Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп’ютерних наук
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: Розробка інформаційної системи для тестування кібератак на Cloud-сервіси

Виконав(ла): студент(ка) IV курсу, групи СН-41
спеціальності 122 Комп’ютерні науки
(шифр і назва спеціальності)

Довганюк М.Е.

Керівник _____ Липак Г.І.
(підпис) (прізвище та ініціали)

Нормоконтроль _____ Шимчук Г.В.
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ Боднарчук І.О.
(підпис) (прізвище та ініціали)

Рецензент _____ _____
(підпис) (прізвище та ініціали)

Тернопіль
2025

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Кафедра комп’ютерноих наук
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Боднарчук І.О.
(підпис) (прізвище та ініціали)
«____» _____ 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 122 Комп’ютерні науки
(шифр і назва спеціальності)

Студенту Довганюку Миколі Едуардовичу
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка інформаційної системи для тестування кібератак на Cloud-сервіси

Керівник роботи Липак Галина Ігорівна кандидат наук із соціальних комунікацій (PhD), доцент кафедри комп’ютерних наук
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «07» 05 2025 року №_4/7-444_____

2. Термін подання студентом завершеної роботи

3. Вихідні дані до роботи

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. Розділ 1. Огляд хмарних технологій та актуальні проблеми захисту віртуалізованих обчислювальних середовищ
Розділ 2. Методологія впровадження руткіту CloudSkulk у хмарному середовищі на базі QEMU/KVM. Розділ 3. Віртуалізована атака типу Rootkit-in-the-Middle: метод розгортання CloudSkulk. Розділ 4. Безпека життєдіяльності основи охорони праці. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, слайдів)

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці	Сенчишин В. С., к.т.н., доцент		

7. Дата видачі завдання 27.01.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Ознайомлення з завданням до кваліфікаційної роботи	27.01.2025	
2.	Підбір джерел по темі роботи	31.01.2025-03.02.2025	
3.	Опрацювання публікацій та збір даних по	04.02.2025-06.02.2025	
	темі роботи		
4.	Виконання роботи згідно мети	07.02.2025-11.02.2025	
5.	Оформлення першого розділу «Огляд хмарних технологій та актуальні проблеми захисту віртуалізованих обчислювальних середовищ»	03.06.2025-05.06.2025	
6.	Оформлення першого розділу «Методологія впровадження руткіту CloudSkulk у хмарному середовищі на базі QEMU/KVM»	06.06.2025-08.06.2025	
7.	Оформлення третього розділу «Віртуалізована атака типу Rootkit-in-the-Middle: метод розгортання CloudSkulk»	09.06.2025-11.06.2025	
8	Виконання завдання до підрозділу “Безпека життєдіяльності”	12.06.2025-13.06.2025	
9.	Виконання завдання до підрозділу “Основи охорони		
	праці”	12.06.2025-13.06.2025	
10.	Оформлення кваліфікаційної роботи	16.06.2025-18.06.2025	
11.	Нормоконтроль	19.06.202-20.06.2025	
12.	Перевірка на плагіат	23.06.2025	
13.	Попередній захист кваліфікаційної роботи	21.06.2025	
14.	Захист кваліфікаційної роботи		

Студент _____ Довганюк М.Е.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Липак Г.І.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інформаційної системи для тестування кібератак на Cloud-сервіси // Кваліфікаційна робота освітнього рівня «Бакалавр» // Довганюк Микола Едуардович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра комп'ютерних наук, група СН-41 // Тернопіль, 2025 // С. 78, рис. – 26, табл. – 5, презен. – 12, додат. – 0, бібліогр. – 57.

Ключові слова: кібер атака, віртуальна машина, операційна система, захист.

У роботі було представлено унікальний тип вкладеного руткіту посередині на основі віртуальної машини CloudSkulk, і який може бути використаний, щоб допомогти зловмисникам захопити та приховати свій контроль над цільовою гостьовою віртуальною машиною, що працює в хмарному середовищі.

Руткіт CloudSkulk не прагне отримати доступ або контролювати системні ресурси хмарної платформи, натомість він прагне захопити та зберегти контроль над однією жертвою (гостьовою системою) власник віртуальної машини) шляхом пасивної передачі всіх даних віртуалізації QEMU/KVM між хост-хмарною платформою та гостьовою системою у форматі MITM. Хоча наш дизайн, реалізація та демонстрація CloudSkulk виконані на широко популярному програмному забезпеченні для віртуалізації QEMU/KVM, наш новий тип руткіту може бути застосований ортогонально до інших гіпервізорів, що підтримують два мінімальні атрибути реалізації: програмне забезпечення для віртуалізації повинно надавати утиліту для живої міграції та забезпечувати вкладені гіпервізори.

ANNOTATION

Development of an information system for testing cyberattacks on Cloud services // Qualification work of the educational level «Bachelor» // Dovganyuk Mykola Eduardovych // Ternopil Ivan Pulyuy National Technical University, Computer and Information Systems and Software Engineering Faculty, Computer Sciences Department, group SN-41 // Ternopil, 2025 // P. 78, fig. – 26, tab. – 5, presen – 12, append. – 0, ref. – 57.

Keywords: cyber attack, virtual machine, operating system, protection.

The work presented a unique type of nested rootkit in the middle based on the CloudSkulk virtual machine, and which can be used to help attackers seize and hide their control over the target guest virtual machine running in a cloud environment.

The CloudSkulk rootkit does not seek to access or control system resources of the cloud platform, instead it seeks to capture and maintain control over a single victim (the guest system or the owner of the virtual machine) by passively transmitting all QEMU/KVM virtualization data between the host cloud platform and the guest system in a MITM format. Although our design, implementation, and demonstration of CloudSkulk are performed on the widely popular QEMU/KVM virtualization software, our new type of rootkit can be applied orthogonally to other hypervisors that support two minimal implementation attributes: the virtualization software must provide a live migration utility and provide nested hypervisors.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ І ТЕРМІНІВ

VM (англ. virtual machine) – віртуальна машина.

OS (англ. operating system) – операціна система.

MITM (англ. Man-in-the-Middle) – людина посередині.

SaaS (англ. Cloud Software as a Service) – хмарне програмне забезпечення як послуга.

IT Information Technology) – інформаційна технологія.

PaaS (англ. Cloud Platform as a Service) – хмарна платформа як послуга.

IaaS (англ. Cloud Infrastructure as a Service) – хмарна інфраструктура як послуга.

KVM (англ. Kernel-based Virtual Machine) – віртуальна машина на основі ядра.

ВСТУП

Віртуалізовані хмарні обчислювальні сервіси є ключовим аспектом у галузі програмного забезпечення сьогодні, і є чіткі докази швидкого зростання їх використання. Дослідження ринку прогнозують збільшення хмарних робочих навантажень більш ніж утричі.

Інтеграція системної безпеки є невід'ємною проблемою системних адміністраторів хмарних платформ, яка зі зростанням використання хмарних технологій стає дедалі актуальнішою. Люди, які працюють у хмарі, потребують безпеки більше, ніж будь-коли. У цій роботі буде застосовано наступальний, зловмисний підхід до таких хмарних середовищ, сподіваючись, що як системні адміністратори хмарних платформ, так і розробники програмного забезпечення цих інфраструктур зможуть підвищити безпеку своїх систем [1-3].

Вразливість може існувати на будь-якому рівні комп'ютерної системи. У спільноті безпеки поширена думка, що боротьба між зловмисниками та захисниками визначається тим, яка сторона може використати ці вразливості, а потім отримати контроль на нижньому рівні системи. Через це сприйняття пропонується захист на рівні ядра для захисту від шкідливого програмного забезпечення на рівні користувача, захист на рівні гіпервізора для виявлення шкідливого програмного забезпечення на рівні ядра або руткітів, захист на рівні апаратного забезпечення для захисту гіпервізорів [4-8].

Щойно зловмисники знаходять спосіб скористатися певною вразливістю та отримати певний рівень контролю над системою жертви, збереження цього контролю та уникнення виявлення стає їхнім головним пріоритетом. Для досягнення цієї мети було розроблено різні руткіти. Однак існуючі руткіти мають спільну слабкість: вони все ще виявляються, якщо захисники можуть отримати контроль на нижчому рівні, такому як операційнийсистемний рівень, рівень гіпервізора або рівень обладнання. У роботі ми представляємо

новий тип руткіту під назвою CloudSkulk, який являє собою вкладений руткіт на основі віртуальної машини (VM). Хоча вкладена віртуалізація привернула достатню увагу спільноти безпеки та хмарних технологій, наскільки нам відомо, ми першими виявляємо та демонструємо, що вкладена віртуалізація може бути використана зловмисниками для розробки шкідливих руткітів. CloudSkulk важко виявити, видаючи себе за оригінальний гіпервізор для зв'язку з оригінальною гостьовою операційною системою (OS) та видаючи себе за оригінальну гостьову ОС для зв'язку з гіпервізором, незалежно від того, чи знаходяться захисники на нижчому рівні (наприклад, в оригінальному гіпервізорі) чи на вищому рівні (наприклад, в оригінальній гостьовій OS).

Було проведено різноманітні експерименти з продуктивності, щоб оцінити, наскільки прихований запропонований руткіт і залишається непоміченим, оскільки впровадження ще одного рівня віртуалізації неминуче призводить до додаткових накладних витрат. Наші дані характеристики продуктивності показують, що встановлення нашого нового руткіту в цільовому вкладеному середовищі віртуалізації, ймовірно, залишиться непоміченим, якщо гостьовий користувач не виконує навантаження з інтенсивним вводом-виводом.

РОЗДІЛ 1. ОГЛЯД ХМАРНИХ ТЕХНОЛОГІЙ ТА АКТУАЛЬНІ ПРОБЛЕМИ ЗАХИСТУ ВІРТУАЛІЗОВАНИХ ОБЧИСЛЮВАЛЬНИХ СЕРЕДОВИЩ

1.1. Огляд хмарних середовищ та постановка проблеми досліджень

Переваги хмарних обчислень добре відомі для бізнесу, організацій такінцеві користувачі. Незважаючи на переважні переваги хмарних обчислень, однією з найважливіших проблем цієї швидкозростаючої обчислювальної моделі є безпека. Хоча як дослідники, так і захисники хмарних технологій продовжують розробляти інноваційні досягнення, що покращують безпеку хмар, ці платформи залишаються вразливими до неминучих, невиявлених слабких місць. Метою нашого дослідження є допомога у виявленні однієї такої вразливості.перш ніж зловмисники зможуть скористатися цими системами.

Хмарні обчислення зручно приховують багато складнощів від своїх користувачів, надаючи всі переваги центру обробки даних, водночас уникаючи майже всіх пов'язаних з ним витрат: офісний простір, живлення, охолодження, мережі, пропускну здатність, сервери, сховище, складніпрограмне забезпечення та висококваліфіковані системні адміністратори, які налаштовують, встановлюють, оновлюють та запускаютьці системи. Обчислювальні ресурси, що надаються хмарними платформами, доступні в будь-який час просто через мережеве підключення. Ці ресурси є високомасштабованими, легко налаштовуваними та зазвичай оцінюються як послуга з оплатою за використання або щомісячна оплата. Модель хмарних обчислень трансформує індустрію програмного забезпечення. Незважаючи на надзвичайно впливову роль хмарних обчислень у індустрії програмного забезпечення,існують серйозні перешкоди, які перешкоджають ширшому впровадженню моделі.

Програмна інфраструктура та фізичне обладнання, відповідальне за розміщення хмарних сервісів є спільними для всіх його користувачів; що ускладнює ізоляцію, втрату та захист цілісності даних користувачів від зловмисних осіб. Згідно з опитуванням, «загальні проблеми безпеки» хмарних

сервісів були головною перешкодою для впровадження хмарних технологій підприємствами, причому 53% цих організацій поділяли це занепокоєння [9]. Хоча це важко кількісно оцінити, про інциденти безпеки хмарних сервісів щороку повідомляють державним органам та громадськості, що підтверджує ці занепокоєння.

Багато програм, які керують надзвичайно конфіденційними та важливими даними користувачів, існують сьогодні в хмарі, незважаючи на ризик, спричинений цими недосконалими системами. Наприклад, хмарний додаток Google Compute Engine (GCE) надає хмарну інфраструктуру як послугу (IaaS) для підприємств по всьому світу; клієнти, такі як Spotify, Coca-Cola, BestBuy, Motorola, HTC та сотні інших. GCE обслуговує багато галузей, починаючи від охорони здоров'я та технологій, і закінчуючи найвідомішою з них: фінансовими послугами. Візьмемо, наприклад, мобільний додаток онлайн-платежів для смартфонів, mCash [14]. Додаток, розміщений на GCE, віртуальний банк, дозволяє користувачам надсилати та отримувати гроші, оплачувати рахунки та переглядати транзакції. Багато реальних хмарних програм, таких як mCash, які вимагають безпеки для своїх користувачів, є потенційними цілями для нового типу шкідливої атаки безпеки, яку ми називаємо CloudSkulk.

У цій роботі ми представляємо новий тип методології атак на безпеку програмного забезпечення, пов'язаний з добре відомими атаками типу "людина посередині" (MITM). Наша методологія використовує вкладену стратегію на основі віртуальних машин (VM) QEMU/KVM, яку ми називаємо CloudSkulk. Ми стверджуємо, що віртуальна машина, контрольована шкідливим джерелом, може являти собою руткіт, який може непомітно підслуховувати обмін даними між гостьовим та хостовим серверами QEMU/KVM пари на хмарній платформі GNU/Linux, що надає хмарні послуги IaaS. Завдяки нашій атаці ми можемо отримати 100% видимість даних та живої взаємодії цільового гостя, що працює в одному з цих віртуалізованих середовищ. Однією з визначальних рис нашого нового типу атаки є те, що вона може дозволити зловмиснику залишатися непоміченим протягом тривалого часу. Наша мета полягає не в тому, щоб посилювати проблеми безпеки хмари, а в тому, щоб підвищити обізнаність та

виявити вразливості їхніх систем постачальниками та розробниками хмарних послуг у надії, що вони посилять свою безпеку.

Сучасні центри обробки даних – це об'єкти, що містять десятки тисяч комп'ютерів зі значною пропускною здатністю мережі. Вони підтримують великі набори інформаційно-технологічного (IT) обладнання, яке використовується для надання як апаратних, так і програмних послуг стороннім користувачам, підключеним через телекомунікаційну мережу [10]. Сучасні хмарні обчислення відрізняються від традиційних центрів обробки даних тим, що послуги, що надаються хмарою, надаються користувачеві через віртуалізоване середовище, яке називається віртуальною машиною (VM). У цій роботі ми використовуватимемо стандартну термінологію для позначення «хоста» як фізичної машини хмарних обчислень, відповідальної за забезпечення віртуалізації для своїх сторонніх «гостей».

Віртуалізація є основою хмарних обчислень. Віртуалізація стосується створення віртуальної версії деяких фізичних ресурсів, таких як ОС, сервер або пристрій. Програми, що працюють з гостьової віртуальної машини, зазвичай виконуються на хості з тим самим рівнем контролю, що й інші користувацькі програми, у так званому «найнижчому рівні привілеїв».

Виконуваним програмам дозволено доступ до ресурсів машини на основі системи категорій привілеїв виконання. Для керування та контролю груп у цій категоріальній системі сучасні набори інструкцій процесорів x86 називають ці різні групи кільцями. Права на виконання варіюються від найвищого рівня контролю до найбільшпривілейованого кільця 0, до найменшого контролю в кільцях найнижчого рівня привілеїв 3. Рисунок 1.1 ілюструє цю концепцію привілеїв.

Однак віртуалізованим гостьовим програмам може знадобитися доступ до апаратного забезпечення хоста або привілейованих ресурсів хоста, що вимагають прав доступу типу «ring 0». Щоб підтримувати цілісність безпеки системи, дозволяючи водночас використовувати ці ресурси, сучасні комп'ютери уникають надання прав доступу.Рівень доступу кільця 0 до віртуальних машин рівня користувача (кільце 3) через програмне обхідне рішення, що реалізується

гіпервізором, також званим монітором віртуальних машин (VMM). Гіпервізори надають віртуалізованим гостьовим процесам ресурси хоста через різні типи віртуалізації, які вони надають: часткова, пара- та повна віртуалізація, і це лише деякі з них. Гіпервізор – це, по суті, низькорівневе програмне забезпечення з високими привілеями, яке керує виконанням однієї або кількох гостьових ОС на одній машині. По суті, гіпервізори функціонують, перехоплюючи та емулюючи конфіденційні операції, що виникають у гостьовому середовищі (такі як зміна таблиць сторінок, що може надати гостевій системі доступ до пам'яті хоста, до якої їй не дозволено доступ) [2].

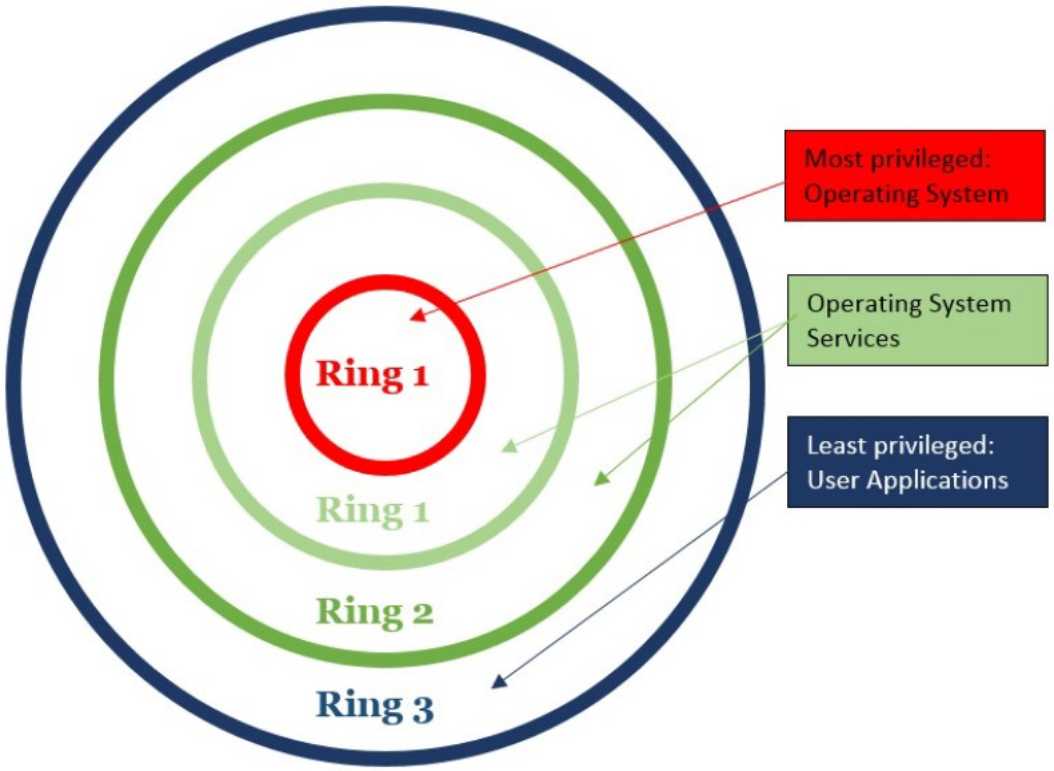

Рисунок 1.1 – Права виконання набору інструкцій процесора

Коли паравіртуалізований гостьовий процес запитує привілейовану команду, інструкцію ОС або невіртуалізовану інструкцію, вона доставляється гіпервізору через HyperCall, подібно до системних викликів Linux. Гіпервізор отримує ці HyperCall,звертається до обладнання, а потім повертає результат.

Однак цей тип віртуалізації вимагає, щоб гостьова ОС була модифікована для правильного використання API HyperCall.

Гостьова система, створена за допомогою повної віртуалізації, отримує повну симуляцію базового обладнання, що дозволяє запускати немодифіковану гостьову ОС.з привілейованим керуванням кільцем 1 на хості. Повну віртуалізацію можна розглядати як використання цілого комп'ютера з повним керуванням, інкапсульованого та запущеного всередині віртуальної машини. Повна віртуалізація повільніша за інші типи віртуалізації [3], такі як пара-, оскільки машинний код з працюючої гостьової ОС має під час виконання бути перетворений на машинну мову хоста всередині гіпервізора хоста. Це можна досягти за допомогою процесу, який називається двійковою трансляцією, якщо процесор хоста підтримує технологію віртуалізації (VT).

Бінарне перетворення перетворює вихідну бінарну програму в цільовий бінарний файл перед виконанням, де запитуваний набір інструкцій або бінарний файл посилається на вихідний файл, а базовий набір інструкцій хост-процесора посилається на цільовий файл. Хоча конфіденційні та невіртуалізовані інструкції повністю віртуалізованої гостьової ОС (що працює з привілеями Ring 1) перетворюються за допомогою бінарного перетворення в гіпервізорі хоста, гостьова віртуальна машина процесу рівня користувача (що працюють з привілеями Ring 3) виконується безпосередньо на головному процесорі для забезпечення високої продуктивності. Сучасні процесори Intel та AMD забезпечують цю підтримку за допомогою розширень набору мов асемблера, відомих як Intel VT-x для процесорів Intel на базі x86 та AMD-v для процесорів типу AMD. Ці розширені команди дозволяють процесору забезпечувати апаратну віртуалізацію. На рисунку 1.2 показано ці основні функціональні відмінності між типами віртуалізації [18].

1.2. Хмарні сервіси

Згідно з визначенням Національного інституту стандартів і технологій (NIST), хмарні обчислення складаються з трьох фундаментальних моделей

обслуговування: хмарне програмне забезпечення як послуга (SaaS), хмарна інфраструктура як послуга (IaaS) та хмарна платформа як послуга (PaaS) [32].

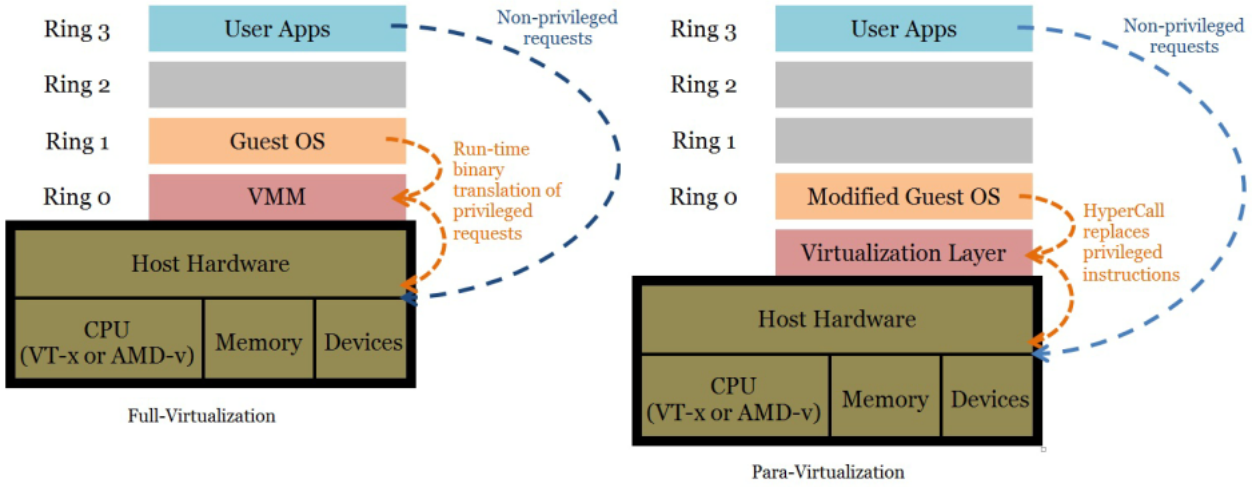

Рисунок 1.2 – Паравіртуалізація та повна віртуалізація

Хмарні SaaS-платформи надають гостям програмне забезпечення. Основна увага в цих хмарних платформах зосереджена на додатках, а не на апаратному забезпеченні. Надане програмне забезпечення часто вимагає суворих та дорогих ліцензій на програмне забезпечення, які зручно обходитигості, які орендують програмне забезпечення з хмари як послугу SaaS. Наприклад, хмарна платформа Microsoft Azure надає користувачам популярний офісний пакет під назвою Microsoft Office 365 як послугу SaaS.

Процес підключення до хмари Azure та використання її сервісів є майже повсюдною формою взаємодії між гостем і хостом для хмарних платформ. Тому розгляд цього процесу ілюструє, як хмарні сервіси зазвичай надаються гостям майже...будь-яке хмарне середовище сьогодні. Гість Azure спочатку вибирає тарифний план за годину на веб-сайті Microsoft на основі набору бажаних конфігурацій та запитуваних послуг. Потім гість підключається до хмари Microsoft через локальну програму порталу Azure на своєму локальному комп'ютері, де він може використовувати Microsoft Office 365, Dynamics 365 або будь-яку іншу запитувану програму з середовища віртуальної машини. Гостьове середовище віртуальної машини Azure, ймовірно, буде створено та спочатку

підтримуватися в рамках одного процесу, фізично запущеного на хмарній машині Microsoft. Продуктивність запущеної програми в хмарі переважно залежить від апаратного та програмного забезпечення хмарної машини. Важливо підкреслити, що це одна з таких переважаючих переваг хмари; орендована програма, що працює в хмарі, ймовірно, працюватиме швидше, ніж якби вона працювала локально. Якщо дорога програма САПР, яка вимагає значної обчислювальної потужності процесора, може зазнати підвищення продуктивності під час роботи в хмарі, користувачі САПР можуть бути переконані цією хмарною характеристикою.

Хмарні платформи на основі PaaS надають набір програмних та апаратних компонентівщо дозволяють гостьовим системам створювати, керувати та запускати програми в хмарі через віртуальну машину. Основна увага цих хмарних платформ приділяється фреймворкам, що пропонуються як послуга, а не окремим програмам чи апаратним компонентам, таким як пам'ять, що орендується як послуга. Велика різниця між послугами PaaS та SaaS полягає в тому, що PaaS призначений більше на ефективне та результативне розміщення налаштованих додатків для своїх гостьових користувачів, які зазвичай самі є підприємствами або організаціями. Ці підприємства або організації, що використовують PaaS-сервіси в хмарі, потім зазвичай розгортають свої додаткидля інших кінцевих користувачів їхнього продукту. Що стосується хмарних платформ SaaS, їхні гостьові користувачі, як правило, є лише окремими кінцевими користувачами.

Google App Engine – це хмарна платформа, яка слугує гарним прикладом хмари, що надає PaaS-послуги. Google App Engine надає своїм гостям масштабовані веб-додатки та мобільні сервери за допомогою фреймворку, що включає такі інструменти, як: NoSQL-сховища даних, memcache та API автентифікації користувачів. Компанія-розробник мобільних ігрових додатків Pocket Gems – один із таких гостьових сервісів, який використовує PaaS-послуги для створення та розгортання своїх ігор для інших кінцевих користувачів. Google Cloud розміщує ці додатки, запускає ігри Pocket Gem та надає інфраструктуру для динамічної кількості кінцевих користувачів. Фактично, однією з суттєвих

переваг PaaS-сервісів є те, що розміщені додатки можна автоматично масштабувати у відповідь на великі вимоги до обчислень та мережевого трафіку. Візьмемо іншу компанію Niantic, яка працює в хмарі Google, яка випустила мобільний додаток у 2016 році під назвою PokemonGO. Google повідомляє, що «протягом 15 хвилин після запуску в Австралії та Новій Зеландії трафік гравців значно перевищив очікування Niantic». [28] Чітку ілюстрацію неочікуваних рівнів трафіку гравців PokemonGo та динамічної потужності моделі хмарних обчислень можна побачити на рисунку 1.3.

Хоча цей фактор масштабованості та динамічна якість хмарних обчислень, що показано на рисунку 1.3, справедливі для SaaS-сервісів, великомасштабні програми, призначені для спочатку невідомої кількості кінцевих користувачів, отримують найбільшу користь від PaaS-сервісів. Ці переваги є однією з головних мотивацій для підприємств та організацій переходити від локальних, дорогих та внутрішньо керованих серверних інфраструктур до PaaS-сервісів на основі хмарних платформ.

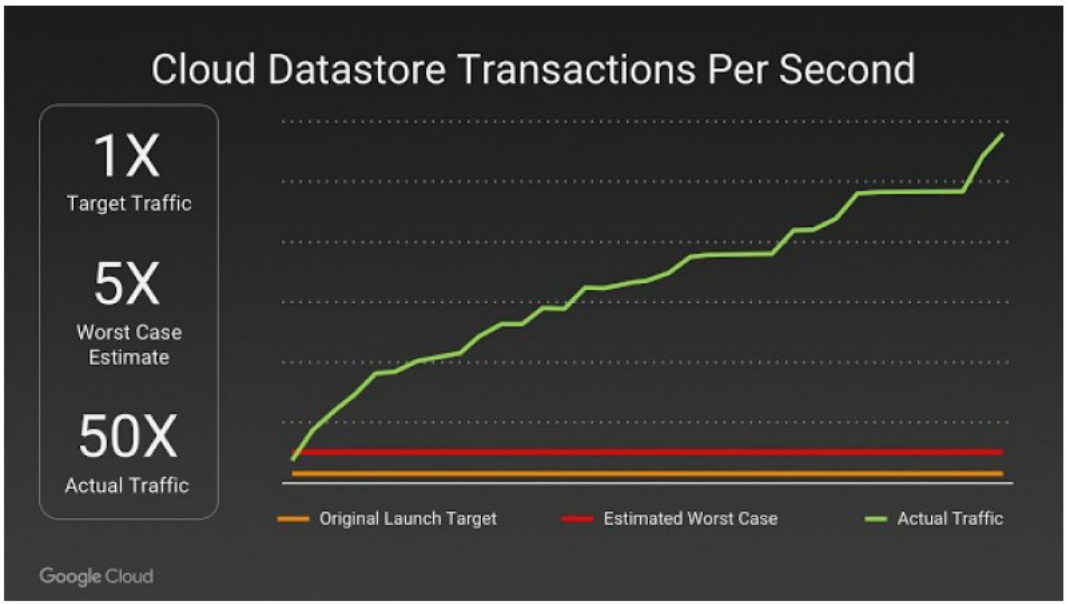

Рисунок 1.3 – Відповідь масштабованості Google Cloud на рішення PokemonGO

Хмарні платформи IaaS надають своїм гостям особливий вид віртуалізованого середовища, відомого як повна віртуалізація. Постачальники хмарних послуг на основі IaaS надають користувачам цілий комп'ютер – простір

користувача, ОС та обладнання – віртуалізований та доступний через хмару. Як і гостьові користувачі послуг PaaS, гості послуг IaaS зазвичай, хоча й не обмежуються ними, є підприємствами чи організаціями. Наприклад, Amazon Web Services (AWS) надає послуги повної віртуалізації IaaS для відомого гостя, який є світовим лідером у наданні онлайн-потокового медіа та відео на вимогу, Netflix. Netflix – це розважальний бізнес, який використовує послуги IaaS на AWS для розміщення онлайн-контенту для 98,75 мільйона кінцевих користувачів [46]. На рисунку 1.4 показано перспективу гостьового користувача проти хмарної платформи [18] послуг, які кожна з них контролює та керує ними, залежно від типів хмарних обчислень.

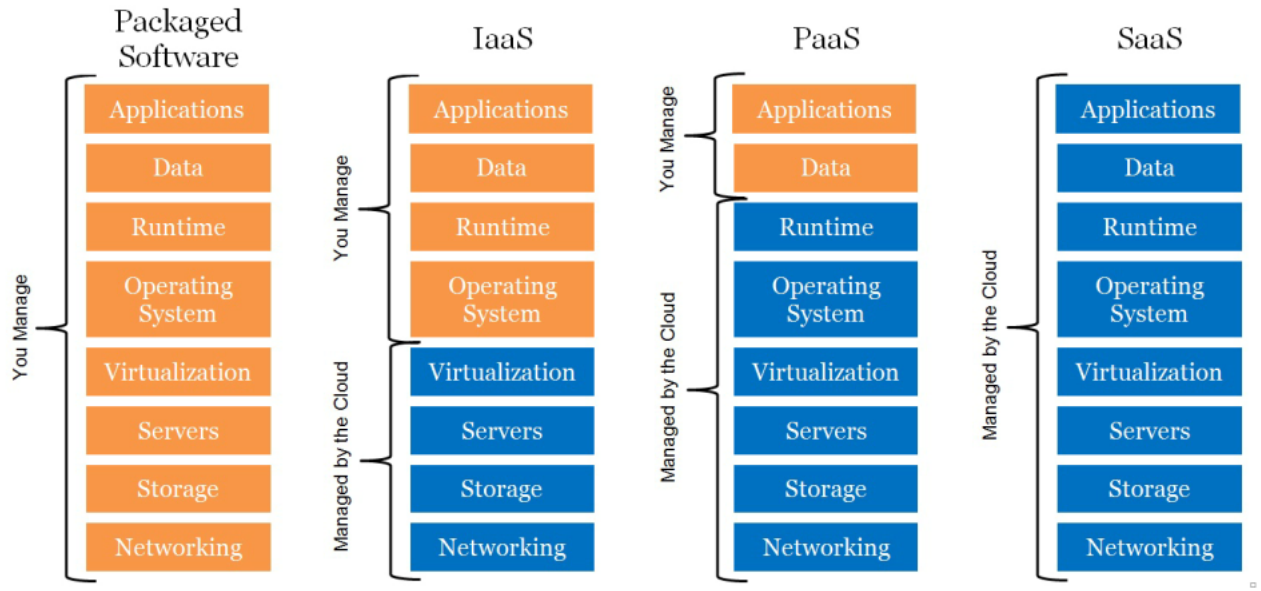

Рисунок 1.4 – Відповідальність користувача за ресурси в хмарі проти відповідальності хмари

1.3. Вкладена віртуалізація

Хоча віртуалізація розпочалася в 1960-х роках, лише у 2010 році дослідники з IBM вперше представили концепцію вкладеної віртуалізації на архітектурах x86 у своєму проекті Turtles [6]. Вони реалізували вкладену віртуалізацію в гіпервізорі Linux KVM 2. Пізніше вкладена віртуалізація була прийнята та реалізована в Xen (починаючи з Xen 4.4).

Концепція вкладеної віртуалізації проста: гіпервізор працює всередині віртуальної машини. Традиційна віртуалізація передбачає одночасне запуск кількох ОС поверх одного й того ж гіпервізора. Вкладена віртуалізація відрізняється від цієї моделі тим, що дозволяє кільком гіпервізорам одночасно працювати поверх одного й того ж гіпервізора.

У проекті Turtles [6] група дослідників представила концепцію віртуалізації Level0 (L0), Level1 (L1) та Level2 (L2), яку ми будемо використовувати в цій роботі. Level0 представляє гіпервізор, що працює поверх реального обладнання, Level1 представляє гіпервізор, що працює поверх Level0, як гостьова система, а Level2 представляє гіпервізор.що працює поверх рівня 1 як вкладена гостьова система. На рисунку 2.5 показано ці рівні віртуалізації гіпервізора.

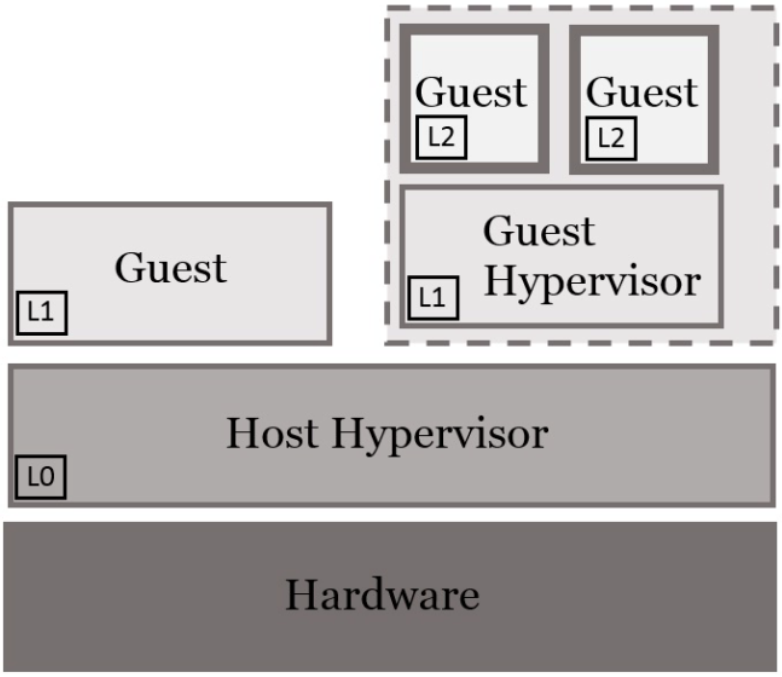

Рисунок 1.5 – Рівні віртуалізації гіпервізора

Проект Turtles був розроблений без достатньої підтримки архітектури процесора для вкладеної віртуалізації. Відтоді постачальники процесорів розробили різні функції у своїх продуктах для підтримки вкладеної віртуалізації

в надії покращити продуктивність вкладеної віртуалізації. Наприклад, Intel представила структуру керування віртуальною машиною (VM Control Structure).

Функція тіньового відтворення (VMCS Shadowing) у процесорах Haswell. Ця функція розширює можливості Intel-VT, з особливим акцентом на усунення VM-виходів, спричинених інструкціями VMREAD та VMWRITE, що виконуються гіпервізором рівня 1. Завдяки цим покращенням VMCS Shadowing, частоту перемикання контексту між гіпервізорами рівня 1 та рівня 0 можна значно зменшити, що підвищує продуктивність вкладеної віртуалізації.

Одним із двох основних обмежень дизайну, що накладаються на руткіт CloudSkulk 3, є те, що гостьове середовище віртуальної машини забезпечується можливостями вкладеної віртуалізації. У цій роботі ми зосереджуємося на хмарних платформах на базі IaaS, оскільки саме ці типи хмарних сервісів традиційно пропонують гостьовим системам повну віртуалізацію, а отже, і вкладені сервіси віртуалізації. Саме цю функцію вкладеної віртуалізації, як ми побачимо в наступних розділах, ми можемо використати для зловмисної атаки на хмарні платформи на базі IaaS за допомогою нашого унікального типу руткіту.

1.4. Віртуалізація KVM/QEMU

Ми зосереджуємося на широко поширеному гіпервізоре Linux – віртуальній машині на базі ядра (KVM) та її застосунку для користувацького простору Quick EMUlator (QEMU). Багато провідних компаній у галузі програмного забезпечення для хмарних обчислень IaaS наразі використовують KVM/QEMU для розміщення своїх хмарних платформ. Варто згадати лише деякі з таких популярних хмарних платформ: Google Compute Engine (GCE), Amazon Web Services (AWS) та IBM SmartCloud Enterprise.

KVM – це основний гіпервізор Linux, який забезпечує базову програмну підтримку, щоб дозволити програмам у просторі користувача використовувати фізичне обладнання комп'ютера не дозволяючи програмам у просторі користувача безпосередньо взаємодіяти з фізичнимипристрої. KVM – це тип гіпервізора, який називається архітектурою емуляції пристроїв у просторі

користувача [20], тобто він забезпечує лише підтримку емуляції пристроїв, а не фактичне створення та виконання таких пристроїв. KVM встановлюється в операційній системі Linux за замовчуванням як модуль ядра у версіях 2.6.20 та вище.

QEMU – це програмний застосунок у просторі користувача, який відповідає за створення екземплярів віртуалізованих пристроїв, а також середовища, в якому працює гостьова ОС. QEMU створює процес Linux для кожного екземпляра віртуальної машини, причому кожен процес інкапсулює все середовище віртуальної машини. Віртуалізовані середовища, що використовують KVM/QEMU, можуть мати майже нативну продуктивність для типових робочих навантажень, як показано у Virtual Open Systems [43]. Для стислості ми можемо називати парадигму KVM/QEMU лише QEMU до кінця цієї статті. У наведеному нижче списку детально описано деякі ключові атрибути цих двох програмних інструментів, що застосовуються до нашого руткіту.

QEMU:

- Модифікована гостьова ОС, що дозволяє використовувати вкладені гіпервізори.
- Модифікована гостьова ОС (повна віртуалізація) надає зловмиснику багатий набір ресурсів для зменшення кількості сигнатур безпеки та аномалій, які можуть призвести до виявлення.
- Гостьова ОС інкапсульована з хост-машини в середовищі віртуальної машини
- Вкладені гостьові ОС інкапсульовані з хост-машини в середовищі віртуальної машини
- Гостьова віртуальна машина створюється та підтримується в межах одного процесу Linux хоста.
- Екземпляри віртуалізованого обладнання, створені для гостьової ОС
- Для підвищення продуктивності гостьової ОС можна створити паравіртуалізоване обладнання.
- Підтримує утиліту живої міграції простору користувача 4

- Стандартне програмне забезпечення для віртуалізації для хмарних платформ на базі IaaS

KVM:

- Містить реалізаціюапаратних пристроїв хоста для повної віртуалізації
- Надає можливість емуляції пристрою програмам у просторі користувача через KVM API
- Стандартний гіпервізор для хмарних платформ на базі IaaS

Жива міграція

У своїй новаторській роботі Кларк та ін. [8] запропонували та впровадили міграцію віртуальних машин у режимі реального часу. Вони визначили міграцію віртуальних машин у режимі реального часу як процедуру міграції всієї ОС та всіх її програм як єдиного цілого з однієї хост-машини на іншу. Переваги міграції віртуальних машин у режимі реального часу полягають головним чином у двох аспектах: (1) забезпечення чіткого розділення апаратного та програмного забезпечення та (2) забезпечення відмовостійкості, балансування робочого навантаження та низькорівневого обслуговування системи. Вони впровадили рішення для міграції у режимі реального часу перед копіюванням у середовищі віртуалізації на базі Xen та стверджували, що двома найважливішими показниками є: загальний час міграції та час простою сервісу. Перший, також відомий як наскрізний час, стосується загального часу, необхідного між початком та завершенням міграції, тоді як другий вказує на тривалість коли віртуальна машина призупинена, а служба недоступна. На рисунку 1.6 наведено графічне зображення міграції віртуальної машини до копіювання.

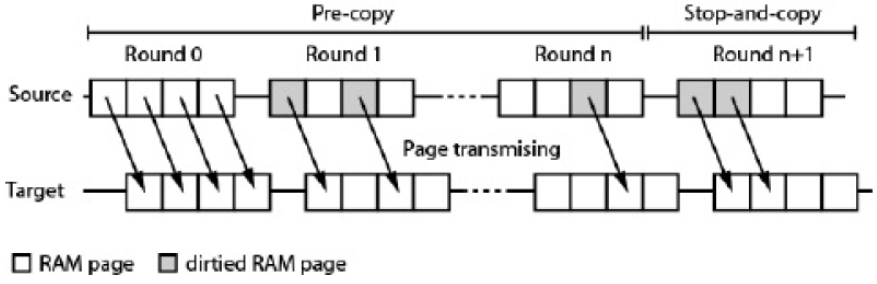

Рисунок 1.6 – ПопереднєкопіюванняАлгоритм живої міграції [40]

З моменту появи живої міграції віртуальних машин, міграція привернула значний інтерес з боку спільноти віртуалізації та хмарних обчислень і стала критично важливою функцією в основних гіпервізорах, включаючи Xen, Qemu/KVM, VMware та Hyper-V. Зазвичай жива міграція віртуальних машин на основі попереднього копіювання включає чотири кроки, як детально описано нижче.

Підготовка. На цьому кроці дві хост-машини, одна з яких служить джерелом, а інша – пунктом призначення, повинні перейти в режим готовності до міграції. Зокрема, сторона призначення повинна прослуховувати певний порт, очікуючи запиту на міграцію, надісланого від сторони джерела. Щойно запит на міграцію, надісланий від джерела, буде отримано пунктом призначення, обидві сторони почнуть встановлювати TCP-з'єднання.

Ітеративне копіювання. Після встановлення TCP-з'єднання пам'ять (та/або диск) віртуальної машини буде скопійовано ітераційно. На першій ітерації всі сторінки будуть передані з джерела на місце призначення. Після цього всі сторінки, які будуть пошкоджені під час першої ітерації, будуть скопійовані на місце призначення, і така процедура буде повторюватися ітеративно. Таке ітераційне копіювання може вплинути на продуктивність системи, оскільки воно споживає такти процесора, пам'ять та пропускну здатність мережі.

Зупинка та копіювання. Якщо взаємодія гостьової системи призводить до частих та інтенсивних змін пам'яті та образу диска, спочатку потрібно зупинити віртуальну машину на стороні джерела, а потім повністю скопіювати решту «несправних» розділів пам'яті та образу диска. Саме на цьому кроці виникає простій служби. Якщо ця частина, що залишилася, достатньо велика, простій служби може бути значним.

Активація. Після завершення операції копіювання віртуальна машина на стороні призначення буде активована. На стороні джерела віртуальна машина буде призупинена, і вихідний хост зможе її видалити.

Наведені вище кроки зображують основну процедуру міграції в реальному часі перед копіюванням. В останніх версіях гіпервізорів також реалізовано підхід до міграції в реальному часі після копіювання. Як правило, після копіювання

відбувається менший час простою, але попереднє копіювання є надійнішим. У цій роботі ми використовуємо попереднє копіювання, але постачальники хмарних послуг можуть використовувати як попереднє, так і кінцеве копіювання. Метод руткітів, який ми представляємо, застосовується до обох підходів до міграції. Крім того, описаний вище метод міграції зазвичай здійснюється між двома фізичними машинами, але це не зовсім так. Наприклад, міграція в реальному часі може використовуватися хостом для динамічного налаштування віртуалізованих пристроїв гостя під часпід час виконання. Гостьову віртуальну машину з 4 ГБ віртуальної пам'яті DIMM можна перенести на нову гостьову віртуальну машину з 8 ГБ доданої віртуальної пам'яті DIMM. У цій роботі, включаючи наше проектування, реалізацію та оцінку, нам потрібна лише одна фізична машина для запуску руткіту на основі вкладеної віртуальної машини. Іншими словами, дві фізичні машини не потрібні для розробки CloudSkulk.

1.5. Забезпечення безпеки

На типовій хмарній обчислювальній машині Linux, на якій розміщені віртуалізовані сервіси QEMU, KVM працює з привілейованими правами суперкористувача, а QEMU працює з менш привілейованими правами доступу до простору користувача. Гостьові системи, зареєстровані в процесах QEMU, не мають дозволів на хост-машині або доступу до неї, якщо вони явно не ініціалізовані як такі під час створення процесу QEMU на хості. Однак, якщо гостьовий процес виявляє помилку та може її використати, як-от virtunoid: breakout of KVM exploit у 2011 році [13], гість може отримати частковий або навіть повний контроль над хост-машиною. Хоча кількісно оцінити це важко, про інциденти хмарної безпеки щороку повідомляють державним органам або громадськості. Фактично, публічно повідомлені помилки QEMU містяться у звітах про помилки Common Vulnerability and Exposure (CVE) на CVEDetails.com [34]. Програмні помилки VM та VMM різняться за ступенем серйозності та проявляються в дослідницькій спільноті з безпеки програмного забезпечення

[13, 11, 50, 23, 34, 49]. Ці незалежні групи змогли створити так звані віртуальні машини-ескейпи, що дозволяли їм повністю контролювати цільову ОС.

Окрім цього обмеженого типу контролю доступу до хост-машини, було написано багато книг, у яких обговорюється, як використовувати помилки безпеки та проникати в комп'ютерні системи та програмне забезпечення [30, 24, 29]. Також добре відомо, що для певної мережі поєднання умов безпеки та низькорівневих вразливостей може призвести, і призводить, до контролю над хостом. Найпоширенішою формою атаки на безпеку, яка загрожує можливому контролю над хостом через мережеві вразливості, є переповнення буфера [10]. Це також можна проілюструвати низкою поточних досліджень, спрямованих на виявлення та запобігання цим типам мережевих атак [7, 37, 44, 35, 27]. На рисунках 1.7 та 1.8 зображено візуальні зображення двох типових типів атак, які призводять до отримання контролю над хостом, як описано вище: атаки типу «вихід з віртуальної машини» та атаки типу «вразливість мережі хоста». На обох рисунках пунктирні чорні стрілки представляють звичайні шляхи зв'язку, а пунктирні червоні лінії представляють собою розширення цього звичайного використання, що зображує режим загрози.

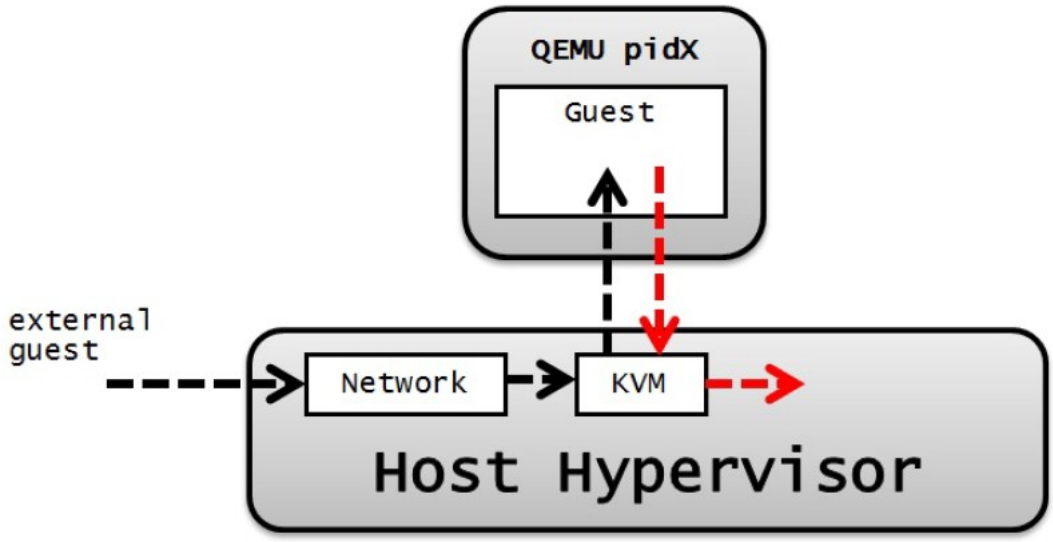

Рисунок 1.7 – Режим 1 потоку вторгнення хоста – вихід з віртуальної машини

Наша шкідлива атака можлива в контексті будь-якого з цих типів подій потокового режиму. Ми представимо наш проект та реалізацію в контексті

режиму загрози 1, роблячи сильне припущення, що ми є гостьовим процесом QEMU, який отримав можливість виходу з віртуальної машини. Важливо зазначити, однак що будь-яка дійсна атака, яка отримує контроль над хостом, дозволить реалізувати нашу атаку. Гість, який отримав доступ до віртуальної машини, може не лише викликати команди всередині хоста, але також намагаються захистити власні внутрішні дані віртуальної машини, атакуючи гіпервізор хоста та/або інструменти системи виявлення вторгнень (IDS) хоста [15].

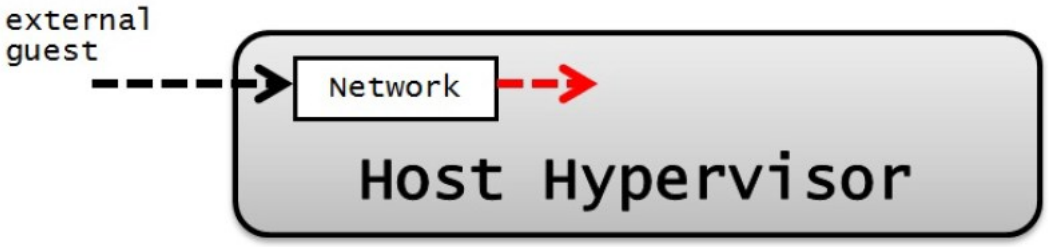

Рисунок 1.8 – Режим 2 потоку вторгнення хоста – вразливість мережі

Характеристики скомпрометованого хост-комп'ютера забезпечують потужну програмну інфраструктуру для реалізації відомої парадигми ізольованих, важковиявлюваних та потенційно шкідливих програм, відомих як руткіти. Хоча не всі руткіти є шкідливими, їх можна визначити як невеликий набір програм, що забезпечують постійний, тривалий та непомітний привілейований доступ до комп'ютера. Загальною стратегією підтримки привілейованого доступу є прихованість. Руткіти прагнуть приховати код та/абодані в комп'ютерній системі, намагаючись залишитися непоміченими. Саме ця ключова характеристика прихованості дозволяє нам концептуально еквівалентувати віртуальну машину руткіту. Це фундаментальна асоціація, яку ми використовуємо – віртуальна машина може діяти як руткіт – і ключова концептуальна характеристика нашого унікального типу руткіту.

Добре відомий та вивчений тип безпеки програмного забезпечення «людина посередині» (MITM)Атаки можна розділити на два різні типи: пасивні та активні. Пасивні MITM-атаки визначаються зловмисною стороною,

«Людиною», яка підслуховує повідомлення, що надсилаються між однією або кількома парами користувачів. Активні MITM-атаки визначаються аналогічно зловмисною стороною, яка, окрім підслуховування, змінює повідомлення, що надсилаються між однією або кількома парами користувачів [21]. Рисунок 1.9 – це візуальне представлення зловмисної атаки типу MITM, де пунктирні чорні стрілки позначають звичайні шляхи зв'язку, а пунктирні червоні лінії – недозволений шлях зв'язку.контролюється пасивно або активно третьою стороною «Людиною».

Рисунок 1.9 – Модель атаки на безпеку програмного забезпечення MITM

Використовуючи асоціацію, що віртуальна машина може діяти як руткіт, ми пропонуємо нову асоціацію: віртуальна машина може діяти як «Людина» в атаці типу MITM між хостом та вкладеним гостьовим користувачем. При співвіднесенні концепцій MITM з віртуальною машиною корисно проілюструвати, що невід'ємною функцією віртуальної машини є передача повідомлень між парою гість-хост. Тоді стає зрозуміло, що віртуальна машина, яка представляє руткіт – «Людина» – може легко та послідовно підслуховувати зв'язок між головною машиноюі вкладеним гостьовим користувачем віртуальної машини. Також цінно проілюструвати, що руткіт на основі вкладеної віртуальної машини природним чином досягає ще однієї головної мети для руткіту, а саме того, що він може залишатися непоміченим протягом тривалого часу, просто емулюючи віртуальне середовище для вкладеної гостьової системи. У цьому типі руткіту віртуальна машина на стороні джерела також може намагатися приховати будь-яку поведінку, пов'язану з її діяльністю з підслуховування,

шляхом модифікації внутрішнього програмного забезпечення та/або дані віртуальної машини. Рисунок 1.10 – це абстракція, яка зображує типове налаштування програмного/апаратного забезпечення віртуалізації для мережі та потоку зв'язку віртуалізації між парою QEMU/KVM гість-хост. На рисунку пунктирні чорні стрілки представляють потік передачі даних мережевих пакетів гостьового користувача.

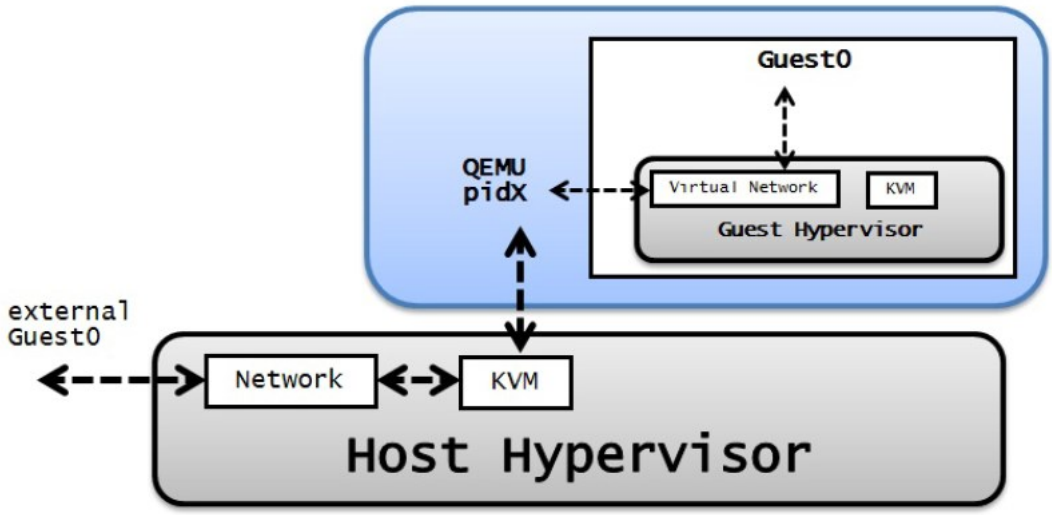

Рисунок 1.10 – Архітектура віртуалізації KVM/QEMU

Додавши наші дві асоціації моделі потоків: 1. віртуальна машина може діяти як руткіт, та 2. віртуальна машина може діяти як «Людина» у вкладеній атаці MITM на основі віртуальних машин, ми можемо створити візуальне представлення цієї концептуальної моделі. На рисунку 1.11 зображено віртуальну машину на стороні хоста (QEMU pidX), яка містить вкладену гостьову віртуальну машину (QEMU pidY), тому pidX може представляти руткіт. Руткіт контролюється Guest0, а вкладений гостьовий користувач контролюється Guest1, обидва підключені до хоста через зовнішнє мережеве з'єднання. Якби Guest0 був зловмисним користувачем, Guest0 міг би пасивно або активно атакувати двох жертв у атаці типу MITM: головну машину та Guest1. Знову ж таки, враховуючи цю вкладену архітектуру віртуалізації, Guest0 може представляти зловмисного «Чоловіка» посередині між хостом та Guest1.

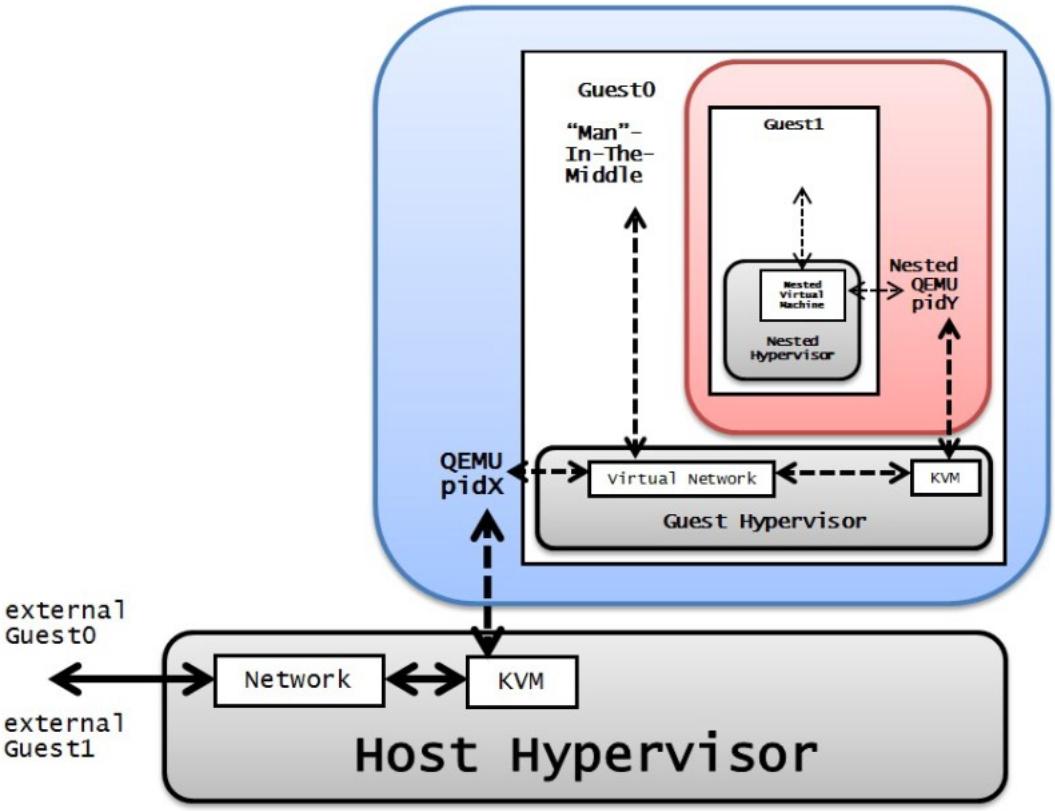

Рисунок 1.11 – Вкладена архітектура віртуалізації KVM/QEMU

Повні деталі проектування та реалізації цього типу атаки будуть наведені нижче, оскільки це лише вступ до концептуальних основ системи.

1.6. Модель загрози

Зазвичай, щойно зловмисники отримують контроль над комп'ютерною системою, вони намагаються або приховати докази вторгнення, або намагаються зберегти цей контроль якомога довше, одночасно виконуючи якусь шкідливу послугу. У роботі ми робимо переконливий висновок припущення, що зловмисники вже скомпрометували хмарну систему жертви, і що наш руткіт забезпечує приховане середовище, в якому зловмисник може непомітно зберігати контроль над власником гостьової віртуальної машини жертви, доки власник гостьової віртуальної машини не відключиться від хмари.

Щоб було зрозуміло, хоча компрометація хмарної системи є прагматично складною, вона можлива. Отримання привілеїв хоста можна досягти шляхом

проникнення в хмарну мережу та використання певних вразливостей хост-машини/ОС, або, як ми раніше наводили докази в розділі 2.2.2, злом віртуальних машин та інші методи були продемонстровані як у реальних атаках, так і в дослідницькій спільноті з безпеки програмного забезпечення [13, 11, 50, 23, 34, 49]. Рівень привілеїв хоста, необхідний після компрометації хмарної системи, буде визначено детально в розділі 2.2. Згідно з нашим сильним припущенням, зловмисники можуть створювати власні віртуальні машини, ініціювати міграцію віртуальних машин у реальному часі, а потім створювати вкладені віртуальні машини всередині своїх віртуальних машин.

Якщо метою зловмисника є утримання контролю над скомпрометованою системою протягом тривалого часу, йому потрібно приховати себе та непомітно отримати доступ до системних ресурсів. З цією метою зловмисники зазвичай встановлюють руткіти на скомпрометовану систему. Ці руткіти можуть допомогти зловмисникам приховати свої шкідливі процеси, шкідливі програми.файли та шкідливі сокетні з'єднання. Наш унікальний тип руткіту не прагне контролювати або підтримувати системні ресурси хмарної платформи, натомість наша мета – отримати контроль над віртуалізованою машиною та службами її власника-жертви-гостьової віртуальної машини, що працює в хмарі.

На основі цієї моделі загроз ми опишемо, як зловмисники можуть використовувати віртуальну машину в реальному часі.міграція та вкладена віртуалізація для створення та встановлення руткіту CloudSkulk у розділі 2. Знову ж таки, для уточнення, наш руткіт дозволяє зловмиснику активно або пасивно перехоплювати, підслуховувати та контролювати весь обмін даними QEMU/KVM між гостьовою віртуальною машиною жертви та її хмарною платформою. Усі живі взаємодії власника гостьової віртуальної машини жертви будуть видимі для зловмисника, який встановив руткіт CloudSkulk у певному цільовому хмарному середовищі на основі IaaS.

РОЗДІЛ 2. МЕТОДОЛОГІЯ ВПРОВАДЖЕННЯ РУТКІТУ CLOUDSKULK У ХМАРНОМУ СЕРЕДОВИЩІ НА БАЗІ QEMU/KVM

2.1. Розробка системи руткіту

Дизайн CloudSkulk базується на віртуальній машині (KVM) на базі ядра Linux гіпервізора. У системі Linux гіпервізор KVM реалізовано як два модулі ядра: один незалежний від архітектури (наприклад, kvm.ko), а інший залежний від архітектури (наприклад, kvm-intel.ko або kvm-amd.ko). KVM використовує підтримку на апаратному рівні, що знаходиться в сучасних розширеннях віртуалізації процесора, Intel VT та AMD-v, для віртуалізації архітектури гостьової віртуальної машини. Кожна віртуальна машина потім розглядається як звичайний процес і планується планувальником процесів Linux за замовчуванням. Для створення та запуску віртуальних машин користувачі найчастіше використовують інструмент рівня користувача під назвою Quick Emulator (QEMU). Програмне забезпечення QEMU використовує функції віртуалізації KVM для емуляції ОС немодифікованої гостьової віртуальної машини, її паравіртуалізованих та/або повністю віртуалізованих пристроїв та всіх її програм.

Руткіт, який ми представляємо, було виконано на платформі Linux, на якій розміщено QEMU/KVM. Парадигма програмного забезпечення віртуальної машини. Є два ключові атрибути, які обмежують наш дизайн: програмне забезпечення для віртуалізації, на яке спрямований наш руткіт, повинно (1) забезпечувати утиліту для міграції в реальному часі та (2) забезпечувати вкладені гіпервізори – QEMU/KVM відповідає цим вимогам.

Концептуально запропонований нами руткіт можна встановити на інші хмарні платформи, які забезпечують ці ж дві атрибути, але ми обрали QEMU/KVM саме через його популярність та гнучкість впровадження (тобто, QEMU має відкритий вихідний код).

Руткіт CloudSkulk має бути впроваджений у хмарному середовищі, яке забезпечує динамічну міграцію. Динамічна міграція є поширеною діяльністю в хмарі; вона використовується для забезпеченнямайже безперебійне обслуговування користувача, тоді як фактичні послуги хостингу сервера для цього користувача можуть змінюватися. Ця діяльність є фундаментальною для хмарних платформ для підтримки балансу навантаження між серверами або у випадках, коли сервер потребує простою для оновлень або фізичного обслуговування. Жива міграція не завжди передбачає перенесення гостьового користувача між незалежними машинами, вона також використовується для динамічного налаштування віртуалізованих пристроїв та конфігурацій віртуальних машин живого користувача під час перенесення гостьової системи поверх того самого гіпервізора. Наш руткіт розроблений для виклику живої міграції в межах одного гіпервізора під час його встановлення; ми називатимемо це методом живої міграції віртуальної машини.

Руткіт CloudSkulk має бути реалізований у віртуалізованому середовищі, яке дозволяє користувачам створювати вкладені віртуальні машини, кожна з яких має власний гіпервізор. Цей тип віртуальної машини найчастіше надається, але не обмежується ними, хмарними платформами на основі IaaS. Постачальники послуг IaaS пропонують користувачам комплексні обчислювальні ресурси, включаючи програми для простору користувача, апаратне забезпечення та ОС, віртуалізовані та доступні через хмару. Дизайн нашого унікального типу руткіту вимагає такого віртуалізованого середовища, оскільки воно дозволяє нам (зловмисному гостьовому користувачеві в хмарі) видавати себе за хост, запускаючи той самий гіпервізор KVM та емулятор віртуальної машини QEMU, що й хост. У цьому сценарії ми можемо імітувати хост для наших власних вкладених гостьових віртуальних машин; ми можемо називати цей сценарій нашою технікою вкладеної віртуалізації. Наш руткіт називається атакою типу Rootkit-in-the-Middle (RITM) на основі вкладених віртуальних машин, оскільки ми можемо передавати зв'язок між початковим гіпервізором хоста та нашим вкладеним гостьовим гіпервізором у стилі Man-in-the-Middle (MITM).

Встановлення нашого унікального типу руткіту базується на комплексному наборі процедури. Наступні процедури визначають розробку методології нашої атаки:

Крок 1. Зазвичай у хмарному середовищі зловмисник, як і звичайні хмарні клієнти, може орендувати віртуальну машину в хмарному середовищі. На одній хост-машині, що й віртуальна машина зловмисника, може співіснувати багато віртуальних машин, і одна з них буде ціллю атаки. На рисунку 2.1 ми вважаємо GuestM віртуальною машиною, що належить зловмиснику, а Guest0 – цільовою віртуальною машиною.

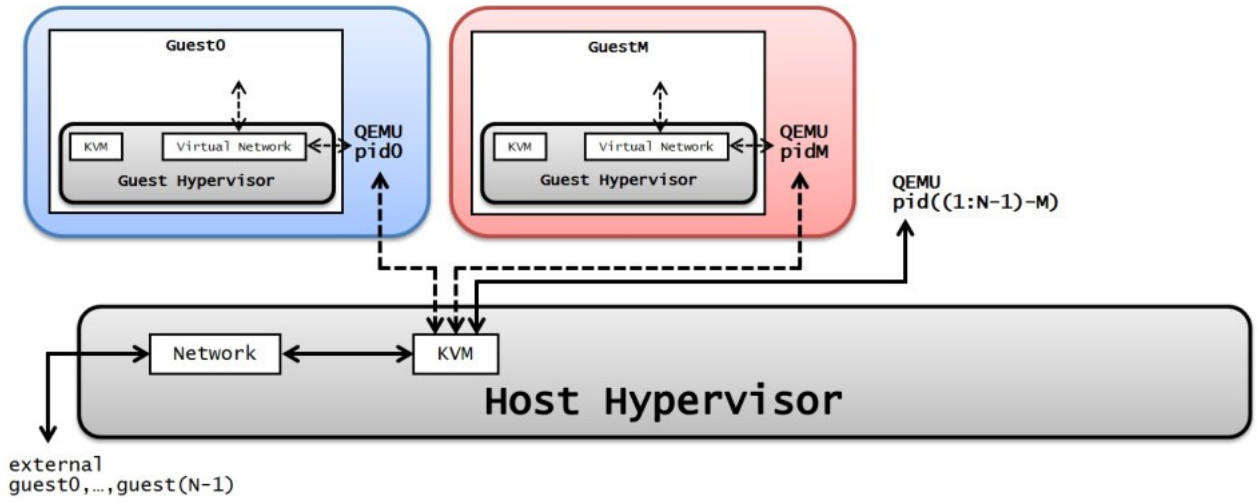

Рисунок 2.1 – Ідентифікація пари "гість-хост" у хмарному середовищі.

Точне визначення цього кроку виглядає наступним чином: на гіпервізорі хоста на базі GNU/Linux існує множина S, що складається з N пар гостевих-хостів QEMU/KVM, де $|S| = N \geq 2$. У цьому сценарії ми позначимо одну ціль нашої атаки як Guest0(t), де $t \in S$, та одного зловмисника як GuestM(m), де $m \in S$.

Крок 2. Ми припускаємо, що, скориставшись існуючими вразливостями гіпервізора, зловмисник може вирватися зі своєї віртуальної машини та отримати певний контроль над хостом. Це можливо насправді, як було продемонстровано в попередніх дослідженнях [13, 23]. Зауважте, що зловмиснику не обов'язково потрібні права системного адміністратора на хості, оскільки процес QEMU може

бути запущений будь-яким звичайним користувачем у системі Linux. На рисунку 2.2 показано червону пунктирну стрілку, яка є просто абстрактним представленням отримання привілеїв хоста.

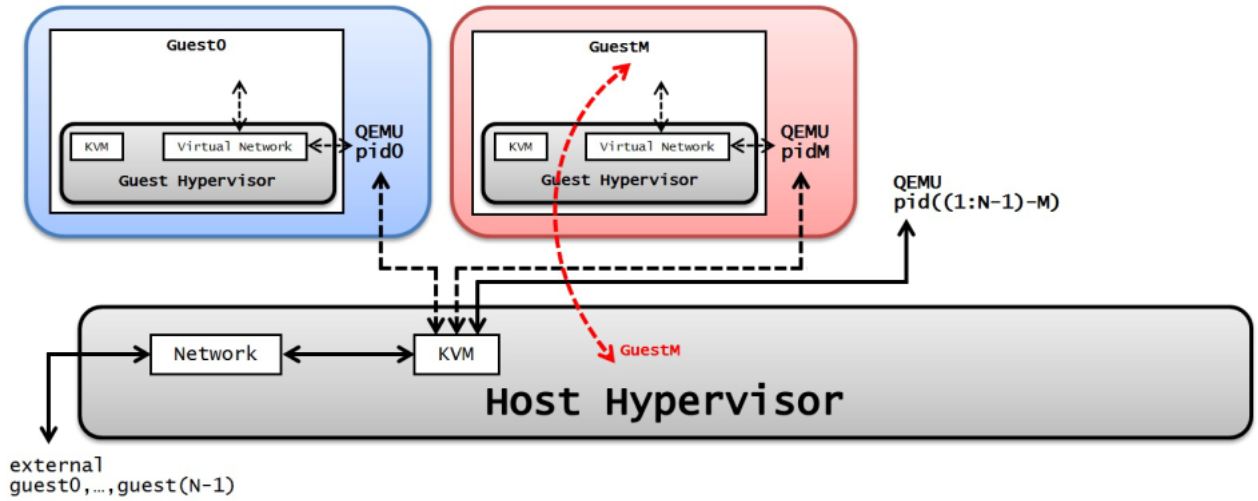

Рисунок 2.2 – Абстракція розділення віртуальної машини

Крок 3. Щойно зловмисник отримає певний контроль над хостом, він може запустити нову віртуальну машину GuestX. Для хоста ця віртуальна машина відображатиметься як місце призначення живої міграції Guest0, оскільки вона буде створена з усіма оригінальними конфігураціями QEMU Guest0. GuestX функціонально представлятиме нашу RITM.

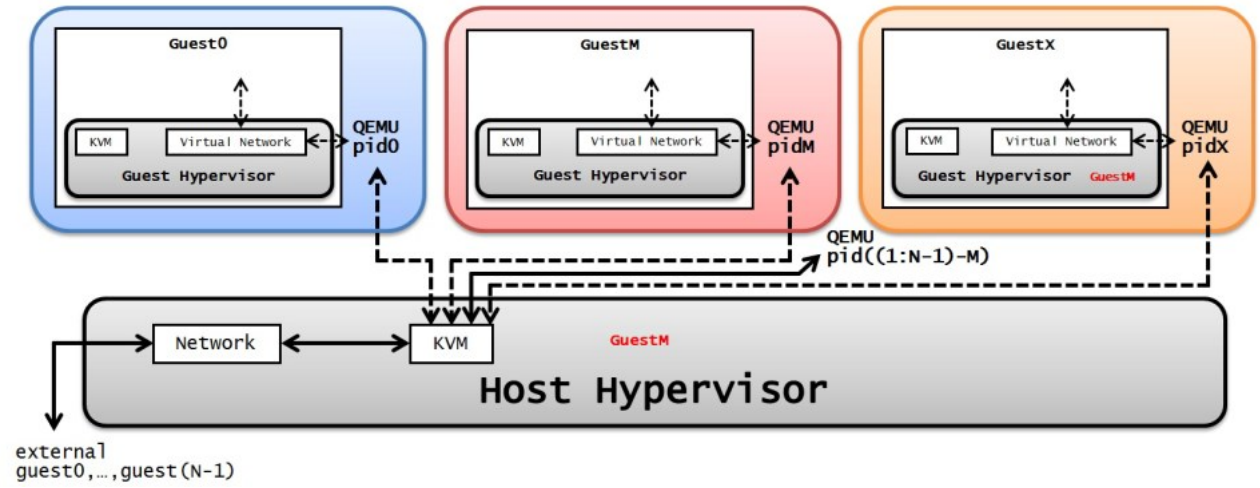

Рисунок 2.3 – Створення руткіту на хості

Точне визначення цього кроку виглядає наступним чином: Зловмисник, GuestM, повиненотримати достатні привілеї хоста, щоб можна було отримати повний набір конфігурацій Guest0 QEMU C03. Потім зловмисник викликає команди на хості для створення нової пари гость-хост GuestX(x) з конфігураціями C1, такими що $C0 \subseteq C1$ та де $x \cup S$, отже $|S| = N + 1 \geq 2$ та $x \in S$ є істинними після цієї події. Якщо C0 не був ініціалізований явним підключенням порту, скажімо, параметром c', під час його початкового створення, то $c' \cup C1$ має відбутися на цьому кроці для подальшої міграції в реальному часі. Важливо зазначити, що хоча мінімальне обмеження $C0 \subseteq C1$ має бути істинним, руткіт CloudSkulk може підтримувати найвищий рівень ухилення від хоста, якщо $C0 \subseteq C1$ та $C1 - C0 = \{c'\}$.

Крок 4. Використовуючи метод вкладеної віртуалізації, зловмисник може запустити віртуальну машину всередині GuestX. Ця вкладена віртуальна машина буде створена з усіма оригінальними конфігураціями QEMU Guest0.

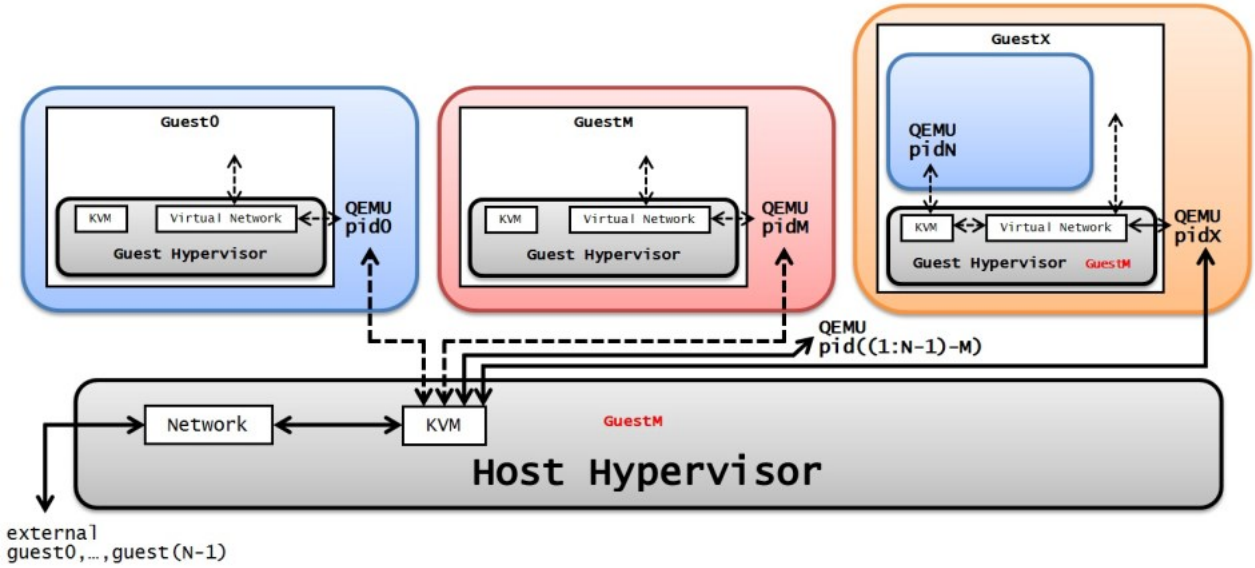

Рисунок 2.4 – Створення вкладених віртуальних машин в Rootkit

Точне визначення цього кроку виглядає наступним чином: Зловмисник, GuestM, викликає команди в GuestX для створення нового процесу QEMU4, pidN, який ініціалізується в призупиненому стані, щоб не містив активних користувачів. Повний набір конфігурацій QEMU pidN C2 має бути

ініціалізований таким чином, щоб C0 ⊆ C2 та C2 − C0 = {c”}, де c” – це додана конфігурація QEMU, яка дозволяє pidN постійно прослуховувати дані міграції через певний заданий параметр QEMU5. Новий процес, pidN, повністю інкапсульований в GuestX і невидимий для хоста; тому |S| не змінюється після цієї події.

Крок 5. Використовуючи техніку міграції віртуальної машини в реальному часі, зловмисник може перенести цільову віртуальну машину (Guest0) до вкладеної віртуальної машини.

Точне визначення цього кроку виглядає наступним чином: Зловмисник, GuestM, викликає команди консолі моніторингу QEMU на хості host6, щоб розпочати міграцію в реальному часі. Зловмисник може вибрати міграцію активної пам'яті або диска Guest0, порт яких при цьому перенаправляється через з'єднання активного порту GuestX-хоста c'. Пункт призначення цих даних міграції в реальному часі негайно надсилається до вкладеного призупиненого процесу QEMU, pidN. Після цього кроку цільова віртуальна машина, Guest0, буде працювати всередині GuestX як вкладена віртуальна машина. GuestX, наш RITM, тепер служить засобом, який може підслуховувати активність Guest0 та зв'язок між гіпервізором хоста та гіпервізором Guest0. На рисунку 2.5 зображено цю подію, де жирна червона лінія позначає міграцію в реальному часі Guest0 з pid0 хоста до pidN GuestX.

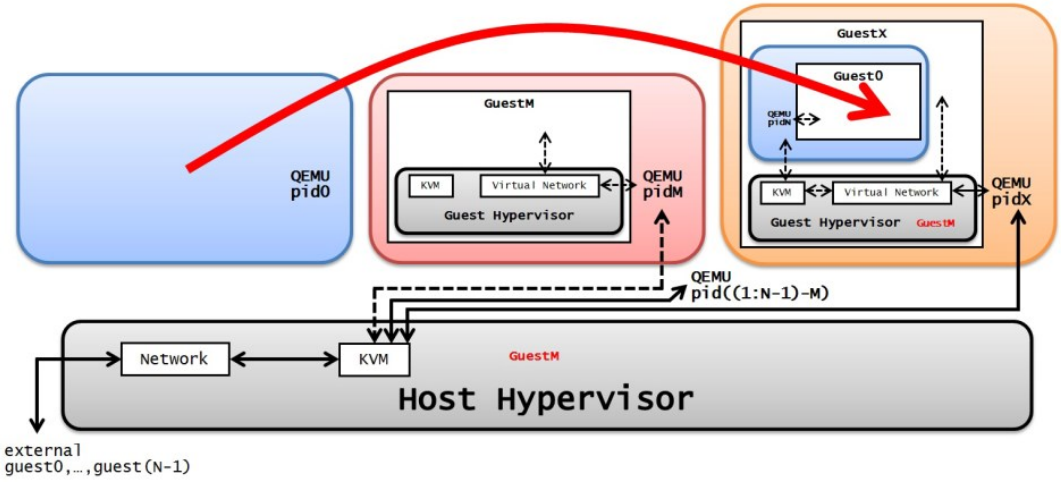

Рисунок 2.5 – Міграція віртуальної машини в реальному часі на вкладену віртуальну машину

На цей момент процес pid0 (джерело міграції) залишатиметься на хості, але в призупиненому стані після міграції, без активного користувача. Це типово для живої міграції в хмарі, оскільки для завершення застарілого процесу потрібен останній крок очищення. Тому після цієї останньої події до хоста застосовуватиметься початкова умова $|S| = N \geq 2$. Для нашої інсталяції, після того, як зловмисник видалив pid0, на хості буде встановлено вкладену RITM на базі віртуальної машини під назвою CloudSkulk.

2.2. Переваги CloudSkulk

Головна перевага руткіту CloudSkulk полягає в його прихованості. Як власнику віртуальної машини (тобто жертві), так і адміністратору хмарної системи важко виявити існування такого руткіту.

З точки зору власника віртуальної машини, майже всі поведінкові та віртуалізовані характеристики з гостьового середовища залишаються незмінними. Є кілька причин, але спочатку давайте порівняємо нашу інсталяцію руткіту з типовою міграцією в реальному часі в хмарі, щоб зрозуміти рівень прихованості, що забезпечується CloudSkulk. Якби звичайний власник віртуальної машини, що працює в хмарі, був мігрований в реальному часі, незалежно від подій, незалежних чи залежних від гостьової віртуальної машини, власник віртуальної машини, ймовірно, не знав би про цю діяльність до, під час або після міграції. Цей вроджений, ухильний характер міграції в реальному часі є запроектованим і дозволяє безперебійне обслуговування власника віртуальної машини – ключовий атрибут хмари. Машина, пристрої, ОС та програми гостьової віртуальної машини віртуалізовані до та після міграції, тому різні методи виявлення віртуалізації не можуть бути застосовані в цьому сценарії. Для CloudSkulk, під час запуску Guest0 та GuestX, зловмисник використовує переадресацію портів, щоб жертва також продовжувала отримувати доступ до своєї віртуальної машини, використовуючи ту саму команду, що й раніше. Власник віртуальної машини не спостерігає жодних очевидних змін, окрім

продуктивності. Власник віртуальної машини відчує зміну продуктивності зміна через додатковий рівень віртуалізації.

З точки зору системного адміністратора, GuestX тепер вважатиметься Guest0. Зловмисник може переконатися, що GuestX та Guest0 використовують однакові віртуалізовані пристрої, однакову гостьову ОС та запускають однакові програми; тим часом, маючи повний контроль всередині GuestX, зловмисник має достатні можливості для маніпулювання різними методами інтроспекції віртуальної машини (VMI). Це було добре задокументовано та вивчено раніше [5]. Інструменти VMI зазвичай покладаються на деякі попередні знання про цільову ОС, зокрема знання на рівні ядра, але коли зловмисники контролюють ядро гостьової системи, маніпулюючи різними структурами даних ядра, зловмисники можуть підірвати існуючі інструменти VMI. Наслідком підриву інструментів VMI є те, що зловмисники зможуть приховати свою діяльність або будь-яку аномалію всередині гостьової ОС.

2.3. Впровадження

Хмарні середовища є дуже динамічними. Характеристики потенційної цільової віртуальної машини можуть бути налаштовані за допомогою різноманітних функцій образу QEMU та емуляції системи. Крім того, певні параметри віртуальної машини є критично важливими для безпеки віртуальної машини в хмарних середовищах.невідомі громадськості. З цих причин зловмисник, який встановлює руткіт CloudSkulk, повинен використовувати деякі утиліти історії системного рівня та/або інструменти перевірки віртуальних машин, щоб викрити цільові конфігурації віртуальної машини.

Ми вже згадували раніше, що зловмиснику, який намагається встановити руткіт CloudSkulk, потрібен певний «певний» контроль над хостом. Для уточнення, рівень необхідного контролю обмежується найвищим рівнем привілеїв серед: інспекція інструменти, необхідні для отримання параметрів конфігурації цільової віртуальної машини, міграції в реальному часі та ініціалізації цільової віртуальної машини з відповідними конфігураціями мережі

як вихідної віртуальної машини. Рівень контролю хоста потім змінюватиметься залежно від цих обмежень.

Найпростішим рішенням для пошуку конфігурацій цільової віртуальної машини було бдослідити історію командного рядка ([host@cloud~]$ history) або повідомити про запущеністатуси процесів ([host@cloud~]$ ps -ef) для визначення оригінальної команди QEMUвикористовується. Якщо з якоїсь причини ці утиліти системного рівня недоступні на хості, можна використовувати один потужний інструмент для користувацького простору – QEMU Monitor. QEMU Monitor реалізовано разом із вихідним кодом QEMU та завантажується разом із QEMU за замовчуванням. Тому можна зробити слабке припущення, що ця утиліта для користувацького простору є поширеним та життєздатним рішенням для хмарних платформ, що розміщують QEMU/KVM. Наприклад, зловмисник може видати команду QEMU Monitor на запущеній цільовій віртуальній машині, щоб визначити, які блокові пристрої емулюються QEMU ([qemu-monitor~]$ info qtree, [qemu-monitor~]$ info blockstats), або визначити розмір та стан активної пам'яті віртуальної машини ([qemu-monitor~]$ info mtree, [qemu-monitor~]$ info mem), або навіть визначити тип, модель та стан мережевого пристрою з ([qemu-monitor~]$ info network). Команди QEMU Monitor також можна використовувати з іншими утилітами простору користувача, такими як qemu-img, для визначення розміру диска запущеної віртуальної машини (блок [qemu-monitor~]$ info для розташування на диску та [host@cloud~]$ qemu-img info для отримання розміру та типу файлу образу).

Як звичайний процес живої міграції в хмарі, адміністратор хмарної системи традиційно повинен зберегти вихідні параметри конфігурації для віртуальної машини користувача.під час його створення або викликати комбінацію будь-яких із перелічених вище інструментів для отримання цих параметрів під час виконання. Для динамічної міграції ця подія має відбутися, оскільки для міграції спочатку потрібно створити цільову віртуальну машину з тими ж конфігураціями як вихідну віртуальну машину.

Реалізація CloudSkulk починається з вибору цільової віртуальної машини, що працює на хості, та отримання її параметрів конфігурації QEMU. Приклад рішеннядо цього першого кроку наведено нижче:

```
[host@yellowstone ~]$ ps -eq | grep -i qemu
... ...
... qemu-system-x86_64 -machine type=pc-i440fx-2.3..
... ...
```

Потім на стороні хоста можна створити руткіт. Руткіт – це процес QEMU, який відповідає параметрам цільового QEMU. Зверніть увагу, що під час тестування цільова віртуальна машина може бути віртуалізована з більшою кількістю пристроїв та прапорців процесора, ніж вихідна віртуальна машина, успішно зберігаючи життєздатність атаки. Якщо цільова віртуальна машина спочатку не створена з конфігурацією мережі переадресації портів хоста, ми повинні або додати це за допомогою команди QEMU Monitor ([qemu-monitor~]$ hostfwd add [tcp |udp]:[hostaddr]:hostport-[guestaddr]:guestport), або ми можемо зробити це під час створення руткіту. Подальший приклад рішення цього кроку описано нижче, де ми додаємо опцію переадресації портів хоста під час створення нашого руткіту:

```
qemu-system-x86_64 \
-machine type=pc-i440fx-2.3 \
-cpu Nehalem,+vmx \
-m size=512M,slots=1,maxmem=1024M \
-boot order=c \
-drive file=disk-RAW60G.img,media=disk,format=raw,
   cache=writeback,aio=threads,if=virtio \

-enable-kvm \
-vga std \
-show-cursor \
-device virtio-net-pci,netdev=net0 \
-netdev user,id=net0,hostfwd=tcp:0:4446-:5556 &
```

Гостьове середовище віртуальної машини в руткіті має власний гіпервізор.та можливість вкладення віртуальних машин/гіпервізорів. Таким чином, ОС в руткіті можна модифікувати, якщо потрібно, щоб увімкнути програмну інфраструктуру QEMU/KVM. Нижче наведено покрокову інформацію про це, але це не вичерпне рішення, а радше життєздатне рішення, яке ми перевірили та використовували під час впровадження:

1. Перевірте поточну ОС та архітектуру GNU/Linux x86 64 bit.

1a. Якщо ні, завантажте, встановіть та перезавантажте Fedora Live Workstation x86 64 версії 22.3. Потім завантажте та встановіть ядро Linux версії 4.4 з конфігураціями за замовчуванням. Перезавантажте та перевірте, чи модуль kvm увімкнено з новою інсталяцією ([host@cloud˜]$ lsmod |grep -i kvm).

2. Перевірте, чи QEMU x86 64-біт версії 2.9 доступний на поточній машині.

2a. Якщо ні, завантажте QEMU x86 64 bit stable версії 2.9, потім налаштуйте та встановіть збірку з мінімальними такими параметрами: –target-list=x86 64-softmmu, –enable-curses, –enable-kvm. Перевірте, чи ввімкнено правильну версію qemu ([host@cloud˜]$ qemu-system-x86 64-version).

3. Визначити, чи ввімкнено параметр kvm intel ([host@cloud˜]$ modinfo kvm intel |grep -i nested).

3a. Якщо ні, тимчасово вивантажте модуль kvm intel ([host@cloud˜]$ modprobe -r kvm intel). Потім змініть параметр вкладеності модулів ядра для kvm intel ([host@cloud˜]$ sudo vi /etc/modprobe.d/dist.conf, потім розкоментуйте рядок options kvm intel nested=y, збережіть зміни та вийдіть з конфігураційного файлу). Перезавантажте та перевірте чи зміни ядра є стійкими ([host@cloud˜]$ cat /sys/module/kvm intel/parameters/nested).

4. Перевірте, чи ввімкнено розширення віртуалізації процесора, Intel VT |AMD-v, за допомогою type = full ([host@cloud˜]$ lscpu |grep Virtualization).

4a. Якщо ні, переконайтеся, що оригінальна команда QEMU додає розширення процесора гостьової системи під часствоpення ([host@cloud˜]$ qemu... -модель процесора,+vmx |+amd).

Після ввімкнення програмної інфраструктури QEMU/KVM у гостьовій системі наступним кроком у впровадженні руткіту CloudSkulk є створення

вкладеної віртуальної машини. Вкладена віртуальна машина – це віртуальна машина призначення для живої міграції з параметрами конфігурації QEMU, що відповідають цільовій віртуальній машині на стороні хоста. Однією з вимог живої міграції є додавання параметрів віртуальної машини призначення таким чином, щоб вона була призупинена у вхідному стані, і тому вона прослуховуватиме дані міграції через певний заданий параметр. Як ми вже згадували раніше, повний список можливих параметрів живої міграції буде детально описано в цьому розділі; нижче наведено цей повний список, обмежений параметрами системного емулятора QEMU PC:

```
-incoming tcp:[host]:port[,to=maxport][,ipv4][,ipv6]
-incoming rdma:host:port[,ipv4][,ipv6]
   Accept incoming migration using host tcp port.

-incoming unix:socketpath
   Accept migration using host unix socket.
-incoming fd:fd
   Accept migration using host file descriptor.
-incoming exec:cmdline
   Accept migration using output specified from external command.
-incoming defer
   Accept migration using a later specified URI via QEMU monitor
   command migration_incoming.
-device ivshmem-plan,memdev=mySharedMem
   Accept migration using shared memory backend file on host.
```

Створення вкладеної віртуальної машини з використанням одного з вищезазначених життєздатних параметрів міграції QEMU відбувається наступним чином:

```
qemu-system-x86_64 \
-machine type=pc-i440fx-2.3 \
-cpu Nehalem,+vmx \
-m size=512M,slots=1,maxmem=1024M \
-boot order=c \
-drive file=copy-RAW60G.img,media=disk,format=raw,
   cache=writeback,aio=threads,if=virtio \
-enable-kvm \
-vga std \
-show-cursor \
-device virtio-net-pci,netdev=net0 \
-netdev user,id=net0 \
-incoming tcp:0:5556 &
```

За винятком незначного очищення, останнім кроком у впровадженні нашого руткіту є виклик утиліти живої міграції через монітор QEMU. Залежно від того, як монітор цільової віртуальної машини емульовано на стороні хоста, її монітор QEMU можна відкрити кількома способами. Наприклад, якщо монітор QEMU цільової віртуальної машини мультиплексовано на інший послідовний порт, такий як telnet-сервер, який прослуховує порт 5555 ([host@cloud~]$ qemu-... -serial mon:telnet:0:5555,server,nowait), тоді telnet на стороні хоста можна викликати для відкриття монітора QEMU віртуальної машини ([host@cloud~]$ telnet 0 5555). У нашому подальшому прикладі, оскільки віртуалізовану відеокарту для віртуальної машини було створено (-vga std), ми відкрили монітор QEMU для цільової віртуальної машини простим натисканням клавіш Ctrl+Alt+Shift+2 після того, як курсор хоста вибрав цільову віртуальну машину. Після відкриття монітора QEMU можна виконати таку команду в середовищі монітора, щоб запустити міграцію в реальному часі:

```
migrate -d tcp:0:4446
```

Номери портів, які ми вибрали в нашому прикладі, є випадковими. Однак, зв'язок номерів портів відносно руткіту, вкладеної віртуальної машини та

міграції в реальному часіКоманда є критично важливою для нашої реалізації. Цільова віртуальна машина починає транзакцію, надсилаючи дані міграції на HOST PORT AAAA. Руткіт був створений на стороні хоста таким чином, що він продовжує транзакцію, пересилаючи HOST PORT AAAA на свій внутрішній ROOTKIT PORT BBBB. Зрештою, призупинена вкладена віртуальна машина завершить транзакцію, отримавши дані міграції від ROOTKIT PORT BBBB. Ця транзакція візуально представлена на рисунку 2.6.

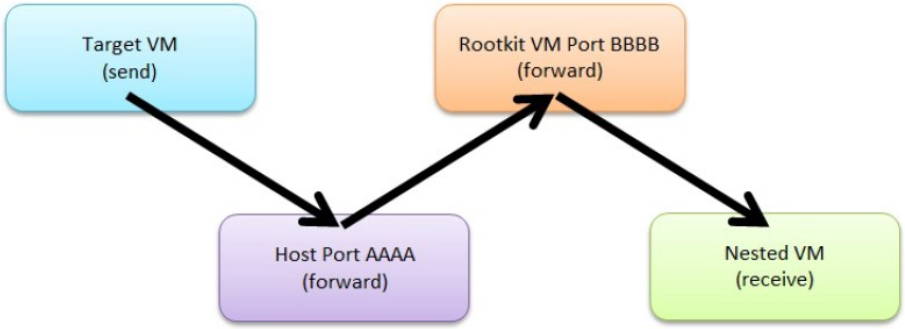

Рисунок 2.6 – Метод вкладеної віртуалізації, транзакція порту

Після завершення міграції в реальному часі потрібне незначне очищення. Це можна зробити, завершивши процес Linux, відповідальний за призупинену цільову віртуальну машину після міграції, на стороні хоста, або просто за допомогою команди QEMU Monitor, яка все ще відкрита на віртуальній машині після міграції ([qemu-monitor˜]$ quit) або на стороні хоста ([host@cloud˜]$ kill -KILL ⟨process id⟩). Цей крок завершує реалізацію руткіту CloudSkulk.

РОЗДІЛ 3. ВІРТУАЛІЗОВАНА АТАКА ТИПУ ROOTKIT-IN-THE-MIDDLE: МЕТОД РОЗГОРТАННЯ CLOUDSKULK

3.1 Перевірка продуктивності руткіту

Немає фіксованого порогу, який би визначав рівні непоміченості, а також немає порогу, який би визначав максимальну тривалість часу встановлення, якого ми повинні досягти, щоб залишитися непоміченим. Тому замість цього ми зосереджуємося на кількісному вираженні здатності нашого руткіту залишатися непоміченим для цільового користувача гостьової віртуальної машини та хост-хмарної платформи, на яку спрямована наша атака. Ми досягаємо цього, характеризуючи як зниження продуктивності, спричинене нашим унікальним типом RITM, так і час його встановлення. Наша мета полягає в тому, щоб ці дані можна було застосовувати в кожному окремому випадку до конкретних віртуалізованих середовищ для оцінки достовірності нашого руткіту.

Усі експерименти проводяться на тестовому стенді під керуванням операційної системи Debian з ядром GNU/Linux версії 4.4.14 (-200.fc22.x86 64), що містить KVM. Гість Рівень 1, Рівень 0 та Рівень 2 також використовують ту саму версію робочої станції з ядром Linux 4.4.14. Усі середовища виконання: Рівень 0, Рівень 1 та Рівень 2 використовують останню стабільну версію QEMU 2.9.50 (v2.9.0-989-g43771d5) з такими параметрами конфігурації встановлення: –enable-kvm, –enable-curses. Наша тестова платформа використовує Dell Precision T1700 з процесором Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz. Хост має 16 ГБ пам'яті, і кожній віртуальній машині ми виділяємо 1 ГБ пам'яті.

Зверніть увагу, що QEMU пропонує безліч параметрів для своїх віртуалізованих машин, налаштовуючиРізні значення цих параметрів можуть суттєво вплинути на продуктивність робочого навантаження у віртуальній машині. Під час наших експериментів ми дотримувалися найкращих практик QEMU/KVM, описаних у [17]. Зокрема, наша конфігурація має такі атрибути:

- Ми використовували драйвери Virtio. Virtio спочатку був платформою віртуалізації вводу/виводу для Linux, але тепер її також застосували

у Windows. По суті, Virtio охоплює ідею паравіртуалізації. З одного боку, гостьова ОС знає про своє середовище віртуалізації та включає драйвери, що діють як інтерфейс (ці драйвери потрібно ввімкнути під час компіляції ядра гостьової системи); з іншого боку, гіпервізор реалізує драйвери сервера, основними завданнями яких є емуляція певних пристроїв. У QEMU, коли ми запускаємо віртуальну машину, нам потрібно додати «-drive if=virtio» та «-device virtio-net-pci» до командного рядка QEMU, щоб увімкнути драйвер сервера диска та драйвер мережевого інтерфейсу відповідно.

- Ми використовували блокові пристрої для зберігання образів віртуальних машин. Зазвичай образ віртуальної машини зберігаєтьсяу файлі на файловій системі хоста, проте віртуальній машині можна призначити пристрій або розділ диска. У наших експериментах ми помітили, що образ віртуальної машини, підкріплений блоковим пристроєм, працює значно краще, ніж віртуальна машина, підкріплена файлом. Наприклад, для компіляції ядра різниця сягала від 20% до 30%.

Крім того, ми також експериментували з різними конфігураціями QEMU. Наші кінцеві результати використовують найкращі показники, що спостерігалися з такими параметрами:

- Формат зображення гостя. Два найпоширеніші формати зображень, що використовуються в QEMU, – це формат необробленого зображення та формат зображення qcow2. Найкращим варіантом нашої роботи та остаточним вибором став формат необробленого зображення. Наші спостереження збігаються з [17], де необроблене зображення пропонує кращу продуктивність, ніж qcow2.

- Розмір пам'яті гостьової системи. Розмір віртуальної оперативної пам'яті для запуску гостьової системи виділяється на основі цього параметра. Наші спостереження для -m size=512M, 1G, 4G, 16G показали, що 1G є найкращим налаштуванням продуктивності для нашого випадку.

- Режим гостьового кешу. Три найпоширеніші режими кешу, що використовуються в QEMU:

“*cache=writeback*”, “*cache=writethrough*”, and “*cache=none*”

Перший режим, показаний на рисунку 3.1, є стандартним, який увімкнув як кеш сторінок хост-ОС, так і кеш запису на фізичний диск. Ця політика кешування повідомлятиме про завершення запису даних, щойно дані з'являться в кеші сторінок хоста.

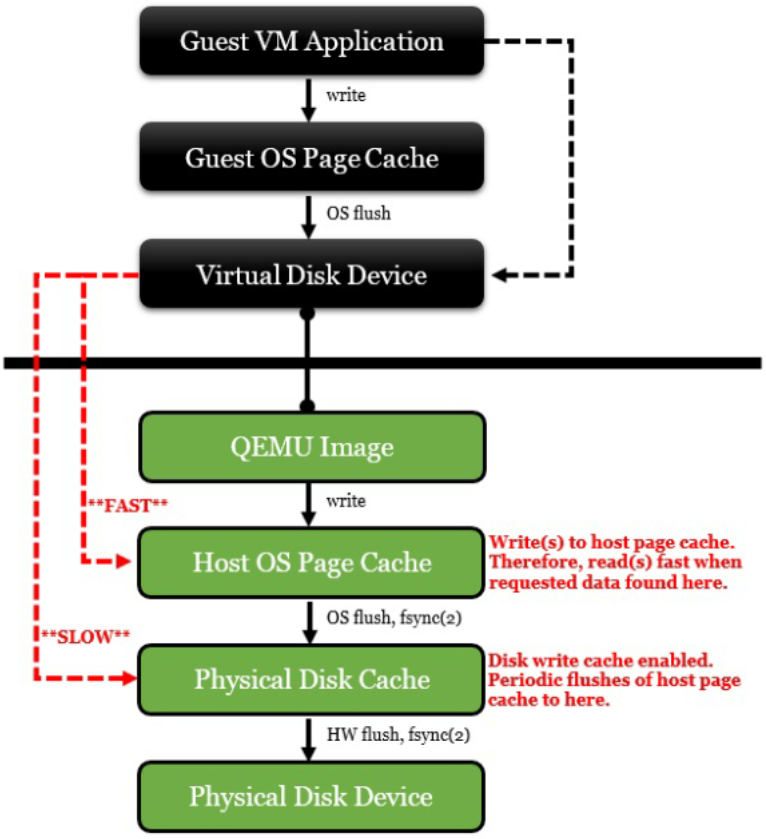

Рисунок 3.1 – Режим керування кешем хоста QEMU/KVM – «зворотний запис».

У другому режимі, показаному на рисунку 3.2, кеш сторінок хоста буде використовуватися для читання та запису даних, але кеш запису фізичного диска буде вимкнено. У цьому режимі запису сповіщення надсилатимуться гостьовій системі лише після того, як QEMU забезпечить скидання кожного запису на диск. Політика кешу наскрізного запису показала найгірші результати в наших експериментах, оскільки кожен запис суттєво впливає на продуктивність вводу-виводу.

Останній режим, показаний на рисунку 3.3, вимикає кеш сторінок хост-ОС та вмикає кеш запису на фізичний диск. У наших експериментах ця політика

кешування виконуваланайкращий, трохи кращий за «writthrough». Нашим вибором було cache=none.

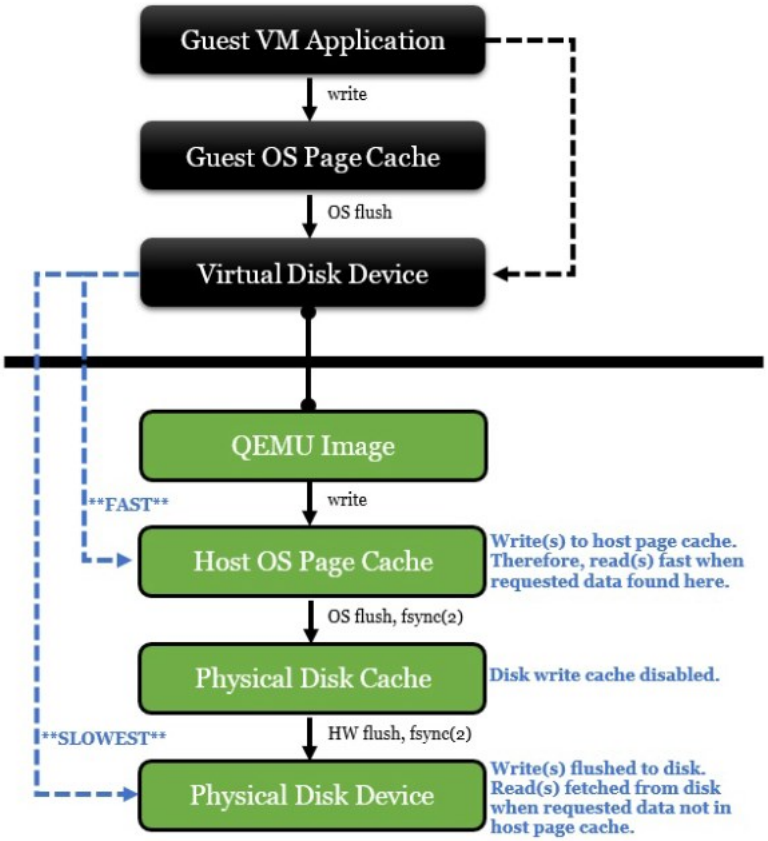

Рисунок 3.3 – Режим керування кешем хоста QEMU/KVM – «немає»

• Планувальник вводу/виводу. Linux пропонує чотири різні планувальники вводу/виводу: планувальник noop, планувальник deadline, планувальник передбачливого виконання та повністю планувальник справедливого виконання.планувальник черг (CFQ). У наших експериментах ми обрали планувальник вводу-виводу за замовчуванням, планувальник CFQ. Було запропоновано [17], що використання планувальника дедлайнів як на хості, так і на гостьовій ОС для робочих навантажень, пов'язаних з вводом-виводом, має пропонують кращу продуктивність, але ми не помітили значного покращення продуктивностіпід час наших експериментів, коли ми перейшли з планувальника CFQ на планувальник дедлайнів.

3.2. Макро-тести

Продуктивність та час міграції в реальному часі віртуальної машини, що працює в хмарі залежить від різноманітного набору змінних, деякі з яких є взаємозалежними. Тому недоцільно вичерпно характеризувати їх, тому ми дотримуємося найкращих практик QEMU/KVM, описаних у [17]. Для нашої стратегії тестування ми вирішили припустити звичайного хмарного гостьового користувача, дотримуючись вищезазначених найкращих практик, лише з одним набором параметрів конфігурації QEMU. Ми також знаємо, що робоче навантаження гостьового користувача в його середовищі є однією з таких змінних, яка відіграватиме значну роль у продуктивності та часі міграції. Тому, використовуючи припущеного хмарного гостьового користувача зі статичними параметрами конфігурації, ми можемо охарактеризувати як зниження продуктивності, так і час міграції в реальному часі, на які впливають три типи робочих навантажень, які підсумовують узагальнену діяльність, яку може виконувати користувач: робочі навантаження, що інтенсивно використовують ввод-вивод, робочі навантаження, що інтенсивно використовують процесор/пам'ять, та робочі навантаження, що інтенсивно використовують мережу.

Параметри конфігурації QEMU передбачуваного гостьового користувача хмари залишаються незмінними протягом усіх тестів наведено наступне:

Level 1 VM:

```
sudo qemu-system-x86_64 \
-name level1 \
-m size=1G,slots=1,maxmem=2G \
-boot c \
-drive file=/dev/sda6,index=0,media=disk,format=raw,
```

```
   cache=none,if=virtio \
-drive file=/dev/sda7,index=1,media=disk,format=raw,
   cache=none,if=virtio \
-cpu qemu64,+x2apic,+vmx \
-machine accel=kvm \
-smp 8 \
-curses \
-serial stdio \
-device virtio-net-pci,netdev=netLevel1 \
-netdev user,id=netLevel1,hostfwd=tcp::5022-:22
```

Level 2 VM:

```
sudo qemu-system-x86_64 \
-name level2 \
-m size=1G,slots=1,maxmem=2G \
-boot c \
-drive file=/dev/vdb,index=0,media=disk,format=raw,
   cache=none,if=virtio \
-cpu qemu64,+x2apic,+vmx \
-machine accel=kvm \
-smp 8 \
-curses \
-serial stdio \
-device virtio-net-pci,netdev=netLevel2 \
-netdev user,id=netLevel2,hostfwd=tcp::5022-:22
```

Знову ж таки, як згадувалося в попередньому розділі, ми використовували блокові пристрої для зберігання образів віртуальних машин. Для тестування це змусило нас запускати QEMU з правами суперкористувача, щоб ми могли отримати доступ до блокового пристрою /dev/sda6 хост-машини.

Для оцінки інтенсивних робочих навантажень вводу/виводу ми обрали широко використовуваний бенчмарк з відкритим кодом.знак під назвою

Filebench [1]. Filebench – це фреймворк для вимірювання файлового рівня програм, написаний на C, який містить набір попередньо визначених та налаштовуваних макросів високого рівня. Ми вибрали три типи макросів, які, на нашу думку, найбільш тісно пов'язані з навантаженнями вводу/виводу, які виконує узагальнений гостьовий користувач: навантаження файлового сервера, навантаження веб-сервера та навантаження поштового сервера.

- Макрос файлового сервера: складається з комбінації операцій createfile, writewholefile, closefile, openfile, appendfilerand, readwholefile, deletefile та statfile-ції. З усіх макросів, що використовуються, макрос файлового сервера унікально викликає операції запису.
- Макрос веб-сервера:однозначно домінують операції читання, з комбінацією операцій openfile, readwholefile, closefile та appendfile.
- Макрос поштового сервера:унікально викликає операції fsync; макрос складається з комбінації операцій deletefile, createfile, appendfilerand, fysnc, closefile, openfile та readwholefile.

Таблиця 3.1 – Параметри макросу Filebench

macro	testdir	filesize	nfiles	meandirwidth	nthreads	nfilereadinstances	iosize	meanappendsize
File Server	/home	64k	50,000	20	50	1	1m	16k
Mail Server	/home	2k	50,000	1,000,000	16	1	16k	8k
Web Server	/home	16k	50,000	20	100	1	1m	16k

Ці три макроси були налаштовані статично з параметрами, наведеними в таблиці 3.1.

Дані для наших тестів Filebench, як показано на рисунках 3.4 та 3.5, були зібрані за допомогою скрипта оболонки Bash, який знаходиться в Додатку B.1, який виконував кожен макрос Filebench п'ять разів поспіль у кожному середовищі виконання та усереднював результати. Ряд даних L0 представляє робоче навантаження Filebench, що працює на тестовій платформі, що нагадує наше цільове хмарне середовище; L1 – це середовище гостьової віртуальної машини, а L2 – це вкладене середовище віртуальної машини.

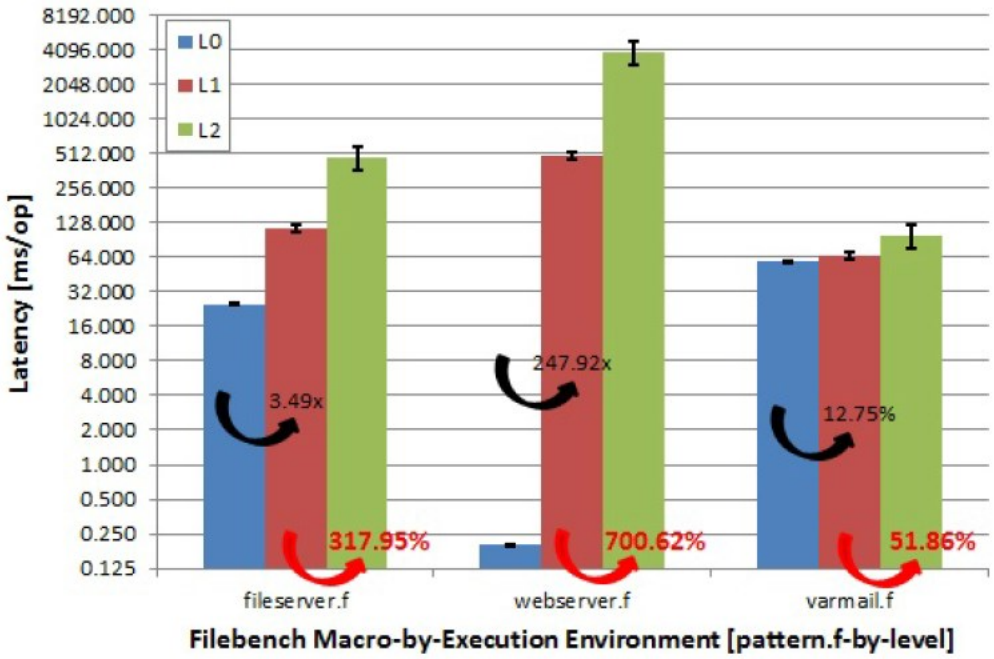

Рисунок 3.4 – Filebench – аналіз затримки інтенсивного робочого навантаження вводу/виводу

Вісь x на обох рисунках відображає три середовища виконання L0:L2, кожне з яких відповідає трьом протестованим макросам .f, для дев'яти загальних даних серії. Вісь Y на обох рисунках відображає результати Filebench у логарифмічному масштабі з основою 10. Мітки даних 1 на рисунку 3.4 показують відсоткове збільшення затримки відносно шару під ним. Мітки даних на рисунку 3.5 показують відсоткове зниження пропускної здатності відносно шару під ним. Кожен ряд даних відображає своє відносне стандартне відхилення у вигляді стовпця, центрованого у верхній частині кожного стовпця.

Оцінюючи відсоткову різницю затримки та пропускної здатності між L1 та L2 на рисунках 3.4 та 3.5, ми можемо кількісно виразити думку гостьового користувача до та після встановлення руткіту CloudSkulk для робочих навантажень з інтенсивним вводом-виводом. Після встановлення нашого руткіту цільовий гостьовий користувач спостерігається зниження швидкості на 51,86% та зниження пропускної здатності на 33,08% для робочих навантажень типу поштового сервера, приблизно в 3,18 раза нижча швидкість та зниження пропускної здатності на 99,67% для робочих навантажень типу файлового

сервера, та приблизно в 7,01 раза нижча швидкість та зниження пропускної здатності на 78,21% для робочих навантажень типу веб-сервера.

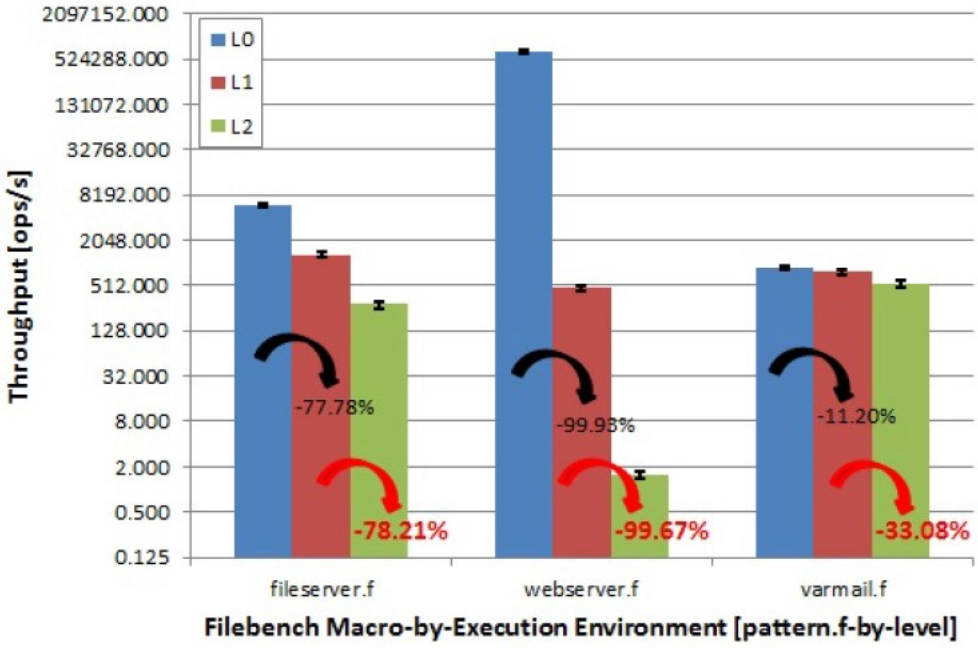

Рисунок 3.5 – Filebench – аналіз пропускної здатності ресурсомістких операцій вводу/виводу

Результати вкладених KVM, описані в літературі, показали зниження пропускної здатності приблизно на 46% у найкращому випадку та приблизно на 67% у найгіршому випадку для аналогічного інтенсивного тестування вводу-виводу Filebench [26]. Хоча наші результати зниження пропускної здатності вводу-виводу збігаються з іншими, наші результати в цілому все ще є поганими. Найголовніше, що з усіх наших тестових даних, представлених у цій статті, наші погані результати затримки вводу-виводу не корелюють з результатами літератури; тоді як аналогічні дані літератури щодо вкладених KVM показали найкращий випадок збільшення затримки вводу-виводу приблизно на 10:30% [26], наші результати показують, що гостьовий користувач хмари, скомпрометований CloudSkulk, майже напевно буде знати про значне зниження продуктивності, якщо він активно виконує навантаження, інтенсивне з вводом-виводом. Цей недолік робить поточну версію нашого руткіту схильною до систем виявлення хмарної безпеки.

Щоб оцінити наш другий фокус на робочих навантаженнях, що інтенсивно використовують процесор/пам'ять, ми вирішили зібрати дані про час декомпресії та компіляції ядра Linux для тих самих трьох завдань.середовища, як і наш попередній тест: L0, L1 та L2. За своєю природою, процес компіляції ядра є ресурсомістким для процесора та пам'яті. Знову ж таки, ми написали скрипт оболонки Bash, який можна знайти в Додатку B.2, використовуючи для всіх тестів той самий файл .config, створений на L0, та розпакували його, потім скомпілювали ядро Linux версії 4.0 п'ять разів поспіль та усереднили результати. Ці дані характеристики можна побачити на рисунках 5.6 та 5.7. Вісь x на обох рисунках відображає три ряди даних, по одному для кожного середовища виконання L0:L2. Вісь y на обох рисунках відображає результати вимірювання часу в логарифмічній шкалі з основою 10. Мітки даних на обох рисунках показують відсоткове збільшення часу відносно рівня під ним. Кожен ряд даних відображає своє відносне стандартне відхилення у вигляді смужки, центрованої у верхній частині кожного стовпця.

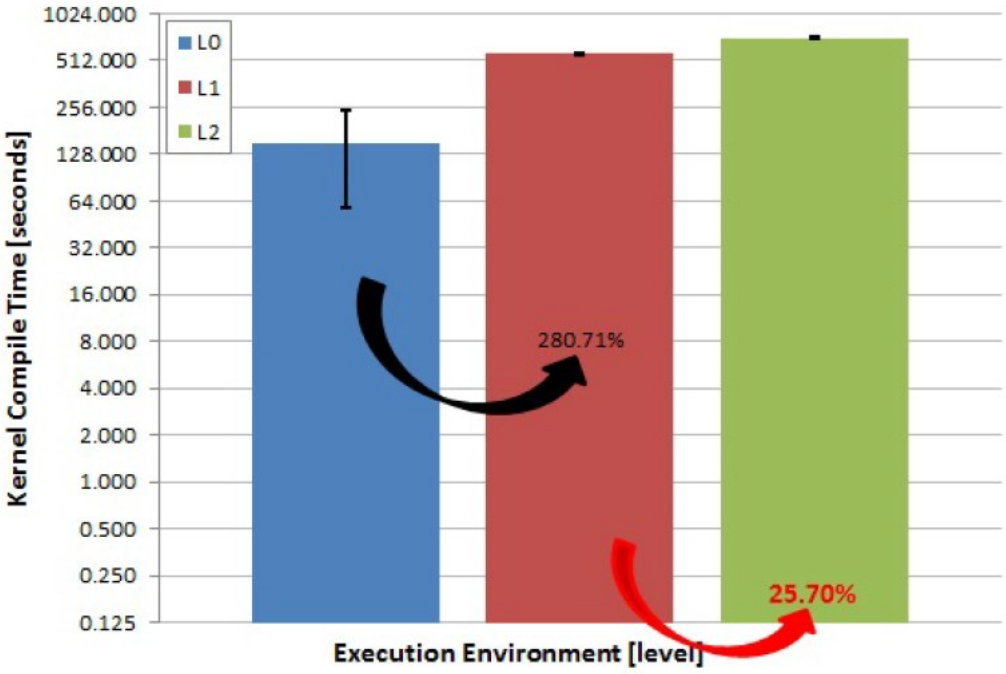

Рисунок 3.6 – Компіляція ядра – аналіз інтенсивного навантаження на процесор/пам'ять

Як раніше, ми може кількісно експрес той/та/те продуктивність деградація сприймається відгостьовий користувач для робочих навантажень, що інтенсивно використовують процесор/пам'ять, до та після встановленняруткіту CloudSkulk шляхом оцінки відсоткової різниці часу декомпресії ядра та компіляції між L1 та L2 на рисунках 3.6 та 3.7. Після встановлення нашого руткіту цільовий гостьовий користувач відчує зниження швидкості на 25,7%, пов'язане з навантаженням процесора/пам'яті типу компіляції ядра. Це збігається з результатами оцінки продуктивності KVM від Intel у 2014 році [12], які склали ~26,31% зниження від віртуалізації L1 до L2: час L2 = 14 с, час L1 = 19 с, ∴ вплив L2 = 100% ∗ (19 с − 14 с)/19 с. Ці дані були взяті з використанням тієї ж кількості vCPUS, що й у нашому тестуванні ("-smp 8") для тестування компіляції ядра. Цікаво відзначити, що відносне стандартне відхилення для часу компіляції ядра L0 було дуже високим значення 92,388% не показано на рисунку). Це сталося через те, що перший запуск був приблизно в 3 рази повільнішим, ніж останні 4 послідовні запуски. Ця точка даних компіляції ядра була повторюваною.

Рисунок 3.7 – Декомпресія ядра – аналіз інтенсивного навантаження на процесор/пам'ять

Для нашого третього фокусу на продуктивності ми вирішили використати ще один відомий, широко використовуваний бенчмарк з відкритим кодом, Netperf [19]. Netperf – це бенчмарк продуктивності мережі, написаний на C, який використовується для вимірювання продуктивності мережі на основі передачі великих обсягів даних та продуктивності запитів/відповідей з використанням мережевих протоколів TCP/UDP.Для нашого тестування ми вирішили виміряти продуктивність передачі великих обсягів даних або однонаправлений потоковий продуктивність TCP. Ми написали скрипт оболонки Bash, який можна знайти в Додатку B.3, який виконав програму Netperf п'ять разів поспіль та усереднив результати для тих самих середовищ виконання L0:L2. Наші дані бенчмарку Netperf можна побачити на рисунку 5.8. Мітки даних на цьому рисунку показують відсоткове зменшеннязатримкою відносно рівня нижче. Вісь x на цьому рисунку відображає три ряди даних, по одному для кожного середовища виконання L0:L2. Вісь y на цьому рисунку відображає результати пропускної здатності в логарифмічній шкалі з основою 10. Кожен ряд даних відображає своє відносне стандартне відхилення у вигляді стовпця, центрованого у верхній частині кожного стовпця.

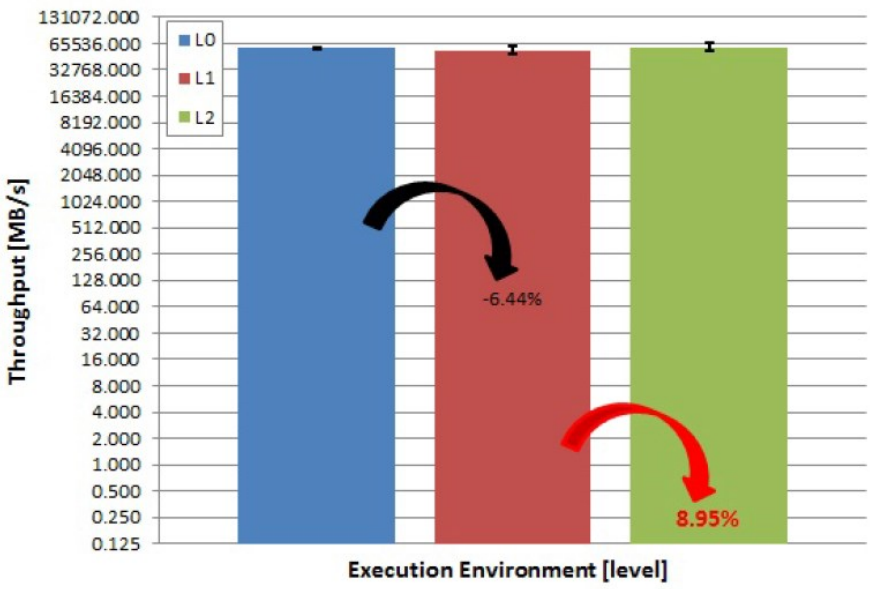

Рисунок 3.8 – Netperf – Аналіз пропускної здатності мережі з інтенсивним робочим навантаженням

Знову ж таки, відсоткова різниця пропускної здатності між L1 та L2 на рисунку 3.8 кількісно показує зниження продуктивності, яке сприймає гостьовий користувач для ресурсомістких мережевих навантажень до та після встановлення руткіту CloudSkulk. Як візуально не видно з смуг відносного стандартного відхилення, всі три рівні середовища виконання працювали майже однаково з перекриваючимися наборами даних.

Середні значення показують збільшення пропускної здатності на 8,95% для мережевих робочих навантажень типу передачі масивних даних TCP після встановлення нашого руткіту, зі стандартними відхиленнями (явні значення не показані на рисунку) для L0:L2, що становлять 1,11%, 10,32% та 3,96% відповідно. Оскільки стандартні відхилення перевищують відсоткові різниці в пропускній здатності, можна зробити висновок, що ця продуктивність була майже однаковою в усіх середовищах виконання. Ці результати демонструють, що гостьовий користувач хмари, скомпрометований CloudSkulk, не зможе виявити зниження продуктивності під час виконання ресурсомістких мережевих робочих навантажень.

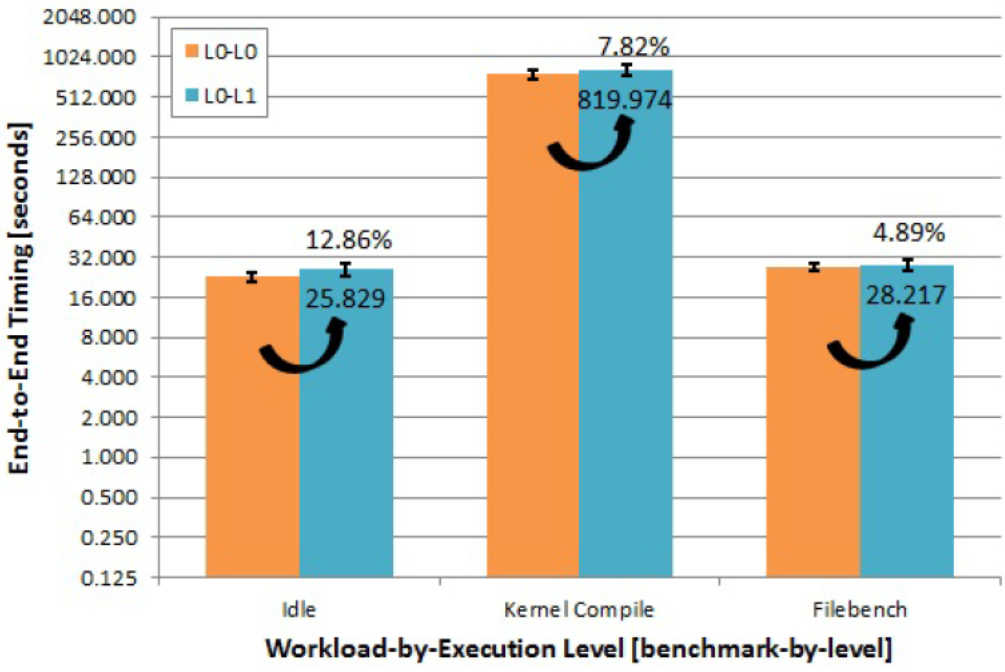

Рисунок 3.9 – Живий ефірМіграція – аналіз синхронізації від початку до кінця

Для нашої останньої оцінки ми вирішили охарактеризувати час живої міграціїгість користувач виконання різні типи з робочі навантаження: бездіяльність, Ядро компілювати, і Filebench.

Неактивне робоче навантаження представлено гостьовим користувачем, який не виконує жодного робочого навантаження.

Так, можна уявити користувача, підключеного до хмари, але далеко від свого пристрою або неактивного. Для цього тестування ми обрали два рівні живої міграції для характеристики: L0-L0 та L0-L1. Дані характеристики живої міграції можна побачити на рисунку 3.9.

Серії даних L0-L0 на рисунку 3.9 зображують типовий виклик живої міграції в хмарі, за винятком того, що під час міграції 2 не генерується мережевий трафік, оскільки як вихідна, так і цільова віртуальні машини співіснують в одному спільному середовищі. Серії даних L0-L1 на рисунку 5.9 зображують нашу унікальну техніку на основі вкладених віртуальних машин, яка використовується для реалізації CloudSkulk. Цей тип міграції передбачає живу міграцію віртуальної машини L1, що працює на стороні хоста, у вкладену віртуальну машину L2, яка інкапсульована в нашу віртуальну машину з руткітом L1 (RITM на основі віртуальної машини). Загальною метрикою для живої міграції є загальний час міграції (час від кінця до кінця). Кожна точка даних на рисунку 3.9 – це середній час від кінця до кінця для 5 послідовних запусків з відповідним відносним стандартним відхиленням, що відображається у стовпцях, по центру над кожним стовпцем.

Важливо зазначити, що конфігурації віртуальних машин L1 та L2 для тестування міграції в реальному часі дещо відрізнялися від інших даних характеристик, обговорених вище. Для узгодженості нижче детально описано різницю між попередніми параметрами та параметрами міграції в реальному часі:

Відмінності конфігурації міграції віртуальної машини в реальному часі:

```
from: -m size=1G,slots=1,maxmem=2G \
to:   -m size=512M,slots=1,maxmem=1024M \
```

```
from: -drive file=/dev/<disk>,index=0,media=disk,
        format=raw,cache=none,if=virtio \
to:   -drive file=/tmp/guest-imgs/copyRAW60G.img,
        media=disk,format=raw,cache=writeback,aio=threads,if=virtio \

from: -cpu qemu64,+x2apic,+vmx \
to:   -cpu Nehalem \

from: -machine accel=kvm \
to:   -enable-kvm \

new:  -machine type=pc-i440fx-2.3 \
```

На рисунку 3.9 показано два набори міток даних. Найнижчий набір міток даних показує числові значення, пов'язані з кожним часом від початку до кінця, а найвищий набір показує відсоткове збільшення часу від L0-L0 до L0-L1. Найнижчий набір міток даних є важливим і безпосередньо стосується нашого процесу оцінювання. Ці значення дозволяють нам визначити час встановлення CloudSkulk на основі того, які дії з навантаженням може виконувати користувач. Оскільки час встановлення CloudSkulk майже повністю залежить від часу, що базується на вкладеному кроці живої міграції, ми наблизимо загальний час встановлення нижче, посилаючись на нього як на цілочисельну стелю його вкладеного часу живої міграції від початку до кінця. Таким чином, найкращий час встановлення руткіту CloudSkulk становить ~26 секунд. Це відбувається, коли цільове гостьове навантаження простоює. Час встановлення CloudSkulk, коли цільовий гостьовий користувач виконує інтенсивні робочі навантаження вводу/виводу, становить ~29 секунд; для робочих навантажень, що витрачають багато ресурсів на процесор/пам'ять, час становить ~820 секунд. Найвищий набір даних на рисунку 5.9 менш релевантний, але все ж важливий для розгляду. Ці мітки даних показують нам, скільки додаткового часу буде додано до нашого

наскрізного часу міграції в реальному часі порівняно з номінальними міграціями типу L0-L0.

Останнє важливе зауваження полягає в тому, що було зафіксовано час простою під час міграції в реальному часі.для кожної точки даних. Однак ці дані не відображалися з двох основних причин: у всіх середовищах виконання стандартне відхилення було високим (~50-60%), і жоден час простою не перевищував 103 мс, що, можливо, було б непомітно для гостьового користувача.

3.3. Мікро-бенчмарки

Для точнішого вимірювання накладних витрат ми провели низку мікробенчмарк-тестів. Як наш мікробенчмарк ми обрали lmbench версії 3.0-a9 [31]. Наші експериментальні результати представлені в таблиці 3.2, таблиці 3.3, таблиці 3.4 та таблиці 3.5.

Таблиця 3.2 – lmbench: Арифметичні операції – час у наносекундах

Config	integer bit	integer add	integer div	integer mod	float add	float mul	float div	double add	double mul	double div
Level0	0.26	0.13	5.94	6.37	0.75	1.25	3.31	0.75	1.25	5.06
Level1	0.25	0.13	5.96	6.39	0.75	1.26	3.32	0.75	1.26	5.07
Level2	0.26	0.13	6.14	6.59	0.78	1.30	3.43	0.78	1.30	5.23

Таблиця 3.3 – lmbench: Процеси – час у мікросекундах

Config	signal handler installation	signal handler overhead	protection fault	pipe latency	AF_UNIX sock stream latency	fork+exit	fork+execve	fork+/bin/sh -c
Level0	0.075	0.50	0.27	3.49	3.58	74.6	245.8	918.7
Level1	0.096	0.58	0.29	6.75	5.37	73.65	275.05	966.67
Level2	0.10	0.60	0.32	65.49	43.98	242.19	588.50	1826.00

Таблиця 3.4 – lmbench: Затримка файлової системи – кількість створення/видалення файлів за секунду – разиу мікросекундах

File Size / Level	File Size 0K		File Size 1K		File Size 4K		File Size 10K	
	File Creation	File Deletion	File Creation	File Deletion	File Creation	File Deletion	File Creation	File Deletion
Level0	126,418	379,158	99,112	280,884	99,627	279,893	79,869	214,767
Level1	121,718	361,860	97,073	268,977	95,821	273,863	77,118	204,260
Level2	2,430	320,349	62,933	262,478	96,588	251,766	70,098	196,449

Таблиця 3.5 – lmbench: Перемикання контексту – час у мікросекундах

Config	2p 0K	2p 16K	2p 64K	4p 0K	4p 16K	4p 64K	8p 0K	8p 16K	8p 64K	16p 0K	16p 16K	16p 64K
Level0	1.43	1.71	1.91	1.67	1.84	1.90	1.84	1.94	2.06	1.88	2.03	2.34
Level1	3.09	3.10	3.52	1.78	3.43	2.84	2.73	3.24	1.96	3.72	5.35	4.21
Level2	19.02	28.21	28.39	10.28	34.70	53.94	5.99	32.42	24.49	6.42	39.53	4.41

Як видно з цих чотирьох таблиць, віртуалізація (включаючи вкладену віртуалізацію) має незначний вплив на всі арифметичні операції. Також для операцій створення та видалення файлів продуктивність як Рівня 2, так і Рівня 1 відповідає базовому рівню, тобто продуктивності Рівня 0. Для перемикань контексту Рівень 1 зазнає значного зниження продуктивності, а Рівень 2 ще більше зазнає зниження продуктивності. Крім того, розгалуження процесу генерує значні накладні витрати на продуктивність Рівня 2, ймовірно, через додаткові пастки гіпервізора Рівня 0 [48].

3.4. Захист від CloudSkulk

Руткіти, встановлені на скомпрометованій системі, дозволяють зловмисникам приховуватися та непомітно отримувати доступ до системних ресурсів. Наш руткіт, CloudSkulk, дозволяє зловмиснику в хмарному середовищі отримати контроль над віртуалізованою машиноюта послуги власника гостьової віртуальної машини жертви, що працює в хмарі, залишаючись непоміченим. Наш унікальний тип руткіту важко виявити, оскільки його встановлення використовує звичайні інструменти та функції, що зустрічаються в типових хмарних операційних середовищах, а його системний вплив після встановлення мінімальний. Тим не менш, руткіт CloudSkulk залишає сліди своєї присутності, про які ми незабаром поговоримо. Для хмарних середовищ, які не прагнуть відстежувати ці ознаки, вплив руткіту CloudSkulk може бути досить високим. Однак, оскільки метою нашого дослідження є підвищення безпеки хмарних систем від таких атак, хмарні середовища, які впроваджують монітори для

виявлення цих ознак, можуть пом'якшити вплив руткіт-атак RITM на основі вкладених віртуальних машин.

Ми можемо класифікувати різні методи, які можна використовувати для запобігання та виявлення присутності руткіту CloudSkulk, за такими групами: 1. Детектори продуктивності. 2. Обмежувальні команди. 3. Інструменти аналізу пам'яті.

3.5. Продуктивність детектора

Встановлення руткіту RITM на основі вкладеної віртуальної машини (VM) неминуче призводить до зниження продуктивності віртуальної машини жертви, що працює в хмарі. Це пов'язано з тим, що гостьова система працює поверх додаткового рівня віртуалізації після атаки, від L1 до L2. Це зниження продуктивності можна дещо пом'якшити, коли гостьова система використовує паравіртуалізовані пристрої, такі як драйвери Virtio, детально описані в розділі 5.1, але не можна повністю мінімізувати. Таким чином, руткіт RITM на основі вкладеної віртуальної машини можна виявити шляхом моніторингу аномалій продуктивності, що накладаються на гостьову віртуальну машину після міграції в реальному часі.

Незважаючи на деякі значні впливи на продуктивність L2 IO, аномалії продуктивності, спричинені нашим руткітом, повинні збігатися з усіма іншими стандартними вкладеними віртуалізаціями QEMU/KVM. Методи хмарної безпеки, що намагаються виявляти наш руткіт через аномалії продуктивності, спричинені вкладеною віртуалізаціюОднак це слабка стратегія. Вона складається з двох частин: 1. Кожна віртуальна машина є дуже динамічною, тому визначення порогових значень продуктивності може бути складним і ненадійним, 2. Крім того, оскільки популярність вкладеної віртуалізації продовжує зростати, ймовірно, її вплив на продуктивність продовжуватиме зменшуватися.

Обмежувальні команди.

Програмне забезпечення для хмарної безпеки може накладати обмеження на команди хоста, щоб запобігти вкладеностіАтаки RITM на основі віртуальних машин. Шляхом моніторингу та обмеження команд живої міграції, системних інструментів, які можна використовувати для пошуку цільових конфігурацій гостьових віртуальних машин, та інших системних інструментів, що використовуються під час встановлення, руткіт CloudSkulk може бути повністю заблокований. Цей метод, хоча й життєздатний, значно слабший за раніше обговорювані стратегії, оскільки набір команд та інструментів, які зловмисник може використовувати під час встановлення руткіту, важко квантувати.

Наприклад, на третьому кроці встановлення руткіту CloudSkulk, зловмисник повинен знайти конфігурації віртуалізації QEMU цільової гостьової віртуальної машини. Кількість способів, якими зловмисник може отримати ці дані, важко приблизно оцінити або визначити, і тому це слабка стратегія запобігання будь-яким таким командам. Ще однією слабкістю цієї стратегії є те, що запобігання таким командам, як набір QEMU Monitor, значно знижує функціональну здатність хоста перевіряти, налаштовувати та контролювати різні аспекти своїх гостьових віртуальних машин. Крім того, якщо програмне забезпечення для хмарної безпеки накладає обмеження привілеїв на команди живої міграції, рівень контролю зловмисника необхідний для отримання рівень доступу просто нижчий за обмеження – з найнижчими привілеями кільця 0. Це може зробити встановлення руткіту CloudSkulk більш прагматично складним для зловмисника, але не повністю запобігає атаці і тому також є слабкою стратегією.

Інструменти аналізу пам'яті

З усіх представлених нами методів, інструменти аналізу пам'яті можуть забезпечити найнадійніший та найсильніший спосіб виявлення наявності вкладеної віртуальної машини на основі RITM. Незалежно від кількості рівнів віртуалізації в цільовій гостьовій віртуальній машині, після встановлення руткіту CloudSkulk це число збільшиться на один. Якщо модифікації (наразі невідомі) до кодової бази віртуалізації QEMU не будуть впроваджені в середовищі L1 (руткіт), руткіт CloudSkulk наразі не може бути створений без нав'язування цієї аномалії пам'яті на цільовій хмарній платформі хоста.

Перший проект та реалізацію фреймворку для аналізу пам'яті, що дозволяє аналізувати та розпізнавати гіпервізори, вкладені чи одиночні, розробили Граціано М. та його команда.дослідників з Eurecom, Франція, у 2013 році [16]. Команда розробила інструмент для аналізу пам'яті гіпервізора, який аналізував структури гіпервізора, знайдені у фізичних дампах пам'яті. Їхній інструмент успішно розпізнав кілька вкладених гіпервізорів з відкритим кодом та комерційних, встановлених з різними конфігураціями. Інструменти для аналізу пам'яті, подібні до розробленого Граціано, М., можуть використовуватися програмним забезпеченням для виявлення хмарної безпеки для виявлення вкладених RITM-атак на основі віртуальних машин. Таке програмне забезпечення для виявлення може використовувати інструменти для аналізу пам'яті для визначення рівнів віртуалізації, зокрема кількості вкладених гіпервізорів, до та після виклику будь-якого набору команд живої міграції, що залежать від віртуальної машини, на хості. Наявність руткіту CloudSkulk можна виявити, якщо збільшити цю кількість шарів віртуалізації. Якщо хмара розгортає стратегію живої міграції віртуальних машин на основі попереднього копіювання, як у випадку з QEMU/KVM, виявлення цієї аномалії гіпервізора можна здійснити шляхом порівняння кількості вкладених гіпервізорів.безпосередньо перед етапом ітеративного копіювання та одразу після етапу активації.

Також важливо зазначити, хоча це й не досліджувалося, що рішення для будь-якого зЗловмисник або захисник можуть бути реалізовані в кодовій базі QEMU. Виявлення або запобігання виявленню кількості рівнів віртуалізації до та після міграції в реальному часі може бути додано до програмного забезпечення QEMU, щоб спробувати знову виявити або запобігти цьому типу атаки. Іншим можливим програмним рішенням QEMU може бути повна заборона створення додаткових рівнів гіпервізорів, що застосовуються безпосередньо до, під час та одразу після міграції в реальному часі.

4 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ХОРОНИ ПРАЦІ

4.1 Ергономічні проблеми безпеки життєдіяльності при роботі за комп'ютером

У сучасному світі все більше людей проводять значну частину свого робочого часу за комп'ютером, що призводить до ряду ергономічних проблем, які негативно впливають на безпеку та здоров'я людей. Неправильна позиція тіла, незручне розташування робочого місця, тривале сидіння та напружена робота з клавіатурою та мишею спричиняють м'язове напруження, біль у спині та напругу очей, що викликає дискомфорт та незручності.

Розташування робочого місця відіграє важливу роль у забезпеченні безпеки та здоров'я під час роботи за комп'ютером. Оптимальна висота столу та стільця є ключовим фактором для забезпечення комфорту і підтримки правильної позиції тіла. Стіл належної висоту, щоб лікті могли спокійно розташовуватися на клавіатурі, а стопи – на підлозі. Стілець обладнаний підтримкою для спини та належними регульованими підлокітниками. Клавіатура розташована на рівні ліктів, монітор – належним чином вирівняний перед очима, а миша – в зоні доступу для зап'ястя.

Правильне розташування робочого місця сприяє уникненню незручностей та проблем зі здоров'ям. Ергономічні пристрої підтримки є корисними для забезпечення комфорту та запобігання напрузі м'язів. Використання регульованих підлокітників та підставок для зап'ястя допомагає знизити напругу на м'язах і запобігти незручностям. Оптимальне розташування робочого місця повинне враховувати індивідуальні особливості користувача та забезпечувати комфортні умови для роботи.

Правильна позиція тіла та рухи є ключовими факторами для забезпечення комфорту та запобігання напрузі м'язів та незручностям.

Важливо пам'ятати про правильну позицію спини та уникати підгорблення або надмірного нахилу голови під час тривалої роботи за комп'ютером. Регулярні перерви для розтяжки та руху є важливими для підтримання здоров'я. Прості фізичні вправи, такі як розтягування шиї, плечей, рук і ніг, допомагають зняти напругу з м'язів та покращити кровообіг. Регулярні перерви дозволяють розслабитися і зберегти енергію для продуктивної роботи. Очі є одними з найбільш вразливих органів під час роботи за комп'ютером. Постійне спрямування погляду на екран призводить до напруження та втому очей.

Для зменшення негативного впливу необхідно використовувати екрани з антиблисковим покриттям, яке знижує відблиск та рефлексію світла. Також важливо налаштовувати яскравість та контрастність екрану для комфортного сприйняття. Щоб зменшити напругу на очі, необхідно робити перерви для відпочинку, фокусуючись на далеких предметах або виконуючи вправи для розслаблення очей.

Використання неправильної клавіатури та миші призводить до м'язового напруження і тунельного синдрому. Важливо використовувати ергономічні клавіатури та миші з додатковою підтримкою для зап'ястя та комфортною формою. Правильна позиція рук та зап'ястя під час роботи з ними має велике значення. Регулярні перерви для розтяжки рук та масажу зап'ястя допомагають уникнути негативних наслідків від тривалого використання клавіатури та миші. Розглядаючи ергономічні проблеми безпеки життєдіяльності при роботі за комп'ютером, варто зазначити, що їх вирішення є ключовим для забезпечення безпеки та здоров'я під час роботи.

Дотримання принципів ергономіки, налагодження робочого місця, правильна позиція тіла та виконання фізичних вправ сприяють покращенню безпеки та створенню здорового робочого середовища. Застосування ергономічних пристроїв підтримки, таких як регульовані підлокітники та підставки для зап'ястя, також сприяє комфорту та попередженню

незручностей. Загальною метою є створення безпечного та здорового робочого середовища для людей, які працюють за комп'ютером та виконують дослідження захищеності веб сервісу електронного навчання Atutor, що забезпечує збереження здоров'я та підвищення продуктивності працівників, сприяє запобіганню травмам та ергономічним проблемам, а також сприяє загальному комфорту та задоволенню від роботи.

Додатково, важливо зазначити, що належна освітленість приміщення також є важливим фактором, який впливає на комфорт та здоров'я під час роботи за комп'ютером. Потрібно забезпечити достатнє природне або штучне освітлення, яке не перевантажує очі. Розміщення робочого місця біля вікна або використання належної освітлювальної техніки допомагає забезпечити оптимальні умови освітлення.

Важливо також уникати відблисків на екрані, розташовуючи монітор під правильним кутом до джерел світла. Безпека та конфіденційність інформації також є важливим аспектом при роботі за комп'ютером. Важливо зберігати конфіденційні дані та захищати їх від несанкціонованого доступу. Використання паролів, шифрування даних та регулярне оновлення програмного забезпечення допомагає забезпечити безпеку інформації.

Крім того, необхідно усвідомлювати потенційні загрози з боку шкідливих програм та фішингових атак і приймати заходи для їх запобігання, такі як використання антивірусного програмного забезпечення та обережне відкривання електронних листів та посилань. Регулярне навчання та свідомість про важливість ергономіки та безпеки при роботі за комп'ютером є важливими.

Працівники повинні мати бути проінформовані про правильні методи роботи, виконання пауз і фізичних вправ, а також процедур безпеки. Організації можуть проводити навчання та інформаційні тренінги, щоб підвищити свідомість та забезпечити правильну поведінку під час роботи за комп'ютером.

Враховуючи всі ці аспекти, створення комфортного, безпечного та здорового робочого середовища є важливим завданням як для працівників, так і для роботодавців. Захист здоров'я та добробуту працівників при роботі за комп'ютером не тільки покращує їхню якість життя, але й сприяє збільшенню продуктивності та задоволення від роботи, що має позитивний вплив на всю організацію.

4.2 Організація безпечної роботи електроустаткування задіяного при роботі системи електронного навчання

Безпечна робота електроустаткування є ключовим елементом для забезпечення стабільної та безперебійної роботи системи електронного навчання. Це включає в себе правильне проектування, встановлення, експлуатацію та обслуговування електроустаткування. Дотримання стандартів і правил охорони праці допомагає мінімізувати ризики електричних ударів, пожеж та інших небезпек.

Перед встановленням електроустаткування необхідно ретельно оцінити його відповідність вимогам системи електронного навчання. Вибір обладнання повинен базуватися на таких критеріях: надійність та безпека: обладнання має відповідати стандартам якості та безпеки, мати сертифікати відповідності та бути розрахованим на довготривалу експлуатацію; відповідність технічним вимогам: обладнання повинно підтримувати необхідні технічні параметри, такі як напруга, потужність, тип з'єднання тощо;сумісність з іншими компонентами: всі компоненти системи повинні бути сумісні між собою, щоб уникнути збоїв та небезпек при експлуатації/

Під час встановлення необхідно дотримуватися таких заходів безпеки:

Правильне заземлення: всі пристрої повинні бути заземлені відповідно до норм, щоб уникнути накопичення статичної електрики та можливості удару струмом;використання захисних пристроїв: встановлення автоматичних

вимикачів, пристроїв захисного вимкнення (ПЗВ) та інших захисних засобів є обов’язковим для забезпечення безпеки; професійний монтаж: монтаж повинен виконуватися кваліфікованими спеціалістами з дотриманням всіх норм і правил.

Для забезпечення безпечної експлуатації електроустаткування слід дотримуватися таких рекомендацій: регулярний контроль та технічне обслуговування: обладнання повинно регулярно перевірятися на наявність зношення, перегріву, пошкоджень проводів та інших дефектів. Технічне обслуговування повинно проводитися відповідно до інструкцій виробника; контроль температурного режиму: устаткування не повинно перегріватися. Необхідно забезпечити достатню вентиляцію та уникати розташування пристроїв поблизу джерел тепла; правильне використання: обладнання має використовуватися тільки за призначенням, з дотриманням інструкцій та рекомендацій виробника.

При обслуговуванні та ремонті електроустаткування необхідно дотримуватися таких заходів безпеки: відключення живлення: перед проведенням будь-яких робіт обладнання має бути відключене від джерела живлення; використання засобів індивідуального захисту (ЗІЗ): спеціалісти повинні використовувати відповідні ЗІЗ, такі як діелектричні рукавички, килимки, інструменти з ізоляційними покриттями; дотримання процедур: всі ремонтні роботи повинні проводитися відповідно до технічних інструкцій та нормативних документів;

Управління ризиками та навчання персоналу є невід'ємною частиною забезпечення безпеки електроустаткування: ідентифікація ризиків: постійний аналіз можливих ризиків та їх мінімізація є ключовим аспектом безпеки. Для цього проводяться регулярні оцінки стану обладнання та аналіз умов експлуатації; навчання та інструктажі: персонал повинен проходити регулярні навчання та інструктажі з питань безпечної роботи з електроустаткуванням. Це включає як базові знання, так і спеціальні навички для роботи з

конкретними видами обладнання; розробка та впровадження процедур безпеки: необхідно розробити детальні інструкції та процедури з безпеки, які повинні бути доступними для всіх працівників та регулярно оновлюватися. Таким чином, організація безпечної роботи електроустаткування задіяного при роботі системи електронного навчання вимагає комплексного підходу, що включає вибір надійного обладнання, правильне його встановлення, регулярне технічне обслуговування, навчання персоналу та управління ризиками. Дотримання цих заходів дозволяє забезпечити стабільну та безпечну роботу системи, що є критично важливим для ефективного функціонування освітнього процесу.

ВИСНОВКИ

У межах кваліфікаційної роботи було здійснено комплексне дослідження архітектури, функціональних можливостей та методів впровадження руткіту CloudSkulk у хмарному середовищі з використанням технологій KVM та QEMU. Детально проаналізовано сучасні підходи до динамічної міграції віртуальних машин та техніки вкладеної віртуалізації, що дозволяють реалізувати приховані атаки типу Rootkit-in-the-Middle (RITM).

У ході розробки було побудовано структурну модель атаки, визначено основні етапи реалізації руткіту, а також виявлено технічні вимоги до хостового середовища, необхідні для коректного впровадження та маскування CloudSkulk. Особливу увагу приділено аналізу переваг цієї моделі з точки зору прихованості, сумісності з платформами IaaS та здатності обходити традиційні інструменти виявлення віртуальних загроз.

Результати дослідження засвідчили актуальність проблем безпеки у віртуалізованих хмарних середовищах, а також продемонстрували можливість створення складних програмно-апаратних рішень, здатних впливати на життєвий цикл віртуальної машини без втрати її функціональності. Отримані висновки можуть бути використані як для поглибленого аналізу ризиків у хмарних інфраструктурах, так і для розробки ефективних механізмів захисту від подібних загроз.

ПЕРЕЛІК ПОСИЛАНЬ

1. Filebench. URL: https://github.com/filebench.

2. Inc Advanced Micro Devices. Secure Virtual Machine Architecture Reference Manual. AMD. URL: http://0x04.net/doc/amd/33047.pdf

3. Sanjay P. Ahuja. Full and para virtualization, 2014. URL: https://www.geeksforgeeks.org/operating-systems/difference-between-full-virtualization-and-paravirtualization/

4. Ahmed M Azab et al. Hypersentry: enabling stealthy in-context measurement of hypervisor integrity. In Proceedings of the 17th ACM CCS, pages 38–49. ACM, 2010. URL: https://www.cs.fsu.edu/~zwang/files/ccs10.pdf.

5. Sina Bahram et al. DKSM: Subverting virtual machine introspection for fun and profit. In Proceedings of the 29th IEEE SRDS, pages 82–91. IEEE, 2010. URL: https://dblp.org/db/conf/srds/srds2010.

6. Muli Ben-Yehuda et al. The turtles project: Design and implementation of nested virtualization. In 9th USENIX OSDI, Vancouver, BC, 2010. USENIX Association. URL: https://www.usenix.org/conference/osdi10/turtles-project-design-and-implementation-nested-virtualization.

7. Tzi-cker Chiueh and Fu-Hau Hsu. RAD: A compile-time solution to buffer overflow attacks. In ICDCS 2001, pages 409–417. IEEE, 2001. URL: https://www.computer.org/csdl/proceedings-article/icdcs/2001/10770409/12Om Nwp74wJ.

8. Christopher Clark et al. Live migration of virtual machines. In Proceedings of the 2nd USENIX NSDI, pages 273–286. USENIX Association, 2005. URL: https://www.usenix.org/legacy/event/nsdi05/tech/full_papers /clark/clark.pdf.

9. CloudPassage. Share the cloud security spotlight report, 2016. URL: https://www.cloudpassage.com/resources/cloud-security-spotlight-report-2016/.

10. Cowan C., Wagle F., Pu C., Beattie S., Walpole J. Buffer overflows: Attacks and defenses for the vulnerability of the decade // DARPA Information Survivability Conference and Exposition (DISCEX'00). – 2000. – Vol. 2. – P. 119–129. URL: https://ieeexplore.ieee.org/document/821515.

11. CVEDetails.com.. Vulnerability details: CVE-2007-1744. 2007. URL: https://www.cvedetails.com/cve/CVE-2007-1744/

12. Das, B., Zhang, Y., & Kiszka, J. Nested virtualization, state of the art and future directions. 2014. URL: https://docslib.org/doc/6178099/nested-virtualization-state-of-the-art-and-future-directions

13. Elhage, N. Virtunoid: Breaking out of KVM. Black Hat USA, 2011. URL: https://blog.nelhage.com/2011/08/breaking-out-of-kvm/

14. Steiro, G., & Director, M. Great overview: Services for mobile payment. 2016. URL: https://www.enisa.europa.eu/publications/security-of-mobile-payments-and-digital-wallets

15. Garfinkel, T., & Rosenblum, M. A virtual machine introspection based architecture for intrusion detection. In: NDSS, Vol. 3, 2003. P. 191–206. URL: https://suif.stanford.edu/papers/vmi-ndss03.pdf.

16. Graziano M., Lanzi A., Balzarotti D. Hypervisor memory forensics // International Workshop on Recent Advances in Intrusion Detection. – Springer, 2013. – P. 21–40. URL: https://link.springer.com/chapter/10.1007/978-3-642-41284-4_2

17. IBM. Kernel virtual machine (KVM) – best practices for KVM. – 2012. [Online; accessed 1-June-2017]. URL: https://www.ibm.com/support/knowledgecenter/linuxonibm/liaat/liaatbestpractices_pdf.pdf

18. IntegrantSoftware. IaaS vs. PaaS vs. SaaS. – 2013. URL: https://www.integrant.com/blog/iaas-vs-paas-vs-saas

19. Jones R. et al. NetPerf: a network performance benchmark. – Information Networks Division, Hewlett-Packard Company, 1996. URL: https://github.com/HewlettPackard/netperf

20. Jones T. Linux virtualization and PCI passthrough. – 2012. URL: https://developer.ibm.com/tutorials/l-pci-passthrough/

21. Katz J. Efficient cryptographic protocols preventing “Man-in-the-Middle” attacks: PhD thesis. – Columbia University, 2002. URL: https://web.cs.ucla.edu/~rafail/STUDENTS/katz-thesis.pdf

22. King S.T., Chen P.M. SubVirt: Implementing malware with virtual machines // Proceedings of the 2006 IEEE Symposium on Security and Privacy (S&P). – IEEE, 2006. – P. 14–pp. URL: https://ieeexplore.ieee.org/document/1624011

23. Kortchinsky K. Cloudburst: A VMware guest to host escape story. – Black Hat USA, 2009. URL: https://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-PAPER.pdf

24. Koziol J., Litchfield D., Aitel D., Anley C., Eren S., Mehta N., Hassell R. The Shellcoder’s Handbook. – Indianapolis: Wiley, 2004. URL: https://www.wiley.com/en-us/The+Shellcoder%27s+Handbook%3A+Discovering+and+Exploiting+Security+Holes%2C+2nd+Edition-p-9780470080238

25. Kumar V. Rootkit – an intruder living in your kernel. – August 2009. [Online; accessed 1-June-2017]. URL: https://www.symantec.com/connect/articles/rootkit-intruder-living-your-kernel

26. Le D., Huang H., Wang H. Understanding performance implications of nested file systems in a virtualized environment // Proceedings of the 10th USENIX Conference on File and Storage Technologies, FAST’12. – Berkeley, CA, USA: USENIX Association, 2012. – P. 8–8. URL: https://www.usenix.org/conference/fast12/understanding-performance-implications-nested-file-systems

27. Liang Z., Sekar R. Automatic generation of buffer overflow attack signatures: An approach based on program behavior models // 21st Annual Computer Security Applications Conference (ACSAC'05). IEEE, 2005. C. 10–pp. URL: https://ieeexplore.ieee.org/document/1565249

28. Stone L. Bringing Pokémon GO to life on Google Cloud. 2016. URL: https://cloud.google.com/blog/products/containers-kubernetes/bringing-pokemon-go-to-life-on-google-cloud

29. McClure S., Scambray J., Kurtz G. Hacking Exposed Fifth Edition: Network Security Secrets & Solutions. McGraw-Hill/Osborne, 2005. URL: https://books.google.com/books/about/Hacking_Exposed_5th_Edition.html?id=HdBQAAAAMAAJ

30. McGraw G. Exploiting software: How to break code // Invited Talk, Usenix Security Symposium. San Diego, 2004. URL: https://www.usenix.org/legacy/publications/library/proceedings/sec04/tech/slides/mcgraw.pdf

31. McVoy L.W., Staelin C. lmbench: Portable tools for performance analysis // USENIX Annual Technical Conference (USENIX ATC). San Diego, CA, USA, 1996. C. 279–294. URL: https://www.usenix.org/conference/usenix-1996-annual-technical-conference/lmbench-portable-tools-performance-analysis

32. Mell P., Grance T. The NIST definition of cloud computing. NIST Special Publication 800-145, 2011. URL: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf

33. Cisco Visual Networking. Cisco Global Cloud Index: Forecast and Methodology, 2012–2017 (White Paper). 2013. URL: https://www.cisco.com/c/dam/global/en_au/assets/cisco-live/ites2013mel/assets/presentations/Cloud_Index_White_Paper.pdf

34. Ozkan S. CVE Details: The Ultimate Security Vulnerability Datasource. 1999. URL: https://www.cvedetails.com/

35. Pasupulati A., Coit J., Levitt K., Wu S.F., Li S.H., Kuo J.C., Fan K.-P. Buttercup: On network-based detection of polymorphic buffer overflow vulnerabilities // Network Operations and Management Symposium, 2004. NOMS 2004. IEEE/IFIP. IEEE, 2004. T. 1. C. 235–248. URL: https://ieeexplore.ieee.org/document/1292921

36. Riley R., Jiang X., Xu D. Guest-transparent prevention of kernel rootkits with VMM-based memory shadowing // Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID). Springer, 2008. C. 1–20. URL: https://link.springer.com/chapter/10.1007/978-3-540-87403-4_1

37. Roesch M., Green C., Kreimendahl J., et al. Snort: Lightweight intrusion detection for networks // Proceedings of the 13th USENIX LISA Conference. – 1999. – Vol. 99. – P. 229–238. URL: https://www.usenix.org/publications/library/proceedings/lisa99/full_papers/roesch/roesch_html/index.html

38. Rutkowska J. Introducing blue pill // The Invisible Things Blog. – 2006. URL: https://blog.invisiblethings.org/2006/06/22/introducing-blue-pill.html

39. Rutkowska J. Subverting VistaTM kernel for fun and profit // Black Hat Briefings. – 2006. URL: https://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf

40. Leshchyshyn, Y., Scherbak, L., Nazarevych, O., Gotovych, V., Tymkiv, P., & Shymchuk, G. (2019, May). Multicomponent Model of the Heart Rate Variability Change-point. In 2019 IEEE XVth International Conference on the Perspective Technologies and Methods in MEMS Design (MEMSTECH) (pp. 110-113). IEEE.

41. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, September). Mathematical model of gas consumption process in the form of cyclic random process. In 2021 IEEE 16th International Conference on Computer Sciences and Information Technologies (CSIT) (Vol. 1, pp. 232-235). IEEE.

42. Kozlovskyi, V., Balanyuk, Y., Martyniuk, H., Nazarevych, O., Scherbak, L., & Shymchuk, G. (2022, April). Information Technology for Estimating City Gas Consumption During the Year. In 2022 International Conference on Smart Information Systems and Technologies (SIST) (pp. 1-4). IEEE.

43. Lytvynenko, I., Lupenko, S., Kunanets, N., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, November). Simulation of gas consumption process based on the mathematical model in the form of cyclic random process considering the scale factors. In 1st International Workshop on Information Technologies: Theoretical and Applied Problems, ITTAP 2021.

44. Kunanets, N., Pasichnyk, V., Bodnarchuk, I., Martsenko, S., Matsiuk, O., Matsiuk, A., ... & Shymchuk, H. (2019). Information system for visual analyzer disease diagnostics. In CEUR Workshop Proceedings (pp. 43-56).

45. Lupenko, S., Lytvynenko, I., Nazarevych, O., Shymchuk, G., & Hotovych, V. (2021, December). Approach to gas consumption process forecasting on the basis of a mathematical model in the form of a random cyclic process. In Proceedings of the International Conference „Advanced applied energy and information technologies 2021", 2021 (pp. 213-219). TNTU, Zhytomyr «Publishing house „Book-Druk "» LLC.

46. Lytvynenko, I., Lupenko, S., Nazarevych, O., Shymchuk, H., & Hotovych, V. (2022). Additive mathematical model of gas consumption process. Вісник Тернопільського національного технічного університету, 104(4), 87-97.

47. Nazarevych, O., Leshchyshyn, Y., Lupenko, S., Hotovych, V., Shymchuk, G., & Shabliy, N. (2020, September). Method of Gas Consumption Change-point Detection Based on Seasonally Multicomponent Model. In 2020 10th International Conference on Advanced Computer Information Technologies (ACIT) (pp. 152-155). IEEE.

48. Palianytsia, Y., Lytvynenko, I., Menoub, A., Shymchuk, H., & Dubchak, A. (2024). Development of an algorithm for identification of damage types on the surface of sheet metal.

49. Nazarevych, O., Gotovych, V., & Shymchuk, G. (2016). Information Technology for Monitoring of Municipal Gas Consumption, Based on Additive Model and Correlated for Weather Factors. Journal of Information and Computing Science, 11(3), 180-187.

50. Shymchuk, G., Lytvynenko, I., Hromyak, R., Lytvynenko, S., & Hotovych, V. (2023). Gas Consumption Forecasting Using Machine Learning Methods and Taking Into Account Climatic Indicators. In CITI (pp. 156-163).

51. Leschyshyn, Y. Z., Nazarevych, O. B., Shymchuk, G. V., Revutskyi, E. A., & Shcherbak, L. M. (2016, September). The Methods of Change Point Detection and Statistical Estimating of Dynamic of the Noise Stochastic Signals Characteristics. In THE SEVENTH WORLD CONGRESS "AVIATION IN THE XXI-st CENTURY" Safety in Aviation and Space Technologies September 19-21, NATIONAL AVIATION UNIVERSITY. Kyiv: NAU.

52. Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни Комп'ютерна графіка для студентів освітнього рівня «бакалавр» спеціальності 125 «Кібербезпека».

53. Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Конспект лекцій з дисципліни «Розподілені системи моніторингу та керування».

54. Шимчук, Г. В., Маєвський, О. В., Назаревич, О. Б., & Стадник, М. А. (2016). Конспект лекцій з дисципліни «Грід-системи та технології хмарних обчислень» для студентів освітніх рівнів «спеціаліст»,«магістр» 122 «Комп'ютерні науки та інформаційні технології».

55. Шимчук, Г., Голотенко, О., & Золотий, Р. З. (2022). Основні проблеми та загрози хмарної безпеки. Матеріали X науково-технічної

конференції „Інформаційні моделі, системи та технології “Тернопільського національного технічного університету імені Івана Пулюя, 59-60.

56. Шимчук, Г. В., Маєвський, О. В., & Назаревич, О. Б. (2016). Методичні вказівки до самостійної робіти студентів та модульного контролю знань з дисципліни «Розподілені системи моніторингу та керування» для студентів освітнього рівня «бакалавр» спеціальності 125–«Кібербезпека».

57. ШИМЧУК, Г., ШЕВЧЕНКО, Н., ШВИРЛО, К., & ГАРМАТЮК, Н. (2025). СИСТЕМА ВІДНОВЛЕННЯ ДАНИХ У БЕЗДРОТОВИХ СЕНСОРНИХ МЕРЕЖАХ НА ОСНОВІ МАШИННОГО НАВЧАННЯ. Herald of Khmelnytskyi National University. Technical sciences, 353(3.2), 246-250.