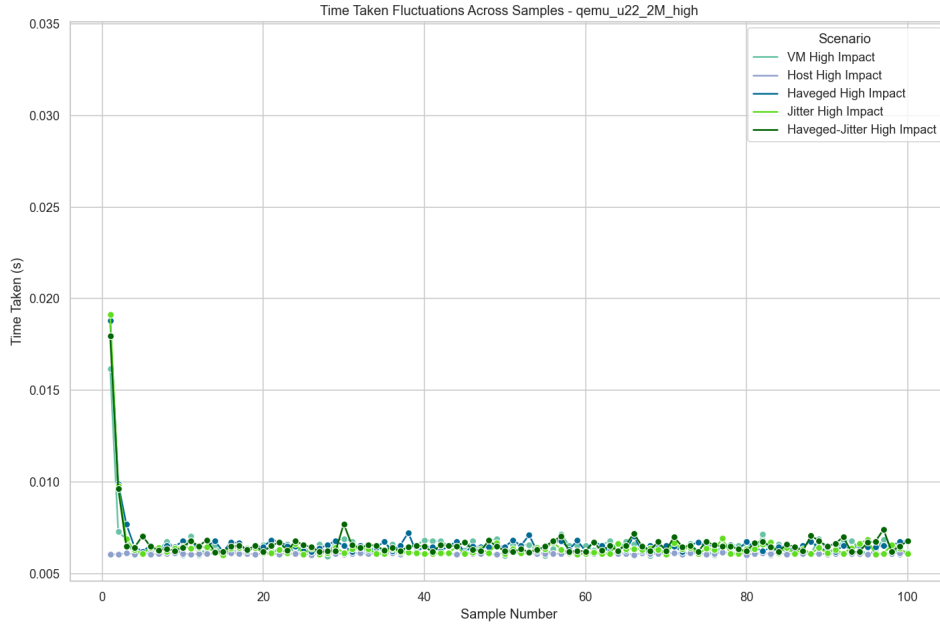
Location	Impact	Drop After Peak	Mean Gap	Skewness Ratio
odsk-qemu-u22	high	63.98	-0.00036218	0.6375
	low	54.28		
odsk-vbox-u22	high	62.93	0.00057375	1.0518
	low	62.57		
odsk-vmw-u22	high	53.48	0.00375932	1.2167
	low	54.89		

Table 4.24: Combined Entropy Enhancement on Ubuntu 22.04 VMs using Haveged and Jitterentropy

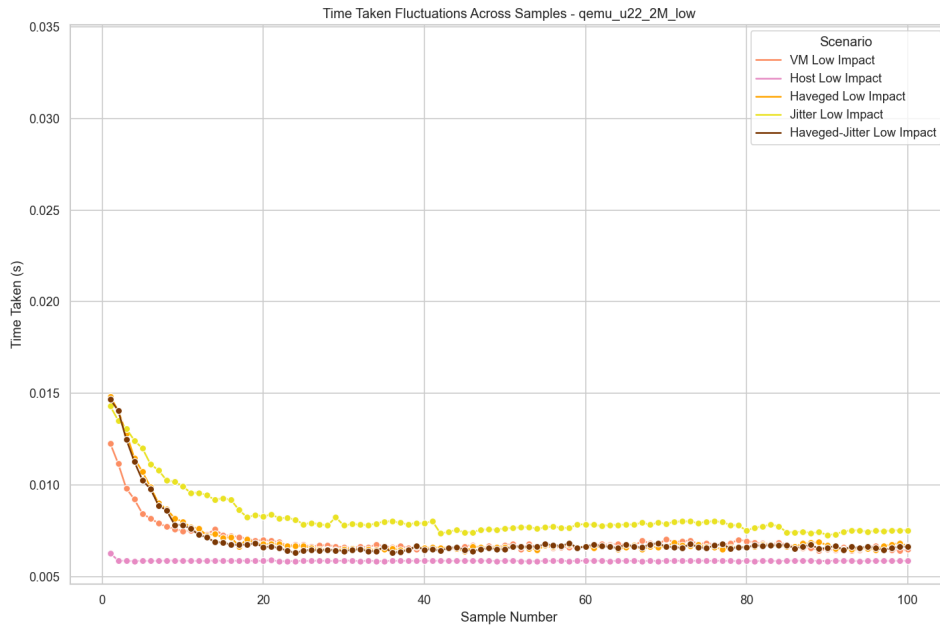
4.3.3 AlmaLinux 9.4 with kernel version 5.14

The evaluation of entropy enhancement tools on AlmaLinux 9.4 reveals that the only detection indicator on AlmaLinux 9.4 “drop after peak” remained consistently above the VM threshold of 38.92% across all virtualized setups. As shown in Tables 4.25, 4.26, and 4.27, this metric did not fall below the threshold in any of the configurations, irrespective of the entropy tool used or the impact level applied. This consistent pattern indicates that none of the entropy enhancement methods were successful in hiding the virtualization presence.

The distribution curves presented in Figures 4.22a and 4.22b displays this conclusion. The virtualized systems under all three platforms, QEMU, VirtualBox, and VMware continued to exhibit steep declines in generation rate after the initial peak, even after entropy enhancements were introduced. These curves closely align with observations in RQ1 further confirming the ineffectiveness of entropy enhancing tools for virtualization obfuscation.

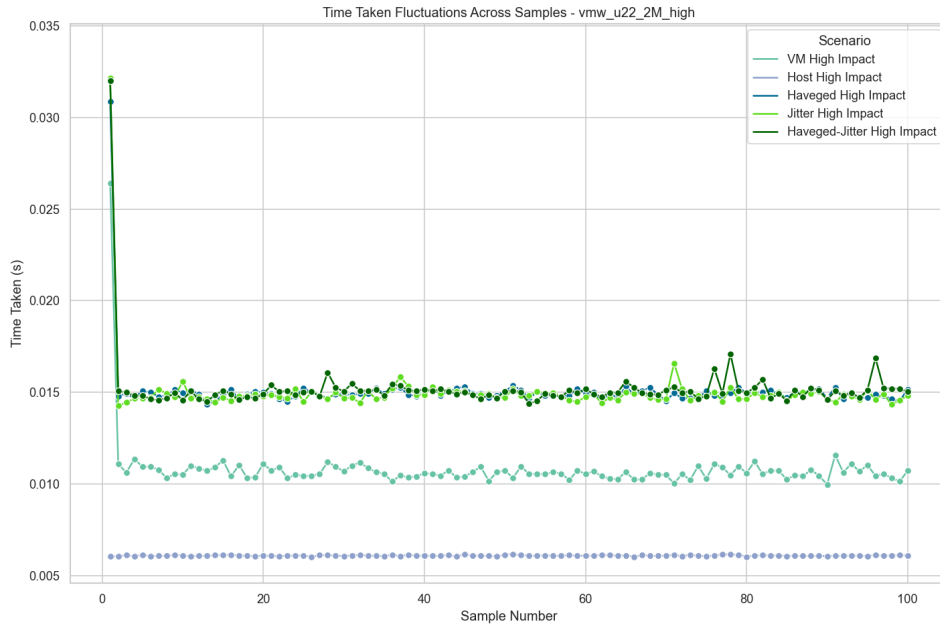


(a) Ubuntu 22.04 on QEMU - High Impact

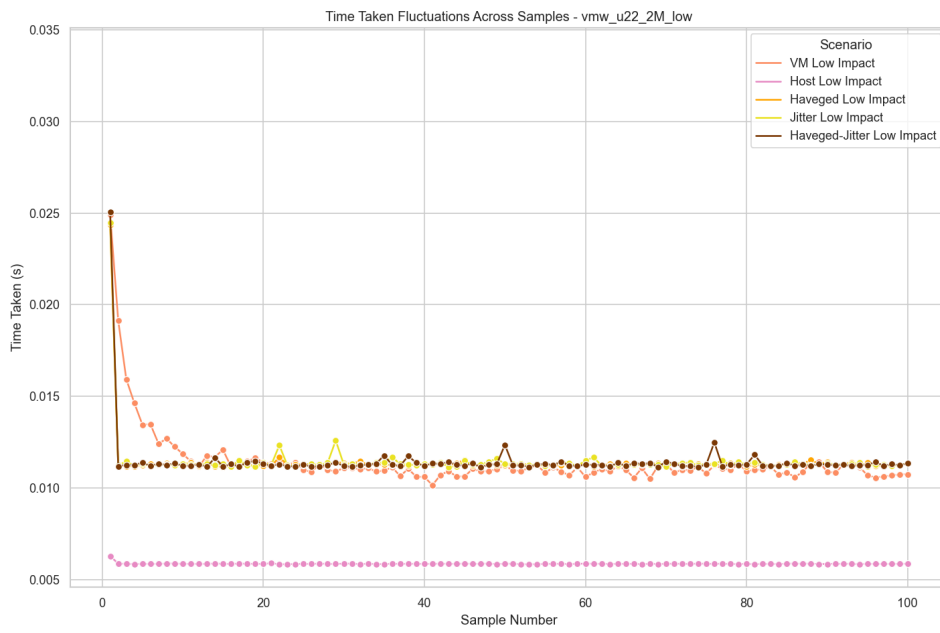


(b) Ubuntu 22.04 on QEMU - Low Impact

Figure 4.20: A comparison of random number generation rate curves of Ubuntu 22.04 on QEMU across bare metal, with and without entropy enhancing tools



(a) Ubuntu 22.04 on VMware - High Impact



(b) Ubuntu 22.04 on VMware - Low Impact

Figure 4.21: A comparison of random number generation rate curves of Ubuntu 22.04 on VMWare across bare metal, with and without entropy enhancing tools

Location	Impact	Drop After Peak
odsk-qemu-al9	high	62.19
	low	57.85
odsk-vbox-al9	high	57.87
	low	56.58
odsk-vmw-al9	high	60.69
	low	59.90

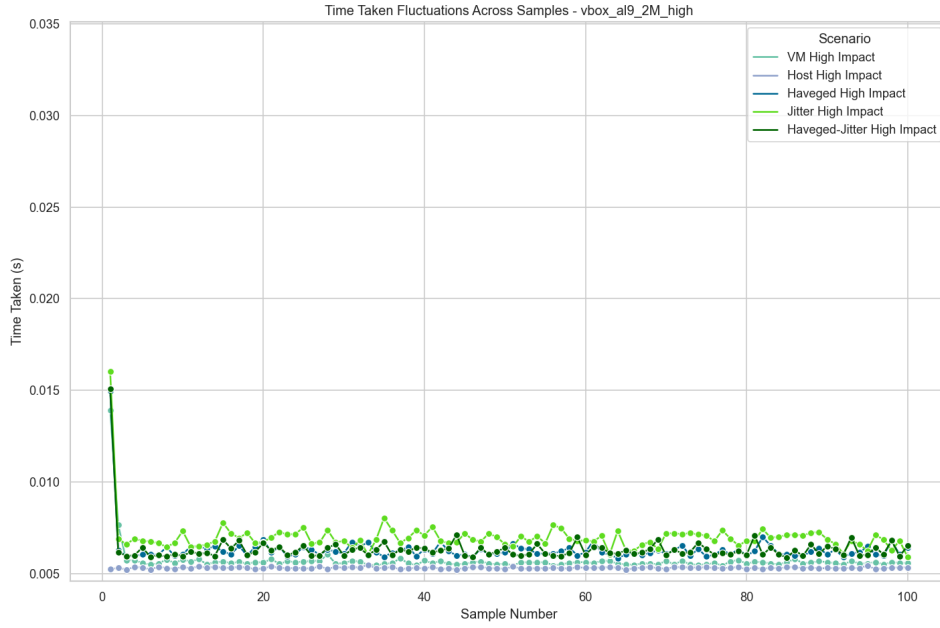
Table 4.25: Entropy Enhancement on AlmaLinux 9.4 VMs only using Haveged

Location	Impact	Drop After Peak
odsk-qemu-al9	high	60.73
	low	56.23
odsk-vbox-al9	high	56.05
	low	52.63
odsk-vmw-al9	high	60.77
	low	59.82

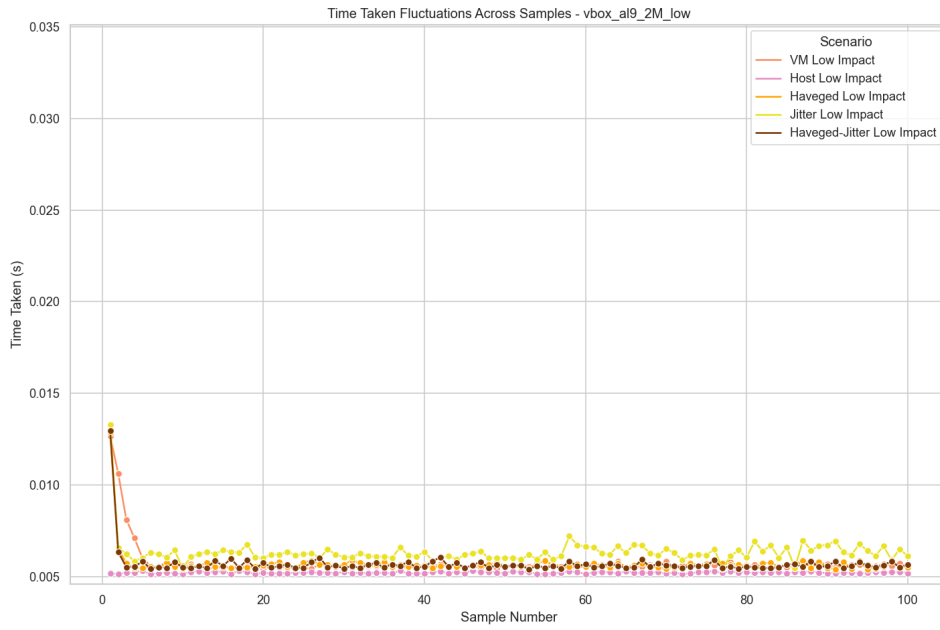
Table 4.26: Entropy Enhancement on AlmaLinux 9.4 VMs only using Jitterentropy

Location	Impact	Drop After Peak
odsk-qemu-al9	high	65.71
	low	56.32
odsk-vbox-al9	high	58.01
	low	56.43
odsk-vmw-al9	high	60.69
	low	59.47

Table 4.27: Combined Entropy Enhancement on AlmaLinux 9.4 using Haveged and Jitterentropy



(a) AlmaLinux 9.4 on Virtual Box - High Impact



(b) AlmaLinux 9.4 on Virtual Box - Low Impact

Figure 4.22: A comparison of random number generation rate curves of AlmaLinux 9.4 on Virtual Box across bare metal, with and without entropy enhancing tools

4.4 Threshold Evaluation

4.4.1 Need for Threshold Merging

Threshold merging is necessary due to the existence of two sets of thresholds derived from the RQ1 (Section 4.1.3.1) and RQ2 (Section 4.2.3.1) experiments, corresponding to single VM and multiple VM setups respectively. During the evaluation of RQ3 (Section 4.3), thresholds from the single VM setup were utilized, given that the RQ3 experiments were explicitly conducted in a single VM environment.

However, in real world malware testing, a program cannot inherently determine whether it is operating within a single VM or a multiple VM environment. Therefore, a unified set of thresholds must be defined to ensure coverage of both scenarios. The following approach was used for combining the thresholds,

- For parameters where a **greater-than** condition flags a VM, the smaller threshold value from the two setups is selected to ensure that both single VM and multiple VM instances are detected. For example, regarding the “drop after peak” measure in Ubuntu 24.04,

In single VM setups, a value greater than 24.85% indicates a VM

In multiple VM setups, a value greater than 13.70% indicates a VM

Therefore, 13.70% is selected as the combined threshold to ensure detection in both cases.

- For parameters where a **less-than** condition flags a VM, the larger threshold value is selected, again to ensure coverage of both scenarios. For example for the SSE ratio in Ubuntu 24.04,

In single VM setups, a value below 7.3 indicates a VM

In multiple VM setups, a value below 1.5 indicates a VM

Therefore, 7.3 is selected as the threshold to cover both setups.

It is acknowledged that the derived thresholds are highly subjective to the data collected through the experiments of this study. As mentioned on the Section 3.3.4, the data collection was repeatedly done to improve generalizability and minimize momentary

system effects. Each experiment was repeated over 10 rounds with sufficient intervals between each round to allow system replenishment, which took a considerable amount of time. The thresholds were then calculated based on the averaged data of these rounds.

However in a practical malware detection scenarios, the decision whether it is a VM or not must be made within a matter of few seconds. Therefore, it was decided to streamline the data collection to a single round using the most effective condition identified by the experimental results. Accordingly random data collection from `/dev/random` interface with a 2MB buffer size was selected.

4.4.2 Proof of Concept Program and Threshold Evaluation

Following the identification of most effective data collection configuration, a C++ Proof of Concept (PoC) program was developed to generate 2MB of random data from the `/dev/random` interface, compute the relevant statistical measures, and determine the virtualization status based on the established thresholds.

The PoC was tested across the VM and bare metal instances used in the previous experiments. The 9 VM instances consisted of three VMs (VirtualBox, VMware and QEMU) across three OSs (Ubuntu 24.04, Ubuntu 22.04 and AlmaLinux 9.4), while 9 bare metal rounds were also included for evaluation.

The PoC was tested across the identified threshold parameters and a detailed analysis of misdetections revealed several important insights. The thresholds associated with the mean gap, SSE ratio, variance ratio, and skewness ratio measures were less reliable across different systems and often fell outside the threshold ranges, causing false positive and false negative detections. However, throughout all evaluation instances, the “drop after peak percentage” threshold proved to be the most reliable, failing the threshold range only twice out of 54 detections.

Given the robustness of the “drop after peak percentage” measure and the less reliability of other measures, it was decided to adopt the drop after peak percentage threshold as the primary VM detection rule, with the remaining parameters retained only as auxiliary indicators. This achieved the best balance between recall, precision, and specificity, reaching an overall accuracy of 94.44%. The final evaluation metrics results are presented in the Table 4.28.

Metric	Result
Accuracy	94.44%
Precision	100.00%
Recall	88.89%
Specificity	100.00%
F1 Score	94.12%

Table 4.28: Evaluation results for virtualization detection

4.4.3 Final Detection Rules

The final set of thresholds for VM detection derived from the experiments, analysis and evaluation are as follows.

- **Ubuntu 24.04**

Drop after peak percentage of both high and low impact levels $> 13.70\%$

- **Ubuntu 22.04**

Drop after peak percentage of both high and low impact levels $> 19.39\%$

- **AlmaLinux 9.4**

Drop after peak percentage of both high and low impact levels $> 38.92\%$

4.5 Summary of the Chapter

This chapter presented the results, evaluation, and discussion of the three research questions addressed in the study. Each research question was analyzed and evaluated systematically based on the experimental data collected.

First, the results for Research Question 1 were presented, focusing on the behaviour of the LRNG under single VM setups. Observations were made regarding patterns in random number generation rates and based on these findings, a set of thresholds for VM detection in single VM environments was derived.

Next, Research Question 2 extended the investigation to multiple VM environments, including desktop, private cloud, and public cloud scenarios. The same analysis struc-

ture was followed, and thresholds for VM detection under multiple VM conditions were similarly derived.

Research Question 3 explored whether entropy enhancing tools could obscure virtualization detection by influencing LRNG behaviour. Analysis of data collected with those tools showed that these methods were largely ineffective in concealing virtualization presence.

Research Question 3 involved evaluating the effectiveness of the thresholds obtained from Research Question 1 and Research Question 2 through the development of a PoC program. The PoC performed real-time detection based on the most prominent experimental findings, focusing on fast execution suitable for practical malware detection.

Following the discussion of the three research questions, a detailed threshold evaluation was conducted. This section addressed the challenge of defining unified thresholds that would be applicable without prior knowledge of the system's virtualization setup (single or multiple VM). Different threshold combination strategies such as conjunctive (AND), disjunctive (OR), and hybrid were evaluated through a PoC program and confusion matrix analysis. The drop after peak percentage threshold was found to provide the most balanced and accurate detection outcomes.

Overall, this chapter established the experimental foundation of the study by presenting key findings, deriving practical thresholds, and evaluating detection strategies, thereby strengthening the support for the research objectives.

5 Conclusion

5.1 Conclusion of the Study

This study was initiated to investigate a crucial problem in the context of malware analysis. The challenge was to explore whether malware could detect that it was running inside a VM, a common setup for safe analysis. If malware identifies the virtualized environment, it may conceal its true behaviour, complicating detection efforts. A review of existing virtualization detection methods revealed that most approaches depended on kernel space data or privileged access, making them unsuitable for user space detection, such as by a malware under analysis. This limitation motivated the search for non privileged techniques capable of identifying virtualization from within user space alone.

While searching for parameters commonly available in Linux systems without requiring privileged access, it was observed that random number generation behaviour differs between physical and virtualized environments. In particular, VMs often employ tools such as rng-tools and haveged to enhance entropy during operations requiring high quality randomness. These differences highlighted the LRNG as a potential indicator of virtualization presence. This observation led to the formulation of the research aim, “to investigate methods for detecting virtualization presence using the LRNG and related parameters from user space, and to explore techniques for obscuring such detection.” Based on this aim, the research objectives and questions were defined.

The research was structured around three key questions, the feasibility of detecting virtualization from a single VM, the extension of detection techniques to multiple VM environments, and the exploration of methods to obscure it if there is a detection from user space. Literature suggested that LRNG entropy fluctuations would be directly measurable, however due to changes in newer kernel versions, these fluctuations were no longer observable. Alternative indirect measures were attempted including using CPU and memory utilization patterns which were ineffective due to the mismatch between the granularity of system monitoring tools and the rapid nature of random number generation. As a result, the study focused on the rate and quality of random number generation as primary indicators. To determine operations with significant LRNG impact, preliminary experiments were conducted on Ubuntu 14, an older system where such effects remained visible. Direct comparisons between bare metal and virtual environments were

avoided due to inherent virtualization overheads. Instead, a delta-based approach was adopted, comparing high and low impact operations within each environment. Experimental setups were then systematically designed to address the research questions, with data collected under controlled conditions across various environments to assess detection feasibility and explore potential countermeasures.

The experimental results revealed that measurable differences in random number generation behaviour exist between bare metal and virtual environments. Even though quality of random numbers did not provide any distinguishable patterns across VMs and bare metal systems, time taken to generate random number samples revealed insightful patterns enabling the successful detection of virtualization presence from user space. A key observation was the presence of an early peak in random number generation distributions, where higher initial times gradually stabilized. However, the detection in multiple VM cloud environments proved more challenging, primarily due to limitations in private and public cloud setups that restricted direct control. Attempts to obscure detection using entropy enhancing tools were unsuccessful, highlighting the need for further exploration into possible mitigation techniques. The thresholds for detecting virtualization were found to be specific to each OS and not universally generalizable, given the limitations of the study. These thresholds were evaluated through various combinations and the best combination was found to provide a detection accuracy of 94.44%. These findings contribute valuable insights into user space based virtualization detection, yet further research is necessary to refine these approaches and explore potential countermeasures.

5.2 Critical Reflection on Research Outcomes

The primary research aim of this study was to investigate methods for detecting virtualization presence using the LRNG and related parameters in virtual environments, as well as to explore techniques to obscure such detection from user space programs. This aim was pursued through three key objectives and research questions.

The first objective was to analyze how a program running in the user space of a Linux VM can detect the underlying virtualization platform using the LRNG and related parameters. The corresponding research question was, “How can a program running in the user space of a Linux VM detect the underlying virtualization using the LRNG and related parameters?” This objective was successfully achieved. Methods for detecting

virtualization presence were identified, and the detection parameters derived from this research showed an accuracy of 94.4%. A significant finding was the presence of an early peak in random number generation distributions on virtual environments, where higher initial generation times gradually stabilized over time. This observation, alongside the accuracy of the thresholds, confirms the success of the first objective.

The second objective focused on evaluating whether this detection capability could be extended to environments with multiple guest VMs on a single host. The related research question was, “Can this detection capability be extended to an environment with multiple guest VMs on a single host?” This objective was partially fulfilled. Detection capability was achieved in multiple VM setups within desktop environments but not consistently in cloud environments (both private and public). The challenges faced in cloud environments impacted the full extension of the detection capability to multiple VM setups, limiting the overall detection in these contexts. However, detection capability in desktop environments proves the potential of the approach in multiple VM environments, achieving this objective partially.

The third objective aimed to identify and propose modifications to the LRNG and related parameters to obscure the presence of virtualization from user space programs running within the VM. The associated research question was, “How can the presence of a VM be obscured from user space programs running within the VM?” This objective, however, was not fully achieved. The approach of using existing entropy enhancing tools to increase entropy richness did not succeed in obscuring the virtualization presence, though it opened a pathway for further exploration. Despite not achieving the intended outcome, this attempt led to a significant finding, that is even with entropy enhancing daemons directly contributing to the entropy pool, the detection method remained robust. This observation highlights the resilience of the detection technique, indicating that it can be detected even in environments where specific efforts are made to enhance entropy. This suggests that the detection capability is strong enough to withstand efforts to obscure it, regardless of the level of system footprint or background service activity. This insight represents the most important takeaway from the third research question, as it emphasises the strength of the detection method, even though the primary objective of obscuring the virtualization presence was not achieved.

Based on the assessment of the three research objectives, where the first objective

was fully achieved, the second partially achieved with certain limitations, and the third providing valuable insights despite not being fully realized, it can be concluded that the research largely accomplished its aim. The study successfully confirmed the capability of detecting virtualization presence through the LRNG in virtual environments, with the detection thresholds demonstrating a 94.44% accuracy. While the exploration of techniques to obscure this detection did not result in conclusive results, it opened important avenues for further investigation and refinement. These findings not only enhance the understanding of virtualization detection but also highlight key areas that require deeper exploration.

5.3 Contributions

This research makes several important contributions to the field of virtualization detection, particularly through the novel use of the LRNG in detecting the presence of VMs. While VM detection has been explored in prior literature, this study uniquely applies the LRNG behaviour for detection, marking a significant change from traditional methods. To the best of the author’s knowledge, the specific approach of utilizing the LRNG behaviour as a VM detection technique has not been explored before.

A key contribution of this work is demonstrating the potential of user space programs in interpreting hardware associated information. While most previous attempts to explore entropy behaviour required kernel level access, this study achieves detection purely from user space, without even the need for root user privileges or additional tool installations. By using C, C++ compiled binaries and Bash scripts, the experiments remained entirely within user space, emphasizing the feasibility and practicality of conducting this kind of analysis without requiring any higher privileges or system modifications.

Furthermore, the study contributes to the understanding of the LRNG in its updated form. The LRNG has undergone frequent updates since 2015, including changes to hash functions, entropy extraction methods, and the nature of its interfaces. While prior studies have analyzed earlier versions, there has been little comprehensive research on the updated LRNG, making this study an important step in examining the current state of the generator. Although not primarily focused on LRNG updates, this research provides valuable insights on the latest version of the LRNG and its implications for VM detection.

In addition, this research also explores the timing side channel vulnerabilities present

in the LRNG. Given that entropy fluctuations could not be directly observed with the updated LRNG, the study employed timing analysis as an indirect measure, effectively utilizing side channel analysis techniques to examine entropy behaviour. This contributes to the broader field of side channel analysis, specifically within the context of random number generation.

An important observation in this study is that the OS wrapper around the kernel plays a significant role in the behaviour of the LRNG and the VM detection process. Differences were observed between Debian based and Red Hat based systems, as well as between Ubuntu 24.04 and Ubuntu 22.04 within the Debian family. These findings suggest that the OS's role in interacting with kernel level measures like entropy could influence VM detection, providing a new avenue for further exploration.

The practical applications of this research extend beyond malware detection. The findings also have implications for other domains where VM detection is critical, such as exam proctoring systems. In these systems, students might attempt to install the exam proctoring software within a VM, enabling them to avoid detection by running other applications on the host system. This study's detection capabilities could prove useful in identifying such scenarios, showing the broader usefulness of this approach in domains where detecting VM presence is essential.

5.4 Limitations

This study faced several limitations that influenced its scope and findings. One of the primary limitations was the scope of virtualization coverage. While the research aimed to cover the most commonly used VMMs and major Linux distributions, the results are specific to these systems. Thus, the findings may not be generalizable to all VM platforms or other Linux-based systems not included in the study.

Additionally, the hardware resources available for experimentation were limited. The inability to incorporate a diverse set of physical machines with varying hardware configurations restricted the generalizability of the results. The study was limited to the available resources, which may not fully represent the wide range of possible hardware setups that could affect the detection process.

Cloud platforms also had constraints, free tier cloud services provided limited resources, and due to the inability to upload custom ISOs had to rely on the cloud plat-

forms default images. These images sometimes had kernel mismatches, requiring kernel downgrades to align with the desktop VM setup. Moreover, cloud environments had certain background services that could not be disabled, impacting the consistency of the experimental procedure. However, as suggested by the results in RQ3, these services may not have significantly impacted the detection outcomes.

Kernel version mismatches between Debian and Red Hat based systems created additional challenges. While the latest Debian kernel version was 6.8, the latest Red Hat kernel was 5.14. To accommodate these differences, Ubuntu 22.04 was used to match the Red Hat kernel version. However Debian with kernel 6.8 also had to be included to make sure the study addresses the latest available kernel version as well.

Performance monitoring tools also presented limitations. As the focus was on user space experiments, avoiding additional tool installations, only built-in system tools were used. But these were unable to capture the finer details of the random number generation process given its fast nature. The smallest monitoring intervals available did not provide the necessary resolution to accurately track performance metrics with random number generation.

Moreover, while significant efforts were made to understand the workings of the LRNG, gaps remain in its complete understanding. The official documentation for the LRNG has not been consistently maintained, which made it challenging to fully explore the updated version. This study offers valuable insights into the latest LRNG but cannot claim a comprehensive understanding of its internal mechanisms.

The evaluation of the study was also constrained by the experimental setup. The lack of additional resources for expanding the evaluation meant that the findings could not be tested across different setups or configurations, affecting their robustness. Also, the thresholds derived in this study for detecting VMs were specific to the tested OSs and could not be generalized across diverse systems. Further work is required to create universal thresholds that can be applied more broadly.

Finally, the NIST Test Suite, used to assess the quality of random number generation, has its own limitations. Although it is considered as one of the most reliable tools available, it has been criticized for not capturing all potential weaknesses in random number generation (Kenny and Mosurski 2005).

5.5 Future Works

Building upon the findings and limitations of this study, several directions for future research emerge, each contributing to a deeper understanding of VM detection using the LRNG. The immediate focus should be on directly extending the scope of this study and addressing its limitations.

A major limitation of this study was the restricted hardware resources, which limited the range of VMMs and OSs that could be evaluated. Future research should aim to use a broader range of VMMs, hardware configurations, and OSs to enhance the generalizability of the results.

A significant insight identified in this work is the inability of existing entropy enhancing tools to obscure VM detection, as demonstrated in Research Question 3 (Section 4.3). This raises an important question for future research, whether enhancing entropy alone can effectively hide the presence of virtualization. The development of custom entropy enhancing techniques is one possible direction to explore, but it is important to recognize that other approaches may also be necessary. This area requires extensive study, not limited to entropy enhancement but also exploring alternative methods that might obscure VM presence.

To effectively pursue the above, further research into the underlying mechanisms of the LRNG is essential. This study has provided insights into the observable behaviours of the LRNG, but the kernel level processes and entropy collection methods that drive these behaviours remain partially unexplored. A deeper understanding of these mechanisms will be crucial in developing effective strategies to obscure VM detection.

Furthermore, given the limitations of built-in performance monitoring tools accessible from user space, and the importance of granular system performance metrics in understanding the underlying functionality of the kernel, future studies can be extended to explore the feasibility of incorporating advanced tools without privileged access or developing more precise performance monitoring mechanisms.

This study found notable differences between Debian based and Red Hat based systems, as well as when comparing Ubuntu 22.04 and Ubuntu 24.04. Specifically, AlmaLinux was notably harder to detect as a VM, presenting only one detection criterion. This characteristic requires further exploration within the context of Red Hat based systems. Moreover, expanding the study to include other Linux families could reveal deeper

insights into how the OS wrapper impacts kernel level entropy collection.

Beyond Linux, examining non Linux OSs and alternative architectures such as ARM or RISC-V, would provide a more holistic understanding of behaviour of the RNG across diverse environments.

Additionally, beyond malware testing, there are other domains where virtualization detection may play a crucial role. For example, exam proctoring systems represent an area where identifying whether a system is running in a VM could help ensure the integrity of the testing process. Exploring such use cases could further expand the applications of virtualization detection, highlighting scenarios where the ability to identify or obscure virtualization is essential.

Furthermore, expanding the scope of virtualization detection research to include other isolation techniques, such as sandboxes and emulation, could provide valuable insights. Understanding how the LRNG behaves under these techniques could help in identifying or obscuring isolation in various contexts, including malware analysis and other security domains.

References

- Adams, Keith and Ole Agesen (Oct. 2006). “A comparison of software and hardware techniques for x86 virtualization”. In: *ACM SIGOPS Operating Systems Review* 40 (5), pp. 2–13. ISSN: 0163-5980. DOI: 10.1145/1168917.1168860.
- Allen, Bo (2016). *Random Number Visualization*. [Accessed: 2024-12-26]. URL: <https://boallen.com/random-numbers.html>.
- Barham, Paul et al. (Oct. 2003). “Xen and the art of virtualization”. In: *SIGOPS Oper. Syst. Rev.* 37.5, pp. 164–177. ISSN: 0163-5980. DOI: 10.1145/1165389.945462. URL: <https://doi.org/10.1145/1165389.945462>.
- Bassham, L E et al. (2010). *A statistical test suite for random and pseudorandom number generators for cryptographic applications*. DOI: 10.6028/NIST.SP.800-22r1a.
- Bulazel, Alexei and Bülent Yener (Nov. 2017). “A Survey On Automated Dynamic Malware Analysis Evasion and Counter-Evasion”. In: *Proceedings of the 1st Reversing and Offensive-oriented Trends Symposium*. ACM, pp. 1–21. ISBN: 9781450353212. DOI: 10.1145/3150376.3150378.
- Chen, Ping et al. (2016). “Advanced or Not? A Comparative Study of the Use of Anti-debugging and Anti-VM Techniques in Generic and Targeted Malware”. In: Springer, pp. 323–336. DOI: 10.1007/978-3-319-33630-5_22.
- Cieslarová, Radka (2018). “Analysis of the Linux random number generator in virtualized environment”. In.
- Cover, Thomas M. and Joy A. Thomas (1991). “Entropy, Relative Entropy and Mutual Information”. In: Wiley.
- Dillon, Nick (2017). *LavaRand in Production: The Nitty-Gritty Technical Details*. [Accessed: 2024-12-27]. URL: <https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/>.
- Donenfeld, Jason (2022). *Inside the Linux Kernel RNG*. Presentation at the Linux Plumbers Conference 2022. <https://www.youtube.com/live/KkOdMwZRpYY?si=47x73zfA69KmKije&t=10093>.
- drivers/char/random.c* (2024). <https://github.com/torvalds/linux/blob/master/drivers/char/random.c>. [Accessed: 2024-11-21]. The Linux Kernel Community.
- Everspaugh, Adam et al. (Nov. 2014). “Not-So-Random numbers in virtualized linux and the Whirlwind RNG”. In: *Proceedings - IEEE Symposium on Security and Privacy*. In-

- stitute of Electrical and Electronics Engineers Inc., pp. 559–574. ISBN: 9781479946860. DOI: 10.1109/SP.2014.42.
- Goldberg, Robert P. (1974). “Survey of virtual machine research”. In: *Computer* 7.6, pp. 34–45. DOI: 10.1109/MC.1974.6323581.
- GPG does not have enough entropy*, serverfault.com (2010). <https://serverfault.com/questions/214605/gpg-does-not-have-enough-entropy>. [Accessed 28-11-2024].
- Intel (2018). *Intel® Digital Random Number Generator (DRNG) Software Implementation...* URL: <https://www.intel.com/content/www/us/en/developer/articles/guide/intel-digital-random-number-generator-drng-software-implementation-guide.html> (visited on 02/24/2025).
- (2023). *Intel® 64 and IA-32 Architectures Software Developer’s Manual, Volume 1: Basic Architecture*. Chapter 6: Procedure Calls, Interrupts, and Exceptions. Intel Corporation. URL: <https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html>.
- Jithin, R. and Priya Chandran (2014). “Virtual Machine Isolation”. In: *Recent Trends in Computer Networks and Distributed Systems Security*. Ed. by Gregorio Martinez Perez et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 91–102. ISBN: 978-3-642-54525-2. DOI: 10.1007/978-3-642-54525-2_8.
- Kelsey, John et al. (1998). “Cryptanalytic Attacks on Pseudorandom Number Generators”. In: *Fast Software Encryption, 5th International Workshop, FSE ’98, Paris, France, March 23-25, 1998, Proceedings*. Vol. 1372. Lecture Notes in Computer Science. Springer, pp. 168–188. DOI: 10.1007/3-540-69710-1_12.
- Kenny, Charmaine and Krzysztof Mosurski (2005). *TRINITY COLLEGE DUBLIN Management Science and Information Systems Studies Project Report Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators*. Tech. rep. TRINITY COLLEGE DUBLIN. URL: <http://www.random.org>.
- Kumari, Rashmi, Mohsen Alimomeni, and Reihaneh Safavi-Naini (Oct. 2015). “Performance analysis of Linux RNG in virtualized environments”. In: *CCSW 2015 - Proceedings of the 7th ACM Cloud Computing Security Workshop, co-located with: CCS 2015*. Association for Computing Machinery, Inc, pp. 29–40. ISBN: 9781450338257. DOI: 10.1145/2808425.2808434.

- Lusky, Yehonatan and Avi Mendelson (Apr. 2021). “Sandbox detection using hardware side channels”. In: *Proceedings - International Symposium on Quality Electronic Design, ISQED*. Vol. 2021-April. IEEE Computer Society, pp. 192–197. ISBN: 9781728176413. DOI: 10.1109/ISQED51717.2021.9424260.
- Müller, Stephan (2022). *Jitter RNG*. en-us. URL: <https://chronox.de/jent/index.html> (visited on 02/25/2025).
- Pék, Gábor, Levente Buttyán, and Boldizsár Bencsáth (June 2013). “A survey of security issues in hardware virtualization”. In: *ACM Computing Surveys* 45 (3), pp. 1–34. ISSN: 0360-0300. DOI: 10.1145/2480741.2480757.
- Popek, Gerald J. and Robert P. Goldberg (July 1974). “Formal requirements for virtualizable third generation architectures”. In: *Communications of the ACM* 17 (7), pp. 412–421. ISSN: 0001-0782. DOI: 10.1145/361011.361073.
- QEMU (2016). *Features/VirtIORNG - QEMU*. URL: <https://wiki.qemu.org/Features/VirtIORNG> (visited on 02/24/2025).
- Raffetseder, Thomas, Christopher Kruegel, and Engin Kirda (2007). “Detecting System Emulators”. In: Springer Berlin Heidelberg, pp. 1–18. DOI: 10.1007/978-3-540-75496-1_1.
- Random.org (2025). *Random.org – True Random Number Service*. [Accessed: 2024-12-14]. URL: <https://www.random.org/>.
- RedHat (2018). 9.6. *Random Number Generator (RNG) Device — Virtualization Administration Guide — Red Hat Enterprise Linux — 6 — Red Hat Documentation*. URL: https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/6/html/virtualization_administration_guide/sect-guest_virtual_machine_device_configuration-random_number_generator_device (visited on 02/24/2025).
- Shannon, C. E. (1949). “Communication Theory of Secrecy Systems”. In: *Bell System Technical Journal* 28.4, pp. 656–715. DOI: <https://doi.org/10.1002/j.1538-7305.1949.tb00928.x>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/j.1538-7305.1949.tb00928.x>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/j.1538-7305.1949.tb00928.x>.

- Smith, James E. and Ravi Nair (2005). *Virtual Machines Versatile Platforms for Systems and Processes*. Morgan Kaufmann Publishers. ISBN: 978-1-55860-910-5. DOI: <https://doi.org/10.1016/B978-1-55860-910-5.X5000-9>.
- Sun, Ming-Kung et al. (Dec. 2011). “Malware Virtualization-Resistant Behavior Detection”. In: *2011 IEEE 17th International Conference on Parallel and Distributed Systems*. IEEE, pp. 912–917. ISBN: 978-0-7695-4576-9. DOI: 10.1109/ICPADS.2011.78.
- Tanenbaum, Andrew S. and Herbert Bos (2014). *Modern Operating Systems*. 4th. USA: Prentice Hall Press. ISBN: 013359162X.
- Torvalds, Linus (Jan. 7, 2024). *random: introduce getrandom(2) system call*. <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=c6e9d6f38894798696f23c8084ca7edbf16ee895>. [Accessed: 2024-11-21].
- Wuertz, Gary and Jirka Hladky (2025). *haveged - a simple entropy daemon*. URL: <https://www.issihosts.com/haveged/> (visited on 06/24/2025).
- Yokoyama, Akira et al. (2016). “SandPrint: Fingerprinting Malware Sandboxes to Provide Intelligence for Sandbox Evasion”. In: *International Symposium on Recent Advances in Intrusion Detection*. URL: <https://api.semanticscholar.org/CorpusID:13578393>.
- Zhang, Zhi et al. (2021). “Detecting Hardware-Assisted Virtualization With Inconspicuous Features”. In: *IEEE Transactions on Information Forensics and Security* 16, pp. 16–27. ISSN: 1556-6013. DOI: 10.1109/TIFS.2020.3004264.

A Appendix : Detailed Statistics for RQ 1 - Rate of Generation

Table A.1: RQ1 Ubuntu 24.04 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/random	2M	dsk-host-u24	high	0.0059	-0.0009	423.8643	385.3518	1.0216	3.2055	1.0009	2.1089	1.3924	-0.1463
			low	0.0068				1.0330	0.2867	0.9597	2.0197	-0.2038	
		dsk-qemu-u24	high	0.0067	0.0006	0.3065	0.9139	1.3432	3.2849	1.0336	26.1683	0.6451	10.3115
			low	0.0061				1.9127	8.2627	1.0942	48.0297	6.6516	
		dsk-vbox-u24	high	0.0063	0.0003	0.3195	1.5502	1.6373	4.8145	1.0501	39.6654	2.3755	2.1595
			low	0.0060				2.1071	6.4074	1.2497	54.3856	5.1298	
		dsk-vmw-u24	high	0.0127	0.0003	0.6751	1.4062	1.7338	6.8684	1.0535	43.2342	3.2983	1.3188
			low	0.0124				1.8979	6.9133	1.1755	48.3606	4.3499	
/dev/random	4M	dsk-host-u24	high	0.0118	-0.0021	549.7047	444.7477	1.0216	4.3136	1.0018	2.1009	1.8148	-0.4879
			low	0.0139				1.0268	0.2992	0.9829	3.2969	-0.8854	
		dsk-qemu-u24	high	0.0127	0.0005	1.5356	0.8601	2.0074	8.5869	1.0805	50.7242	6.6136	0.6890
			low	0.0122				1.7250	6.4217	1.1174	42.5670	4.5565	
		dsk-vbox-u24	high	0.0122	0.0001	1.0617	1.6154	2.0378	8.1939	1.0779	51.1877	6.2656	0.7487
			low	0.0121				2.0034	6.1953	1.2235	51.4453	4.6908	
		dsk-vmw-u24	high	0.0243	-0.0008	3.2673	2.1362	1.7294	8.4681	1.0591	43.0305	6.1943	0.6407
			low	0.0251				1.8493	6.9554	1.1124	46.0776	3.9690	
/dev/random	6M	dsk-host-u24	high	0.0176	-0.0028	470.7257	407.0271	1.0181	3.8459	1.0018	1.7112	2.1761	-0.4296
			low	0.0204				1.0326	0.3985	0.9772	2.5181	-0.9348	
		dsk-qemu-u24	high	0.0185	0.0000	1.7646	1.4854	1.8646	8.5745	1.0707	47.1875	6.7491	0.7265
			low	0.0185				1.9019	7.3245	1.1252	47.6237	4.9029	
		dsk-vbox-u24	high	0.0181	0.0000	2.0095	1.5655	2.0005	8.7199	1.0838	50.3079	7.3217	0.6660
			low	0.0181				2.0274	7.1656	1.1590	51.5634	4.8762	
		dsk-vmw-u24	high	0.0358	-0.0018	6.9454	2.8123	1.5907	8.6799	1.0430	37.3939	6.7074	0.4040
			low	0.0376				1.6612	6.0883	1.0772	41.2420	2.7095	

Table A.2: RQ1 Ubuntu 24.04 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/urandom	2M	dsk-host-u24	high	0.0059	-0.0009	337.5484	322.8851	1.0125	1.7105	0.9999	1.2618	1.6938	-0.1279
			low	0.0068				1.0392	0.3439	0.9834	5.5292	-0.2166	
		dsk-qemu-u24	high	0.0067	0.0006	0.2990	0.3836	1.6128	5.2713	1.0481	38.0767	1.8734	2.2928
			low	0.0061				1.4522	5.6770	1.0693	31.2872	4.2955	
		dsk-vbox-u24	high	0.0063	0.0004	0.5442	0.7406	1.6413	4.7744	1.0387	39.3387	2.4297	1.7553
			low	0.0059				1.5624	4.5774	1.1360	37.5221	4.2647	
		dsk-vmw-u24	high	0.0127	0.0002	1.1943	1.2799	1.8087	7.2513	1.0638	45.3163	3.8388	1.0160
			low	0.0126				1.7039	5.5002	1.1695	41.6879	3.9002	
	4M	dsk-host-u24	high	0.0118	-0.0021	434.8598	404.5438	1.0182	3.3422	1.0020	1.7409	2.4018	-0.3177
			low	0.0138				1.0401	0.4291	0.9692	2.9846	-0.7630	
		dsk-qemu-u24	high	0.0126	0.0003	1.1994	2.0431	1.6953	7.1926	1.0519	41.0471	4.2142	1.3031
			low	0.0124				2.0290	7.2993	1.1602	51.2796	5.4916	
		dsk-vbox-u24	high	0.0122	0.0002	0.8970	1.6563	2.0964	8.6265	1.0878	52.8768	6.7161	0.8299
			low	0.0121				2.2234	7.3710	1.2265	56.6740	5.5739	
		dsk-vmw-u24	high	0.0243	-0.0009	3.1934	1.8214	1.7036	8.3587	1.0572	41.3374	5.9797	0.5742
			low	0.0252				1.7279	6.6362	1.1043	42.8377	3.4338	
/dev/urandom	6M	dsk-host-u24	high	0.0176	-0.0028	422.4028	387.5695	1.0145	3.0820	1.0020	1.4406	2.1593	-0.3925
			low	0.0204				1.0113	0.1419	0.9667	2.6237	-0.8476	
		dsk-qemu-u24	high	0.0185	0.0001	1.4619	1.8381	1.6604	7.3651	1.0556	40.0377	4.9167	0.9874
			low	0.0184				1.8740	7.1610	1.1231	47.3916	4.8547	
		dsk-vbox-u24	high	0.0181	-0.0001	4.6124	1.5173	2.1309	9.4814	1.0886	53.7607	8.7725	0.5409
			low	0.0182				2.0161	6.9427	1.1813	51.0872	4.7450	
		dsk-vmw-u24	high	0.0358	-0.0018	6.2829	2.4493	1.5714	8.6780	1.0427	36.7347	6.6657	0.3709
			low	0.0376				1.5985	6.1030	1.0575	37.8140	2.4721	

Table A.3: RQ1 Ubuntu 22.04 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/random	2M	dsk-host-u22	high	0.0061	0.0002	1.1981	1.3492	1.0062	0.4845	0.9984	0.6230	0.6593	3.5241
			low	0.0059				1.0713				2.3235	
		dsk-qemu-u22	high	0.0065	-0.0004	1.0826	0.6774	2.4704	6.7781	1.1127	60.8538	4.3103	0.5941
			low	0.0070				1.7609				2.5607	
		dsk-vbox-u22	high	0.0064	-0.0016	8.2099	1.7316	2.8327	8.8633	1.1514	65.0486	7.3972	0.2185
			low	0.0079				1.8316				1.6166	
		dsk-vmw-u22	high	0.0108	-0.0006	1.6540	1.2510	2.4469	8.3358	1.1226	59.4027	6.4568	0.5898
			low	0.0114				2.1800				3.8085	
/dev/random	4M	dsk-host-u22	high	0.0121	0.0004	8.9752	9.2754	1.0026	0.2558	0.9995	0.2707	1.1321	3.2841
			low	0.0117				1.0639				3.7179	
		dsk-qemu-u22	high	0.0130	-0.0012	1.5534	0.6710	2.5731	7.2306	1.1456	61.7050	4.5032	0.4587
			low	0.0141				1.7492				2.0656	
		dsk-vbox-u22	high	0.0127	-0.0019	6.2620	1.6763	2.7179	8.8221	1.1567	64.1066	7.3436	0.2753
			low	0.0146				1.9982				2.0217	
		dsk-vmw-u22	high	0.0219	-0.0008	1.2633	1.2532	2.0565	7.3332	1.0817	51.8216	4.8114	0.7987
			low	0.0226				2.0443				3.8429	
/dev/random	6M	dsk-host-u22	high	0.0181	0.0008	0.3555	0.4443	1.0062	0.6224	0.9992	0.5444	1.1470	2.3844
			low	0.0173				1.0298				2.7348	
		dsk-qemu-u22	high	0.0194	-0.0016	1.1369	0.8474	2.1145	5.7970	1.0684	53.6726	3.2112	0.6176
			low	0.0210				1.7206				1.9833	
		dsk-vbox-u22	high	0.0190	-0.0028	5.7798	1.5934	2.6263	8.6729	1.1216	62.5530	6.9596	0.2218
			low	0.0218				1.9100				1.5437	
		dsk-vmw-u22	high	0.0325	-0.0017	1.3720	1.2174	1.8515	6.5994	1.0728	46.1653	3.8257	0.7152
			low	0.0342				1.7874				2.7363	

Table A.4: RQ1 Ubuntu 22.04 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/urandom	2M	dsk-host-u22	high	0.0061	0.0002	1.1442	1.3019	1.0250	1.5099	1.0034	2.3874	1.7975	1.7530
			low	0.0059				1.0840	4.2786	1.0068	7.7511	3.1509	
		dsk-qemu-u22	high	0.0066	-0.0004	0.7092	0.4888	2.2972	6.3296	1.1297	58.1940	3.7105	0.8757
			low	0.0070				1.7191	5.3178	1.1178	41.2866	3.2494	
		dsk-vbox-u22	high	0.0064	-0.0012	5.7121	1.7041	2.8711	8.7769	1.1711	66.3666	7.1952	0.2206
			low	0.0076				1.8709	3.7312	1.3469	42.6353	1.5876	
		dsk-vmw-u22	high	0.0109	-0.0003	1.1910	1.2289	2.2509	7.8296	1.0830	55.6011	5.4883	0.7903
			low	0.0112				2.1501	6.6705	1.2189	55.2002	4.3375	
/dev/urandom	4M	dsk-host-u22	high	0.0121	0.0005	5.5970	6.1873	1.0057	0.5737	1.0000	0.5347	0.9720	4.5539
			low	0.0116				1.0815	3.1658	1.0186	7.3289	4.4265	
		dsk-qemu-u22	high	0.0130	-0.0012	0.7214	1.0892	2.1072	5.6765	1.0969	52.0736	3.2958	0.7440
			low	0.0142				1.9756	5.2370	1.3195	47.6972	2.4520	
		dsk-vbox-u22	high	0.0127	-0.0025	7.5556	1.4578	2.8323	8.9672	1.1679	65.2989	7.5550	0.2721
			low	0.0152				1.9019	4.3740	1.3155	45.3779	2.0557	
		dsk-vmw-u22	high	0.0218	-0.0011	1.8954	1.3969	2.0872	7.7706	1.0840	52.9312	5.3447	0.6561
			low	0.0229				2.0038	6.3883	1.1840	51.2148	3.5068	
/dev/urandom	6M	dsk-host-u22	high	0.0180	0.0006	9.5955	7.9194	1.0031	0.3274	0.9999	0.3889	1.3227	2.4246
			low	0.0174				1.0354	1.2614	1.0095	2.8165	3.2068	
		dsk-qemu-u22	high	0.0191	-0.0015	1.0433	0.9627	1.8031	4.7685	1.0906	44.3247	2.8538	0.6997
			low	0.0207				1.7362	4.8156	1.1477	41.5497	1.9969	
		dsk-vbox-u22	high	0.0188	-0.0035	6.5446	2.0962	2.4924	8.7351	1.1180	60.5773	7.2756	0.1459
			low	0.0223				1.8611	4.1308	1.2729	41.3465	1.0613	
		dsk-vmw-u22	high	0.0325	-0.0025	1.5395	1.1775	1.9027	6.7447	1.0814	47.8653	4.0450	0.5951
			low	0.0349				1.7817	5.7889	1.1348	44.6405	2.4071	

Table A.5: RQ1 AlmaLinux 9.4 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/random	2M	dsk-host-al9	high	0.0053	0.0001	0.8607	0.8760	1.0162	0.5726	0.9987	1.4422	1.4143	1.0095
			low	0.0052				1.0194	0.7178	0.9975	1.9163	1.4277	
		dsk-qemu-al9	high	0.0054	0.0001	0.5382	0.7056	2.6763	9.0363	1.1373	63.4390	8.2517	0.8821
			low	0.0054				2.3295	8.4471	1.1616	57.9297	7.2788	
		dsk-vbox-al9	high	0.0057	0.0000	0.7708	1.0621	2.4369	8.6403	1.1420	59.6043	7.4135	0.7644
			low	0.0058				2.2021	7.0685	1.2138	56.5406	5.6672	
		dsk-vmw-al9	high	0.0054	0.0001	0.2632	0.5452	2.9069	9.2555	1.1514	66.2252	8.4308	0.7872
			low	0.0053				2.2596	8.1878	1.1793	57.0398	6.6363	
			high	0.0105	0.0002	0.7838	0.7594	1.0131	0.8180	1.0005	1.3983	1.5130	0.8454
			low	0.0103				1.0017	0.1164	0.9974	-0.0778	1.2791	
	4M	dsk-qemu-al9	high	0.0107	0.0001	0.4495	0.6212	2.6554	9.1863	1.1342	62.4668	8.3682	0.9902
			low	0.0106				2.3063	9.0941	1.1340	57.2822	8.2864	
		dsk-vbox-al9	high	0.0113	-0.0002	3.0664	1.0467	2.5196	9.3661	1.1338	61.0390	8.7348	0.6087
			low	0.0115				2.1581	7.0744	1.1710	55.5529	5.3172	
		dsk-vmw-al9	high	0.0106	0.0001	0.2352	0.5172	2.9061	9.3831	1.1447	65.7980	9.1263	0.8037
			low	0.0105				2.2808	8.6857	1.1543	57.2750	7.3345	
			high	0.0157	0.0004	1.0209	0.9558	1.0056	0.5823	1.0004	0.4856	1.3299	0.8547
			low	0.0153				1.0049	0.5085	0.9989	0.3203	1.1366	
/dev/random	6M	dsk-qemu-al9	high	0.0159	0.0002	0.6372	0.6846	2.2865	9.1028	1.1031	57.1515	8.2514	1.0410
			low	0.0157				2.0819	9.1594	1.0889	52.6063	8.5896	
		dsk-vbox-al9	high	0.0169	-0.0002	1.0276	1.1251	2.3229	8.9735	1.1029	57.6522	8.5246	0.6799
			low	0.0171				2.2045	7.7727	1.1504	55.5851	5.7956	
		dsk-vmw-al9	high	0.0158	0.0002	0.1665	0.6494	2.5067	8.8825	1.1049	60.5417	7.9933	0.9615
			low	0.0157				2.2342	8.9234	1.1339	56.3494	7.6856	
			high	0.0157	0.0002	0.1665	0.6494	2.5067	8.8825	1.1049	60.5417	7.9933	0.9615
			low	0.0157				2.2342	8.9234	1.1339	56.3494	7.6856	

Table A.6: RQ1 AlmaLinux 9.4 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Ratio		
/dev/urandom	2M	dsk-host-a19	high	0.0053	0.0001	0.8053	0.8032	1.0121	0.4127	0.9993	1.3390	1.3370	1.0751		
			low	0.0052			1.0143	0.5305	0.9986	1.1774	1.4374				
		dsk-qemu-a19	high	0.0055	0.0001	0.5356	0.7014	2.5110	8.7860	1.1158	60.7508	7.5753	1.0527		
			low	0.0054			2.3161	8.9703	1.1209	57.6043	7.9748				
		dsk-vbox-a19	high	0.0057	0.0000	5.6952	0.7657	2.4292	9.2619	1.1307	59.6025	8.2762	0.4663		
			low	0.0057			1.6568	4.8385	1.1255	40.9699	3.8593				
		dsk-vmw-a19	high	0.0054	0.0001	0.7741	0.4497	3.0644	9.4402	1.1677	68.0933	8.8530	0.7243		
			low	0.0053			2.1787	7.9268	1.1630	55.4734	6.4120				
		/dev/urandom	4M	dsk-host-a19	high	0.0105	0.0002	0.9312	0.8988	1.0050	0.3457	0.9991	0.4790	1.3940	0.9657
					low	0.0103			1.0102	0.7225	1.0009	1.0719	1.3461		
				dsk-qemu-a19	high	0.0107	0.0001	0.4708	0.6083	2.5822	9.1526	1.1267	61.5644	8.2375	1.0179
low	0.0106						2.2430	9.0985	1.1255	56.1601	8.3849				
dsk-vbox-a19	high			0.0113	-0.0002	1.0616	1.1777	2.4016	9.0153	1.1215	59.1368	8.2187	0.7023		
	low			0.0114			2.2441	7.4796	1.1917	56.7104	5.7718				
dsk-vmw-a19	high			0.0107	0.0001	0.1620	0.5035	2.8614	9.0859	1.1491	65.9404	8.1296	0.8959		
	low			0.0105			2.2771	8.6634	1.1582	57.1472	7.2837				
/dev/urandom	6M			dsk-host-a19	high	0.0157	0.0004	0.8020	0.7665	1.0244	2.0267	1.0014	2.4248	1.9668	0.5333
					low	0.0153			1.0074	0.6798	1.0015	0.8468	1.0490		
				dsk-qemu-a19	high	0.0159	0.0001	0.5144	0.6342	2.3666	8.8710	1.1096	58.6775	7.9283	0.9781
		low	0.0158				2.0627	8.5848	1.1136	52.0986	7.7550				
		dsk-vbox-a19	high	0.0169	-0.0002	2.5405	1.1124	2.3774	9.4704	1.1107	58.3887	8.9081	0.6455		
				0.0171			2.1849	7.8007	1.1551	55.0689	5.7498				
		dsk-vmw-a19	high	0.0159	0.0003	0.1692	0.4877	2.7674	9.1796	1.1308	64.6679	8.2561	0.9381		
			low	0.0156			2.2264	8.9678	1.1355	56.1469	7.7451				

B Appendix : Detailed Statistics for RQ 1 - Quality of Generation

Table B.1: RQ1 - /dev/random - Percentage of samples that passed all NIST quality tests

Interface	OS	Size	Impact	Host	Virtual Box	VMWare	QEMU
/dev/random	Ubuntu 24	2M	high	90	84	85	90
			low	84	82	83	89
		4M	high	79	90	86	85
			low	80	81	88	83
		6M	high	88	80	85	84
			low	82	86	87	90
/dev/random	Ubuntu 22	2M	high	86	84	83	86
			low	84	88	86	88
		4M	high	84	85	83	84
			low	85	81	85	86
		6M	high	82	78	80	81
			low	85	82	87	88
/dev/random	Alma Linux 9	2M	high	89	80	81	79
			low	83	81	92	85
		4M	high	79	84	83	89
			low	82	83	87	79
		6M	high	74	81	80	86
			low	75	88	77	85

Table B.2: RQ1 - /dev/urandom - Percentage of samples that passed all NIST quality tests

Interface	OS	Size	Impact	Host	Virtual Box	VMWare	QEMU
/dev/urandom	Ubuntu 24	2M	high	86	85	83	82
			low	85	87	86	82
		4M	high	81	89	80	83
			low	87	81	84	82
		6M	high	90	83	86	80
			low	85	83	81	82
/dev/urandom	Ubuntu 22	2M	high	81	84	85	81
			low	77	82	83	85
		4M	high	83	86	88	85
			low	90	81	80	85
		6M	high	85	83	85	80
			low	89	82	84	80
/dev/urandom	Alma Linux 9	2M	high	76	79	83	80
			low	87	77	90	84
		4M	high	89	88	85	87
			low	84	81	83	83
		6M	high	84	88	87	82
			low	84	84	82	83

Table B.3: RQ1 - /dev/random - Percentage of samples that passed all NIST quality tests, split by peak presence

Interface	OS	Size	Impact	Host		Virtual Box		VMWare		QEMU	
				Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak
/dev/random	Ubuntu 24	2M	high	83.33	90.91	91.67	82.95	91.67	84.09	83.33	90.91
			low	83.33	84.09	83.33	81.82	100.00	80.68	91.67	88.64
		4M	high	58.33	81.82	83.33	90.91	75.00	87.50	83.33	85.23
			low	75.00	80.68	91.67	79.55	83.33	88.64	91.67	81.82
		6M	high	83.33	88.64	66.67	81.82	83.33	85.23	83.33	84.09
			low	75.00	82.95	83.33	86.36	83.33	87.50	83.33	90.91
/dev/random	Ubuntu 22	2M	high	83.33	86.36	83.33	84.09	75.00	84.09	83.33	86.36
			low	91.67	82.95	83.33	88.64	83.33	86.36	66.67	90.91
		4M	high	100.00	81.82	83.33	85.23	83.33	82.95	83.33	84.09
			low	83.33	85.23	83.33	80.68	100.00	82.95	83.33	86.36
		6M	high	83.33	81.82	66.67	79.55	83.33	79.55	66.67	83.91
			low	100.00	82.95	75.00	82.95	83.33	87.50	66.67	90.91
/dev/random	Alma Linux 9	2M	high	75.00	90.91	75.00	80.68	66.67	82.95	83.33	78.41
			low	91.67	81.82	75.00	81.82	91.67	92.05	83.33	85.23
		4M	high	91.67	77.27	83.33	84.09	83.33	82.95	100.00	87.50
			low	75.00	82.95	83.33	82.95	91.67	86.36	83.33	78.41
		6M	high	58.33	76.14	83.33	80.68	91.67	78.41	91.67	85.23
			low	75.00	75.00	91.67	87.50	75.00	77.27	75.00	86.36

Table B.4: RQ1 - /dev/urandom - Percentage of samples that passed all NIST quality tests, split by peak presence

Interface	OS	Size	Impact	Host		Virtual Box		VMWare		QEMU	
				Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak
/dev/urandom	Ubuntu 24	2M	high	75.00	87.50	83.33	85.23	83.33	82.95	75.00	82.95
			low	66.67	87.50	91.67	86.36	75.00	87.50	66.67	84.09
		4M	high	91.67	79.55	75.00	90.91	91.67	78.41	83.33	82.95
			low	91.67	86.36	58.33	84.09	100.00	81.82	83.33	81.82
		6M	high	83.33	90.91	91.67	81.82	66.67	88.64	83.33	79.55
			low	83.33	85.23	83.33	82.95	75.00	81.82	75.00	82.95
/dev/urandom	Ubuntu 22	2M	high	66.67	82.95	83.33	84.09	83.33	85.23	91.67	79.55
			low	75.00	77.27	83.33	81.82	75.00	84.09	91.67	84.09
		4M	high	91.67	81.82	83.33	86.36	83.33	88.64	91.67	84.09
			low	91.67	89.77	83.33	80.68	83.33	79.55	91.67	84.09
		6M	high	100.00	82.95	100.00	80.68	83.33	85.23	83.33	79.55
			low	91.67	88.64	75.00	82.95	66.67	86.36	91.67	78.41
/dev/urandom	Alma Linux 9	2M	high	75.00	76.14	83.33	78.41	83.33	82.95	100.00	77.27
			low	91.67	86.36	83.33	76.14	75.00	92.05	91.67	82.95
		4M	high	83.33	89.77	100.00	86.36	83.33	85.23	75.00	88.64
			low	91.67	82.95	91.67	79.55	83.33	82.95	83.33	82.95
		6M	high	100.00	81.82	83.33	88.64	91.67	86.36	100.00	79.55
			low	83.33	84.09	75.00	85.23	91.67	80.68	83.33	82.95

C Appendix : Detailed Statistics for RQ 2 - Rate of Generation

Table C.1: RQ2 - Desktop - Ubuntu 24.04 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/random	2M	dsk-host-u24	high	0.0059	-0.0009	423.8643	385.3518	1.0216	3.2055	1.0060	2.3671	1.3924	-0.1463
			low	0.0068				0.9466	-0.4640	0.8923	-1.1329	-0.2038	
		mdsk-qemu-u24	high	0.0072	0.0009	0.1199	0.3085	1.9891	4.1867	1.3059	53.1039	3.7160	1.3068
			low	0.0063				2.0144	6.7625	1.6452	44.1523	4.8562	
		mdsk-vbox-u24	high	0.0094	0.0032	0.0296	0.0516	2.5921	3.0143	1.4763	60.8460	0.9329	4.7383
			low	0.0063				2.2672	7.0091	1.4196	55.6334	4.4203	
		mdsk-vmw-u24	high	0.0256	0.0131	0.3524	0.3862	1.1781	2.7935	1.0505	15.1710	1.2737	2.0230
			low	0.0125				1.3414	4.1952	1.0979	26.2766	2.5768	
	4M	dsk-host-u24	high	0.0118	-0.0021	549.7047	444.7477	1.0216	4.3136	1.0072	2.2459	1.8148	-0.4879
			low	0.0139				0.9275	-0.8100	0.9216	-9.9479	-0.8854	
		mdsk-qemu-u24	high	0.0133	0.0008	0.2797	0.4779	2.0569	5.6206	1.3234	54.3548	6.3584	0.6810
			low	0.0126				1.8701	6.3065	1.5583	42.3166	4.3300	
		mdsk-vbox-u24	high	0.0209	0.0086	0.0219	0.0767	2.1428	3.5975	1.3383	54.7077	0.3661	18.0375
			low	0.0123				2.2754	8.5363	1.4165	57.0968	6.6036	
		mdsk-vmw-u24	high	0.0498	0.0259	0.4209	0.5984	1.0851	2.1995	1.0185	7.9048	0.9007	4.3143
			low	0.0239				1.3640	5.8405	1.1078	26.2714	3.8859	
/dev/random	6M	dsk-host-u24	high	0.0176	-0.0028	470.7257	407.0271	1.0181	3.8459	1.0062	1.7525	2.1761	-0.4296
			low	0.0204				0.9430	-0.6975	0.9147	-6.8554	-0.9348	
		mdsk-qemu-u24	high	0.0191	0.0002	1.4293	0.8405	2.1340	8.4653	1.3616	54.7952	7.8928	0.6276
			low	0.0188				1.8335	6.6985	1.4800	42.6276	4.9533	
		mdsk-vbox-u24	high	0.0328	0.0144	0.0685	0.2173	1.8021	5.1964	1.2226	44.1082	2.8432	2.5268
			low	0.0184				2.1191	8.7193	1.3585	53.6123	7.1842	
		mdsk-vmw-u24	high	0.0739	0.0384	0.3014	0.3839	1.0797	2.1696	1.0358	7.4094	1.1990	2.7547
			low	0.0355				1.2661	5.6112	1.0853	21.5866	3.3030	

Table C.2: RQ2 - Desktop - Ubuntu 24.04 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/urandom	2M	dsk-host-u24	high	0.0059	-0.0009	337.5484	322.8851	1.0125	1.7105	1.0020	1.1435	1.6938	-0.1279
			low	0.0068				0.9297	-0.6172	0.9068	-6.4738	-0.2166	
		mdsk-qemu-u24	high	0.0072	0.0008	0.1063	0.3643	1.7278	3.2240	1.2058	45.2037	3.4782	1.4873
			low	0.0063				2.0693	6.9228	1.6772	47.1567	5.1731	
		mdsk-vbox-u24	high	0.0099	0.0036	0.0392	0.0483	2.5146	2.9422	1.4949	61.4491	0.7765	4.6827
			low	0.0064				1.9351	5.3123	1.2896	49.0289	3.6359	
		mdsk-vmw-u24	high	0.0256	0.0133	0.2928	0.3661	1.1715	2.5589	1.0517	14.4315	1.5026	2.2242
			low	0.0124				1.4474	5.3199	1.1503	30.8968	3.3421	
		dsk-host-u24	high	0.0118	-0.0021	434.8598	404.5438	1.0182	3.3422	1.0070	1.6824	2.4018	-0.3177
			low	0.0138				0.9527	-0.5070	0.9114	-1.6086	-0.7630	
/dev/urandom	4M	mdsk-qemu-u24	high	0.0133	0.0008	0.2193	0.5341	1.8173	4.4631	1.2538	48.0728	6.1149	0.8685
			low	0.0126				1.9779	6.8858	1.6064	43.7608	5.3107	
		mdsk-vbox-u24	high	0.0215	0.0088	0.0503	0.0581	1.6399	2.1794	1.2139	37.2733	0.0079	454.6642
			low	0.0127				1.5931	4.9573	1.1693	40.0725	3.6131	
		mdsk-vmw-u24	high	0.0498	0.0257	0.8161	1.1962	1.0835	2.2019	1.0172	8.1871	0.5962	8.6649
			low	0.0241				1.5082	5.9315	1.1594	34.3447	5.1657	
		dsk-host-u24	high	0.0176	-0.0028	422.4028	387.5695	1.0145	3.0820	1.0051	1.3039	2.1593	-0.3925
			low	0.0204				0.9550	-0.5657	0.9177	-0.3364	-0.8476	
		mdsk-qemu-u24	high	0.0193	0.0003	0.5903	0.9661	1.8371	6.3419	1.2668	47.2041	7.0137	0.7382
			low	0.0190				1.9206	6.9914	1.5361	45.7597	5.1773	
/dev/urandom	6M	mdsk-vbox-u24	high	0.0334	0.0147	0.1170	0.1348	1.4982	3.2907	1.1495	33.4754	2.3704	1.9040
			low	0.0188				1.5457	5.5078	1.1696	36.6969	4.5133	
		mdsk-vmw-u24	high	0.0739	0.0382	0.4809	0.6115	1.0539	1.8068	1.0141	5.1590	0.4033	7.1852
			low	0.0357				1.2512	5.2064	1.0803	20.1155	2.8974	

Table C.3: RQ2 - Cloud - Ubuntu 24.04 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Ratio		
/dev/random	2M	dsk-host-u24	high	0.0059	-0.0009	423.8643	385.3518	1.0216	3.2055	1.0060	2.3671	1.3924	-0.1463		
			low	0.0068				0.9466	-0.4640	0.8923	-1.1329	-0.2038			
		pvt-vmw-u24	high	0.0107	0.0031	0.0662	0.0657	1.0291	0.1665	1.0016	3.9530	0.8162	-0.1404		
			low	0.0076				1.0064	0.1018	0.9869	0.1124	-0.1146			
		pub-aws-u24	high	0.0153	0.0086	0.0399	0.0387	1.0017	0.0084	0.9674	-1.1515	0.8543	1.0358		
			low	0.0067				1.0451	0.4974	1.0326	5.5666	0.8849			
		pub-ggl-u24	high	0.0148	0.0065	0.0001	0.0001	0.9500	-0.0445	0.9091	2.6039	1.1180	1.5478		
			low	0.0084				1.0285	1.5185	1.0236	1.4124	1.7305			
		/dev/random	4M	dsk-host-u24	high	0.0118	-0.0021	549.7047	444.7477	1.0216	4.3136	1.0072	2.2459	1.8148	-0.4879
					low	0.0139				0.9275	-0.8100	0.9216	-9.9479	-0.8854	
				pvt-vmw-u24	high	0.0210	0.0062	0.0748	0.0745	1.0370	0.2549	0.9898	3.3716	0.8750	3.2435
					low	0.0148				1.0189	0.3355	1.0024	1.2033	2.8380	
pub-aws-u24	high			0.0295	0.0167	0.0260	0.0266	1.0463	0.3269	1.0218	4.5796	0.6523	1.5119		
	low			0.0129				1.0170	0.3209	0.9965	-0.2241	0.9863			
pub-ggl-u24	high			0.0815	0.0651	0.0000	0.0000	0.3216	-0.6513	0.3134	1.3252	1.9108	0.8774		
	low			0.0164				1.0347	1.9020	1.0330	0.8315	1.6765			
/dev/random	6M			dsk-host-u24	high	0.0176	-0.0028	470.7257	407.0271	1.0181	3.8459	1.0062	1.7525	2.1761	-0.4296
					low	0.0204				0.9430	-0.6975	0.9147	-6.8554	-0.9348	
				pvt-vmw-u24	high	0.0305	0.0087	0.0450	0.0445	0.9996	-0.0032	0.9941	-1.6910	0.3090	2.3328
					low	0.0219				1.0356	0.9900	1.0030	4.2768	0.7208	
		pub-aws-u24	high	0.0443	0.0251	0.0183	0.0192	0.9991	-0.0079	0.9713	1.2538	0.5203	1.8014		
			low	0.0192				1.0154	0.4241	1.0077	1.8369	0.9372			
		pub-ggl-u24	high	0.1871	0.1597	0.1443	0.0325	0.2038	-1.1084	0.2007	1.1982	0.3022	6.3241		
			low	0.0275				0.9049	-0.1077	0.9026	0.8555	1.9109			

Table C.4: RQ2 - Cloud - Ubuntu 24.04 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/urandom	2M	dsk-host-u24	high	0.0059	-0.0009	337.5484	322.8851	1.0125	1.7105	1.0020	1.1435	1.6938	-0.1279
			low	0.0068				0.9297	-0.6172	0.9068	-6.4738	-0.2166	
		pvt-vmw-u24	high	0.0106	0.0030	0.0648	0.0646	1.1903	1.0439	1.0695	16.5080	0.9544	-0.0263
			low	0.0076				1.0332	0.5158	1.0119	4.3862	-0.0251	
		pub-aws-u24	high	0.0153	0.0087	0.0382	0.0389	1.0785	0.3903	1.0373	5.4539	0.6791	1.2518
			low	0.0066				1.0382	0.4164	1.0190	2.5160	0.8501	
		pub-ggl-u24	high	0.0134	0.0051	0.0283	0.0359	1.0274	0.3945	0.9792	-1.0866	-0.1525	-11.9324
			low	0.0084				1.0438	2.0690	1.0331	2.8575	1.8201	
	4M	dsk-host-u24	high	0.0118	-0.0021	434.8598	404.5438	1.0182	3.3422	1.0070	1.6824	2.4018	-0.3177
			low	0.0138				0.9527	-0.5070	0.9114	-1.6086	-0.7630	
		pvt-vmw-u24	high	0.0208	0.0061	0.0658	0.0632	1.0252	0.1758	0.9798	0.9715	0.7346	2.1953
			low	0.0147				1.0732	1.4346	1.0278	6.3771	1.6127	
		pub-aws-u24	high	0.0296	0.0167	0.0275	0.0270	1.0069	0.0504	0.9976	4.1943	0.5766	1.6730
			low	0.0129				1.0329	0.6350	1.0011	2.8586	0.9647	
		pub-ggl-u24	high	0.0253	0.0090	0.0819	0.0990	1.0393	1.0417	1.0129	3.7029	-0.0361	-61.0241
			low	0.0164				1.0277	1.5066	1.0215	1.7827	2.2036	
/dev/urandom	6M	dsk-host-u24	high	0.0176	-0.0028	422.4028	387.5695	1.0145	3.0820	1.0051	1.3039	2.1593	-0.3925
			low	0.0204				0.9550	-0.5657	0.9177	-0.3364	-0.8476	
		pvt-vmw-u24	high	0.0305	0.0086	0.0713	0.0741	1.0308	0.2633	0.9805	0.1273	0.3451	5.7135
			low	0.0219				1.0512	1.1527	1.0300	4.9564	1.9718	
		pub-aws-u24	high	0.0442	0.0251	0.0168	0.0166	1.0608	0.5327	0.9954	6.8212	0.6193	0.9906
			low	0.0192				1.0186	0.5479	1.0065	1.8766	0.6134	
		pub-ggl-u24	high	0.0373	0.0129	0.1165	0.1575	1.0256	0.9447	1.0118	1.4631	0.0584	35.7913
			low	0.0243				1.0368	2.2345	1.0290	1.6294	2.0884	

Table C.5: RQ2 - Desktop - Ubuntu 22.04 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/random	2M	dsk-host-u22	high	0.0061	0.0002	1.1981	1.3492	1.0028	0.2243	0.9966	0.4415	0.6593	3.5241
			low	0.0059				1.0713	4.6496	1.0242	6.8419	2.3235	
		mdsk-qemu-u22	high	0.0065	-0.0004	1.2010	0.9714	2.4783	7.3364	1.5300	58.2804	5.9145	0.5887
			low	0.0069				1.9434	5.0449	1.7367	34.5915	3.4818	
		mdsk-vbox-u22	high	0.0073	0.0007	0.2216	0.4713	2.2477	7.4834	1.3922	57.7223	5.7844	1.4042
			low	0.0065				2.1295	8.8518	1.3566	54.7225	8.1226	
		mdsk-vmw-u22	high	0.0147	0.0038	0.4456	0.4712	1.4762	4.6065	1.1431	31.2366	2.3646	1.3981
			low	0.0109				1.4773	4.9863	1.1618	31.6041	3.3058	
	4M	dsk-host-u22	high	0.0121	0.0004	8.9752	9.2754	1.0023	0.2221	1.0003	0.1664	1.1321	3.2841
			low	0.0117				1.0639	1.9778	1.0420	5.7871	3.7179	
		mdsk-qemu-u22	high	0.0128	-0.0007	1.2916	0.6646	2.6775	7.5626	1.5860	61.6924	6.5665	0.3855
			low	0.0135				1.7486	4.3722	1.5576	33.9974	2.5313	
		mdsk-vbox-u22	high	0.0141	0.0011	0.1381	0.8103	2.0089	7.7170	1.3231	51.7689	6.1761	1.4604
			low	0.0130				2.2156	9.5352	1.3859	56.1659	9.0197	
		mdsk-vmw-u22	high	0.0291	0.0074	0.5443	0.4046	1.5273	6.1034	1.1616	35.9067	3.2900	0.8191
			low	0.0218				1.3381	4.5968	1.1069	25.2196	2.6948	
/dev/random	6M	dsk-host-u22	high	0.0181	0.0008	0.3555	0.4443	1.0011	0.1137	0.9981	0.4247	1.1470	2.3844
			low	0.0173				1.0298	4.3015	1.0135	2.5897	2.7348	
		mdsk-qemu-u22	high	0.0198	0.0003	0.3475	0.3611	2.1853	5.1027	1.5020	51.2033	5.0219	0.6778
			low	0.0195				1.6349	4.4805	1.4115	34.7559	3.4039	
		mdsk-vbox-u22	high	0.0208	0.0014	0.3396	1.0492	1.7287	7.1819	1.2258	44.0668	5.4865	1.5194
			low	0.0194				2.0135	9.0934	1.3173	51.8077	8.3365	
		mdsk-vmw-u22	high	0.0429	0.0106	0.5503	0.5710	1.2632	3.7939	1.0869	20.4125	1.3394	1.6172
			low	0.0323				1.3093	4.4398	1.0837	23.4477	2.1661	

Table C.6: RQ2 - Desktop - Ubuntu 22.04 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Ratio		
/dev/urandom	2M	dsk-host-u22	high	0.0061	0.0002	1.1442	1.3019	1.0248	1.4981	1.0073	2.3537	1.7975	1.7530		
			low	0.0059				1.0840	4.2786	1.0282	7.9213	3.1509			
		mdsk-qemu-u22	high	0.0065	-0.0007	0.8662	0.9273	2.6288	7.0567	1.5588	63.3559	6.3932	0.4986		
			low	0.0073				2.0180	5.0800	1.8107	35.1164	3.1879			
		mdsk-vbox-u22	high	0.0074	0.0008	0.2320	0.4121	2.7521	9.1839	1.5534	65.8817	8.0706	1.0900		
			low	0.0065				2.3105	9.5141	1.4221	58.2171	8.7971			
		mdsk-vmw-u22	high	0.0147	0.0038	0.4772	0.5527	1.3921	3.9515	1.1230	27.8156	1.8994	1.6512		
			low	0.0109				1.5286	5.3292	1.1613	35.4803	3.1364			
		/dev/urandom	4M	dsk-host-u22	high	0.0121	0.0005	5.5970	6.1873	1.0033	0.3342	0.9977	0.1150	0.9720	4.5539
low	0.0116				1.0815	3.1658				1.0445	6.4615	4.4265			
mdsk-qemu-u22	high			0.0130	-0.0012	0.8247	0.5793	2.5465	6.4042	1.4799	62.4254	5.8170	0.6410		
	low			0.0142				1.9058	5.3795	1.6238	40.8978	3.7286			
mdsk-vbox-u22	high			0.0142	0.0013	0.3575	0.4918	2.3940	8.7381	1.4402	60.2291	7.4869	1.1082		
	low			0.0130				2.1148	9.0799	1.3547	54.3594	8.2971			
mdsk-vmw-u22	high			0.0293	0.0074	0.5109	0.6263	1.3282	4.4111	1.1137	25.9452	1.8363	1.5968		
	low			0.0219				1.4143	5.2576	1.1245	29.8391	2.9323			
/dev/urandom	6M			dsk-host-u22	high	0.0180	0.0006	9.5955	7.9194	1.0018	0.1900	1.0009	0.1149	1.3227	2.4246
		low	0.0174		1.0149	0.5309				1.0099	1.8162	3.2068			
		mdsk-qemu-u22	high	0.0197	-0.0005	0.4581	0.4924	2.1055	4.7859	1.4148	52.2990	4.4731	0.7777		
			low	0.0202				1.7494	4.7325	1.4939	32.5092	3.4789			
		mdsk-vbox-u22	high	0.0210	0.0016	0.5602	0.6289	2.0396	8.3627	1.3425	53.0644	7.1806	1.0603		
			low	0.0194				1.9151	8.5717	1.2883	49.2680	7.6135			
		mdsk-vmw-u22	high	0.0435	0.0110	0.5569	0.5408	1.3739	5.1825	1.1092	28.3432	2.0186	1.1554		
			low	0.0325				1.3233	4.5529	1.1045	24.3834	2.3322			

Table C.7: RQ2 - Cloud - Ubuntu 22.04 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Ratio	
/dev/random	2M	dsk-host-u22	high	0.0061	0.0002	1.1981	1.3492	1.0028	0.2243	0.9966	0.4415	0.6593	3.5241	
			low	0.0059				1.0713	4.6496	1.0242	6.8419	2.3235		
		pvt-vmw-u22	high	0.0079	0.0010	0.4163	0.4592	1.0464	0.9198	1.0241	-0.9176	0.1107	45.5451	
			low	0.0070				1.0859	2.2115	1.0501	6.1415	5.0415		
		pub-aws-u22	high	0.0076	0.0012	0.3574	0.3486	1.0565	0.7291	1.0467	5.2955	0.3518	2.8992	
			low	0.0064				0.9958	-0.0773	0.9915	-0.8376	1.0198		
	pub-ggt-u22	high	0.0837	0.0376	0.6733	0.6312	0.1883	-0.6684	0.1811	3.1548	0.9772	1.9549		
		low	0.0461				0.2242	-0.4427	0.2232	0.5029	1.9104			
	/dev/random	4M	dsk-host-u22	high	0.0121	0.0004	8.9752	9.2754	1.0023	0.2221	1.0003	0.1664	1.1321	3.2841
				low	0.0117				1.0639	1.9778	1.0420	5.7871	3.7179	
			pvt-vmw-u22	high	0.0157	0.0020	0.4524	0.4373	1.0948	3.2833	1.0329	7.9698	2.0639	2.1419
				low	0.0137				1.0280	1.2783	1.0134	2.4159	4.4206	
pub-aws-u22			high	0.0150	0.0022	0.5261	0.5064	1.0289	0.6881	1.0003	3.3054	-0.0717	-8.1553	
			low	0.0128				1.0067	0.1896	0.9988	1.1620	0.5850		
pub-ggt-u22		high	0.1958	0.0661	2.4196	1.3211	0.1509	-1.7323	0.1482	0.1712	-1.1077	-0.0287		
		low	0.1297				0.1596	-0.9880	0.1574	2.0129	0.0317			
/dev/random		6M	dsk-host-u22	high	0.0181	0.0008	0.3555	0.4443	1.0011	0.1137	0.9981	0.4247	1.1470	2.3844
				low	0.0173				1.0298	4.3015	1.0135	2.5897	2.7348	
			pvt-vmw-u22	high	0.0235	0.0030	0.2308	0.2342	1.0383	1.4627	1.0137	3.5089	2.5124	1.5473
				low	0.0205				1.0379	2.6097	1.0170	3.4322	3.8873	
	pub-aws-u22		high	0.0224	0.0034	0.6344	0.6327	1.0150	0.5471	1.0062	2.1966	0.3213	1.2100	
			low	0.0191				1.0132	0.5156	1.0061	1.6437	0.3888		
	pub-ggt-u22	high	0.3015	0.1078	0.1845	0.3895	0.1466	-1.6782	0.1434	2.4650	-0.2766	4.1133		
		low	0.1938				0.1569	-1.7072	0.1554	1.7736	-1.1376			

Table C.8: RQ2 - Cloud - Ubuntu 22.04 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/urandom	2M	dsk-host-u22	high	0.0061	0.0002	1.1442	1.3019	1.0248	1.4981	1.0073	2.3537	1.7975	1.7530
			low	0.0059				1.0840	4.2786	1.0282	7.9213	3.1509	
		pvt-vmw-u22	high	0.0080	0.0010	0.3714	0.3761	1.1000	1.3819	1.0620	6.7516	2.5405	2.3096
			low	0.0069				1.1224	2.4030	1.0537	10.2438	5.8674	
		pub-aws-u22	high	0.0080	0.0016	0.0320	0.0576	1.1038	0.5356	1.0744	5.3347	0.2167	4.0396
			low	0.0064				1.0529	0.9131	1.0154	5.4509	0.8753	
		pub-ggl-u22	high	0.0848	0.0745	0.0000	0.0000	0.7506	-0.2182	0.5540	40.1238	1.0413	2.6941
			low	0.0103				1.0290	1.2655	1.0070	1.6018	2.8054	
		4M	dsk-host-u22	high	0.0121	0.0005	6.1873	1.0033	0.3342	0.9977	0.1150	0.9720	4.5539
				low				1.0815	3.1658	1.0445	6.4615	4.4265	
			pvt-vmw-u22	high	0.0158	0.0020	1.7266	1.0823	2.5959	1.0466	7.4876	2.0061	2.2963
				low				1.0699	1.4671	1.0289	7.0425	4.6068	
			pub-aws-u22	high	0.0160	0.0033	0.0325	1.0075	0.0518	0.9692	5.1305	-0.2027	-2.2599
				low				1.0150	0.4551	1.0074	1.1483	0.4580	
			pub-ggl-u22	high	0.2077	0.1595	0.4323	0.3620	-1.3083	0.2873	31.2748	-0.7876	-2.4174
				low				0.4224	-0.4180	0.4202	0.8925	1.9039	
/dev/urandom	6M	dsk-host-u22	high	0.0180	0.0006	9.5955	7.9194	1.0018	0.1900	1.0009	0.1149	1.3227	2.4246
			low	0.0174				1.0149	0.5309	1.0099	1.8162	3.2068	
		pvt-vmw-u22	high	0.0236	0.0031	0.8458	0.7642	1.0773	2.3730	1.0323	4.9057	2.7887	1.5217
			low	0.0205				1.0549	1.6740	1.0235	5.5334	4.2435	
		pub-aws-u22	high	0.0225	0.0034	0.4165	0.4199	1.0263	0.8088	1.0130	2.3431	0.9981	0.4109
			low	0.0190				1.0118	0.4761	1.0079	0.5962	0.4101	
		pub-ggl-u22	high	0.2930	0.1201	0.6893	0.4991	0.1488	-1.6018	0.1468	0.4967	-0.0473	1.4531
			low	0.1729				0.1743	-1.2982	0.1741	0.4604	-0.0687	

Table C.9: RQ2 - Desktop - AlmaLinux 9.4 - /dev/random Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Ratio		
/dev/random	2M	dsk-host-al9	high	0.0053	0.0001	0.8607	0.8760	0.9997	-0.0115	0.9897	0.1418	1.4143	1.0095		
			low	0.0052				0.9987	-0.0476	0.9927	-0.1946	1.4277			
		mdsk-qemu-al9	high	0.0058	0.0004	0.0428	0.0495	1.1143	0.5061	1.0777	-5.8882	3.6347	0.4842		
			low	0.0054				1.1216	2.2493	1.0575	7.7843	1.7599			
		mdsk-vbox-al9	high	0.0065	0.0008	0.2698	0.2689	1.0426	0.3108	1.0188	3.7215	3.1999	0.9630		
			low	0.0058				1.0075	0.0934	1.0045	-0.6671	3.0814			
		mdsk-vmw-al9	high	0.0072	0.0018	0.3387	0.3684	1.8402	6.3490	1.2933	46.0026	4.1598	1.2549		
			low	0.0054				1.6897	6.4663	1.2113	41.4551	5.2203			
		/dev/random	4M	dsk-host-al9	high	0.0105	0.0002	0.7838	0.7594	1.0131	0.8180	1.0063	1.3276	1.5130	0.8454
					low	0.0103				0.9995	-0.0324	0.9979	0.1876	1.2791	
				mdsk-qemu-al9	high	0.0112	0.0005	0.0348	0.0380	1.2794	1.5279	1.1828	13.7574	3.7550	0.2961
low	0.0107				1.0495	1.3319				1.0400	2.3721	1.1119			
mdsk-vbox-al9	high			0.0126	0.0011	0.2928	0.2871	1.0054	0.0469	1.0012	0.8258	2.9336	1.1113		
	low			0.0115				1.0078	0.1144	0.9969	-0.1151	3.2602			
mdsk-vmw-al9	high			0.0132	0.0026	0.6195	0.5868	1.8246	7.5585	1.2772	44.9819	5.7585	1.0283		
	low			0.0106				1.7716	7.3960	1.2517	44.6974	5.9215			
/dev/random	6M			dsk-host-al9	high	0.0157	0.0004	1.0209	0.9558	1.0037	0.3829	0.9999	0.0492	1.3299	0.8547
					low	0.0153				0.9988	-0.1225	0.9968	0.1610	1.1366	
				mdsk-qemu-al9	high	0.0162	0.0002	0.5025	0.4576	1.1186	1.5696	1.0789	7.1007	3.6762	0.5297
		low	0.0161		1.1597	3.0925				1.0832	11.5604	1.9471			
		mdsk-vbox-al9	high	0.0188	0.0016	0.2174	0.2224	1.0630	0.5409	1.0199	5.7051	3.4782	0.9923		
			low	0.0172				1.0115	0.1924	1.0004	1.3115	3.4514			
		mdsk-vmw-al9	high	0.0195	0.0038	0.3518	0.9631	1.5701	6.7504	1.1924	36.4800	4.4752	1.7361		
			low	0.0157				1.9004	8.7567	1.2940	47.5711	7.7694			

Table C.10: RQ2 - Desktop - AlmaLinux 9.4 - /dev/urandom Detailed Statistics Sheet

Interface	Size	Location	Impact	Mean	Mean Gap	SSE Ratio	Variance Ratio	Early PMR	Z for Early Peak	Early Mean Ratio	Drop After Peak	Mean Skewness	Mean Skew Ratio
/dev/urandom	2M	dsk-host-a19	high	0.0053	0.0001	0.8053	0.8032	1.0071	0.2408	1.0001	0.6417	1.3370	1.0751
			low	0.0052				0.9920	-0.2973	0.9907	-0.6202	1.4374	
		mdsk-qemu-a19	high	0.0059	0.0002	13.5345	13.0990	1.3139	1.3246	1.2400	7.3265	3.2186	0.8491
			low	0.0057				3.7973	3.1620	1.9550	69.2210	2.7328	
		mdsk-vbox-a19	high	0.0066	0.0009	0.2314	0.2206	1.0541	0.3918	1.0268	5.9007	2.6939	1.0506
			low	0.0058				1.0091	0.1216	0.9995	-0.4058	2.8301	
		mdsk-vmw-a19	high	0.0070	0.0016	0.3831	0.4609	1.9331	6.9942	1.3014	48.3582	5.0285	1.1698
			low	0.0054				1.8537	7.2811	1.2791	47.1687	5.8822	
	4M	dsk-host-a19	high	0.0105	0.0002	0.9312	0.8988	1.0034	0.2369	0.9974	0.4400	1.3940	0.9657
			low	0.0103				1.0102	0.7225	1.0033	1.1219	1.3461	
		mdsk-qemu-a19	high	0.0113	0.0005	0.0600	0.0536	1.3242	1.8182	1.2379	3.5160	3.0602	0.4474
			low	0.0108				1.0652	1.5087	1.0377	4.4157	1.3690	
		mdsk-vbox-a19	high	0.0128	0.0013	0.2348	0.2375	1.0128	0.1030	1.0006	0.5067	2.9072	1.1347
			low	0.0115				1.0377	0.5570	1.0067	3.5863	3.2987	
		mdsk-vmw-a19	high	0.0133	0.0027	0.4098	0.7348	1.6923	6.7466	1.2249	41.6758	4.9094	1.4053
			low	0.0105				1.9101	8.2114	1.2946	48.3606	6.8991	
/dev/urandom	6M	dsk-host-a19	high	0.0157	0.0004	0.8020	0.7665	1.0244	2.0267	1.0097	2.5922	1.9668	0.5333
			low	0.0153				1.0074	0.6798	1.0027	0.7929	1.0490	
		mdsk-qemu-a19	high	0.0167	0.0006	0.2493	0.2577	1.3087	1.8681	1.2145	19.7380	3.7082	0.4313
			low	0.0161				1.0650	0.7450	1.0375	0.8242	1.5994	
		mdsk-vbox-a19	high	0.0188	0.0017	0.2680	0.2710	1.0307	0.2737	1.0057	0.6142	3.4409	1.0255
			low	0.0172				1.0261	0.4066	1.0151	2.1682	3.5288	
		mdsk-vmw-a19	high	0.0195	0.0038	0.4235	0.4480	1.7949	7.7711	1.2617	45.0842	6.0475	0.9632
			low	0.0157				1.6298	7.4079	1.2123	39.1419	5.8247	

D Appendix : Detailed Statistics for RQ 2 - Quality of Generation

Table D.1: RQ2 - Desktop - /dev/random - Percentage of samples that passed all NIST quality tests

Interface	OS	Size	Impact	Host	Virtual Box	VMWare	QEMU
/dev/random	Ubuntu 24	2M	high	90	85	86	86
			low	84	79	82	75
		4M	high	79	83	78	88
			low	80	78	89	83
		6M	high	88	81	80	85
			low	82	82	85	84
/dev/random	Ubuntu 22	2M	high	86	84	86	84
			low	84	88	87	83
		4M	high	84	81	85	73
			low	85	79	87	84
		6M	high	82	86	88	83
			low	85	79	77	85
/dev/random	Alma Linux 9	2M	high	89	86	82	85
			low	83	89	81	85
		4M	high	79	84	77	82
			low	82	82	81	82
		6M	high	74	85	84	81
			low	75	86	85	83

Table D.2: RQ2 - Desktop - /dev/urandom - Percentage of samples that passed all NIST quality tests

Interface	OS	Size	Impact	Host	Virtual Box	VMWare	QEMU
/dev/urandom	Ubuntu 24	2M	high	86	85	86	89
			low	85	79	80	84
		4M	high	81	87	85	84
			low	87	90	88	79
		6M	high	90	82	72	82
			low	85	79	82	89
/dev/urandom	Ubuntu 22	2M	high	81	89	86	80
			low	77	87	84	83
		4M	high	83	81	88	79
			low	90	82	86	85
		6M	high	85	86	81	83
			low	89	78	84	79
/dev/urandom	Alma Linux 9	2M	high	76	83	84	79
			low	87	79	82	88
		4M	high	89	78	88	86
			low	84	86	87	82
		6M	high	84	85	81	79
			low	84	84	85	83

Table D.3: RQ2 - Cloud - /dev/random - Percentage of samples that passed all NIST quality tests

Interface	OS	Size	Impact	Host	UCSC	AWS	GGLC
/dev/random	Ubuntu 24	2M	high	90	82	88	89
			low	84	87	81	83
		4M	high	79	81	83	86
			low	80	90	82	85
		6M	high	88	87	84	88
			low	82	92	88	85
/dev/random	Ubuntu 22	2M	high	86	88	85	85
			low	84	82	83	85
		4M	high	84	83	85	82
			low	85	86	88	80
		6M	high	82	85	83	84
			low	85	79	81	80
/dev/random	Alma Linux 9	2M	high	89	85	-	-
			low	83	83	-	-
		4M	high	79	80	-	-
			low	82	88	-	-
		6M	high	74	83	-	-
			low	75	82	-	-

Table D.4: RQ2 - Cloud - /dev/urandom - Percentage of samples that passed all NIST quality tests

Interface	OS	Size	Impact	Host	UCSC	AWS	GGLC
/dev/urandom	Ubuntu 24	2M	high	86	83	85	81
			low	85	91	83	80
		4M	high	81	93	86	90
			low	87	84	89	85
		6M	high	90	76	89	83
			low	85	84	81	85
/dev/urandom	Ubuntu 22	2M	high	81	76	80	89
			low	77	87	82	82
		4M	high	83	85	85	81
			low	90	80	82	88
		6M	high	85	88	87	84
			low	89	82	79	90
/dev/urandom	Alma Linux 9	2M	high	76	85	-	-
			low	87	90	-	-
		4M	high	89	86	-	-
			low	84	88	-	-
		6M	high	84	79	-	-
			low	84	87	-	-

Table D.5: RQ2 - Desktop - /dev/random - Percentage of samples that passed all NIST quality tests, split by peak presence

Interface	OS	Size	Impact	Host		Virtual Box		VMWare		QEMU	
				Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak
/dev/random	Ubuntu 24	2M	high	83.33	90.91	83.33	85.23	91.67	85.23	75.00	87.50
			low	83.33	84.09	83.33	78.41	100.00	79.55	75.00	75.00
		4M	high	58.33	81.82	75.00	84.09	75.00	78.41	100.00	86.36
			low	75.00	80.68	66.67	79.55	91.67	88.64	66.67	85.23
		6M	high	83.33	88.64	66.67	82.95	83.33	79.55	83.33	85.23
			low	75.00	82.95	66.67	84.09	83.33	85.23	91.67	82.95
/dev/random	Ubuntu 22	2M	high	83.33	86.36	83.33	84.09	91.67	85.23	83.33	84.09
			low	91.67	82.95	83.33	88.64	91.67	86.36	66.67	85.23
		4M	high	100.00	81.82	83.33	80.68	91.67	84.09	66.67	73.86
			low	83.33	85.23	83.33	78.41	83.33	87.50	75.00	85.23
		6M	high	83.33	81.82	91.67	85.23	83.33	88.64	75.00	84.09
			low	100.00	82.95	91.67	77.27	66.67	78.41	83.33	85.23
/dev/random	Alma Linux 9	2M	high	75.00	90.91	91.67	85.23	91.67	80.68	83.33	85.23
			low	91.67	81.82	83.33	89.77	91.67	79.55	91.67	84.09
		4M	high	91.67	77.27	91.67	82.95	75.00	77.27	83.33	81.82
			low	75.00	82.95	83.33	81.82	100.00	78.41	41.67	87.50
		6M	high	58.33	76.14	100.00	82.95	91.67	82.95	91.67	79.55
			low	75.00	75.00	66.67	88.64	91.67	84.09	66.67	85.23

Table D.6: RQ2 - Desktop - /dev/urandom - Percentage of samples that passed all NIST quality tests, split by peak presence

Interface	OS	Size	Impact	Host		Virtual Box		VMWare		QEMU	
				Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak
/dev/urandom	Ubuntu 24	2M	high	75.00	87.50	83.33	85.23	83.33	86.36	91.67	88.64
			low	66.67	87.50	83.33	78.41	75.00	80.68	83.33	84.09
		4M	high	91.67	79.55	83.33	87.50	66.67	87.50	83.33	84.09
			low	91.67	86.36	83.33	90.91	91.67	87.50	83.33	78.41
		6M	high	83.33	90.91	83.33	81.82	91.67	69.32	58.33	85.23
			low	83.33	85.23	91.67	77.27	75.00	82.95	83.33	89.77
/dev/urandom	Ubuntu 22	2M	high	66.67	82.95	83.33	89.77	91.67	85.23	75.00	80.68
			low	75.00	77.27	100.00	85.23	75.00	85.23	91.67	81.82
		4M	high	91.67	81.82	83.33	80.68	83.33	88.64	91.67	77.27
			low	91.67	89.77	83.33	81.82	83.33	86.36	83.33	85.23
		6M	high	100.00	82.95	83.33	86.36	75.00	81.82	91.67	81.82
			low	91.67	88.64	75.00	78.41	83.33	84.09	75.00	79.55
/dev/urandom	Alma Linux 9	2M	high	75.00	76.14	58.33	86.36	75.00	85.23	75.00	79.55
			low	91.67	86.36	58.33	81.82	75.00	82.95	91.67	87.50
		4M	high	83.33	89.77	58.33	80.68	83.33	88.64	91.67	85.23
			low	91.67	82.95	100.00	84.09	75.00	88.64	66.67	84.09
		6M	high	100.00	81.82	91.67	84.09	75.00	81.82	75.00	79.55
			low	83.33	84.09	75.00	85.23	75.00	86.36	83.33	82.95

Table D.7: RQ2 - Cloud - /dev/random - Percentage of samples that passed all NIST quality tests, split by peak presence

Interface	OS	Size	Impact	Host		UCSC		AWS		GGLC	
				Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak
/dev/random	Ubuntu 24	2M	high	83.33	90.91	75.00	82.95	91.67	87.50	100.00	87.50
			low	83.33	84.09	100.00	85.23	91.67	79.55	83.33	82.95
		4M	high	58.33	81.82	91.67	79.55	100.00	80.68	100.00	84.09
			low	75.00	80.68	83.33	90.91	83.33	81.82	91.67	84.09
		6M	high	83.33	88.64	66.67	89.77	58.33	87.50	100.00	86.36
			low	75.00	82.95	83.33	93.18	75.00	89.77	83.33	85.23
/dev/random	Ubuntu 22	2M	high	83.33	86.36	83.33	88.64	100.00	82.95	83.33	85.23
			low	91.67	82.95	100.00	79.55	100.00	80.68	75.00	86.36
		4M	high	100.00	81.82	91.67	81.82	91.67	84.09	91.67	80.68
			low	83.33	85.23	75.00	87.50	91.67	87.50	58.33	82.95
		6M	high	83.33	81.82	83.33	85.23	66.67	85.23	83.33	84.09
			low	100.00	82.95	83.33	78.41	83.33	80.68	100.00	77.27
/dev/random	Alma Linux 9	2M	high	75.00	90.91	75.00	86.36	-	-	-	-
			low	91.67	81.82	91.67	81.82	-	-	-	-
		4M	high	91.67	77.27	83.33	79.55	-	-	-	-
			low	75.00	82.95	100.00	86.36	-	-	-	-
		6M	high	58.33	76.14	75.00	84.09	-	-	-	-
			low	75.00	75.00	83.33	81.82	-	-	-	-

Table D.8: RQ2 - Cloud - /dev/urandom - Percentage of samples that passed all NIST quality tests, split by peak presence

Interface	OS	Size	Impact	Host		UCSC		AWS		GGLC	
				Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak	Within Peak	After Peak
/dev/urandom	Ubuntu 24	2M	high	75.00	87.50	83.33	82.95	100.00	82.95	75.00	81.82
			low	66.67	87.50	91.67	90.91	83.33	82.95	75.00	80.68
		4M	high	91.67	79.55	91.67	93.18	83.33	86.36	83.33	90.91
			low	91.67	86.36	58.33	87.50	91.67	88.64	75.00	86.36
		6M	high	83.33	90.91	75.00	76.14	83.33	89.77	91.67	81.82
			low	83.33	85.23	91.67	82.95	83.33	80.68	91.67	84.09
/dev/urandom	Ubuntu 22	2M	high	66.67	82.95	58.33	78.41	91.67	78.41	83.33	89.77
			low	75.00	77.27	91.67	86.36	91.67	80.68	75.00	82.95
		4M	high	91.67	81.82	66.67	87.50	66.67	87.50	91.67	79.55
			low	91.67	89.77	100.00	77.27	100.00	79.55	75.00	89.77
		6M	high	100.00	82.95	100.00	86.36	66.67	89.77	91.67	82.95
			low	91.67	88.64	66.67	84.09	83.33	78.41	100.00	88.64
/dev/urandom	Alma Linux 9	2M	high	75.00	76.14	83.33	85.23	-	-	-	-
			low	91.67	86.36	100.00	88.64	-	-	-	-
		4M	high	83.33	89.77	66.67	88.64	-	-	-	-
			low	91.67	82.95	75.00	89.77	-	-	-	-
		6M	high	100.00	81.82	66.67	80.68	-	-	-	-
			low	83.33	84.09	91.67	86.36	-	-	-	-