

# Détection de virtualisation de la machine et techniques d'évasion associées

March 28, 2025

Projet de semestre présenté par

**Ricardo DOS SANTOS**

**Informatique et systèmes de communication avec  
orientation Sécurité**

Professeurs-es HES responsables

**Florent GLÜCK**

**Mickaël HOERDT**

# Table des matières

<b>Liste des acronymes</b>	<b>5</b>
<b>I Introduction</b>	<b>6</b>
A Remerciements . . . . .	6
B Résumé du projet . . . . .	6
C Mise en contexte . . . . .	6
D Méthodologie de travail . . . . .	7
E Structure du rapport . . . . .	7
<b>II État de l'Art</b>	<b>8</b>
<b>III Pourquoi QEMU/KVM ?</b>	<b>11</b>
<b>IV Méthodes de détection</b>	<b>12</b>
A Nom des composants émulés . . . . .	12
I Explication . . . . .	12
II Méthode de détection . . . . .	12
III Parades . . . . .	12
B Table DMI + System Management BIOS Byte 2 . . . . .	13
I Explication . . . . .	13
II Méthode de détection . . . . .	13
III Parades . . . . .	14
C CPUID (x86_64) . . . . .	16
I Explication . . . . .	16
II Méthodes de détection . . . . .	17
III Parades . . . . .	17
D Température . . . . .	18
I Explication . . . . .	18
II Méthode de détection . . . . .	18
III Parades . . . . .	18
E Taille de Disque . . . . .	19
I Explication . . . . .	19
II Méthode de détection . . . . .	19
III Parade . . . . .	19
F Voltage . . . . .	19
I Explication . . . . .	19
II Méthode de détection . . . . .	19
G Hostname + Username . . . . .	20
I Explication . . . . .	20

II	Méthode de détection . . . . .	20
III	Parades . . . . .	20
H	Ventilateurs CPU . . . . .	20
I	Explication . . . . .	20
II	Méthode de détection . . . . .	20
I	RDTSC force VM exit . . . . .	20
I	Explication . . . . .	20
II	Méthode de détection . . . . .	21
III	Parade . . . . .	21
J	Adresse MAC . . . . .	21
I	Explication . . . . .	21
II	Méthode de détection . . . . .	21
III	Parades . . . . .	22
K	Tables ACPI . . . . .	22
I	Explication . . . . .	22
II	Méthode de détection . . . . .	22
III	Parades . . . . .	22
L	Dmesg . . . . .	23
I	Explication . . . . .	23
II	Méthode de détection . . . . .	23
III	Parades . . . . .	23
<b>V</b>	<b>Création d'une VM et exécution des techniques d'évasions</b>	<b>23</b>
A	Pré-requis . . . . .	23
I	Paquets à installer . . . . .	24
II	Patching de QEMU . . . . .	24
III	Patching de EDKII . . . . .	25
B	Mise en place des différentes parades . . . . .	25
<b>VI</b>	<b>Résultats &amp; discussion</b>	<b>27</b>
A	Systemd-detect-virt . . . . .	27
I	Résultats . . . . .	27
II	Discussions . . . . .	28
B	VMAware . . . . .	28
I	Résultat . . . . .	28
II	Discussion . . . . .	29
C	Al-khaser . . . . .	29
I	Résultats . . . . .	29
II	Discussion . . . . .	30
D	Pafish . . . . .	30

I Résultats . . . . .	30
II Discussion . . . . .	30
E Anti-cheats & Autres . . . . .	31
F Limitations de ce projet . . . . .	31
<b>VII Conclusion</b>	<b>31</b>
<b>Références documentaires</b>	<b>33</b>
 <b>Liste des illustrations</b>	
1 Screenshot d'une VM Windows 10 dont le nom de disque contient le mot "QEMU" . . . . .	12
2 Screenshot d'une table DMI/SMBIOS classique sur QEMU	14
3 Screenshot de la table DMI/SMBIOS après modification de QEMU . . . . .	16
4 Screenshot de la VM sur Arch Linux, exécutant la commande <i>dmesg</i> . . . . .	23
5 Screenshot des résultats de Systemd-detect-virt . . . . .	27
6 Screenshot des résultats de VMAware . . . . .	28
7 Screenshot des résultats de Al-khaser pour QEMU . . . . .	30

## Liste des acronymes

.

**CPU** Unité Centrale de Traitement (Central Processing Unit).

**CPUID** Identification du Processeur (CPU Identification).

.

**EDKII** Kit de Développement EFI II (EFI Development Kit II).

**I/O** Entrée/Sortie (Input/Output).

**I2C** Circuit Intégré Interconnecté (Inter-Integrated Circuit).

**ID** Identification.

**KVM** Machine Virtuelle Basée sur le Noyau (Kernel-based Virtual Machine).

**Pafish** Outil de Détection de Phishing pour Environnements Virtualisés  
(A phishing detection tool designed to check for virtualized environments).

**QEMU** Émulateur Rapide (Quick Emulator).

**SMBIOS** Système de Gestion du BIOS (System Management BIOS).

**UEFI** Interface de Micrologiciel Extensible Unifiée (Unified Extensible Firmware Interface).

**VM** Machine Virtuelle (Virtual Machine).

**VMware** Outil de Détection de Machines Virtuelles (Virtual Machine Aware).

# I Introduction

## A Remerciements

Mes grands remerciements à Mr. Glück pour l'encadrement du projet, ainsi qu'à Mr Hoerdts pour avoir proposé mon sujet pour les projets de semestre.

Je remercie aussi Mr Giles Antoine, pour ses conseils lors de la réalisation de ce document et de ses idées lors du projet.

## B Résumé du projet

Le projet a débuté avec une phase de découverte des différents outils de détection de virtualisation d'une machine. Par virtualisation, nous parlons de *full system emulation*, avec l'hyperviseur QEMU utilisant KVM, sans l'ajout de *libvirt* ou *virt-manager*. Ensuite, il y a eu une recherche des différentes techniques employées par ces outils et des mécanismes employés pour inférer la virtualisation d'un système. Ensuite, une étude des options de QEMU via diverses sources, telles que la documentation de ce dernier ou celle d'autres acteurs comme Intel, AMD (Advanced Micro Devices Inc.) ou DMTF (Distributed Management Task Force). Avec une bonne compréhension des mécanismes employés, il a été possible d'appliquer des parades à ces derniers. Ceci est possible via diverses options, telles que des arguments natifs à QEMU en ligne de commande, une image UEFI (Unified Extensible Firmware Interface) ou allant jusqu'à la modification de QEMU lui-même au niveau du code source. Enfin, après avoir appliqué une parade, il y a eu des phases de test afin de savoir si les différents outils de détection ont pu être contournés ou non. Ces tests ont été répétés jusqu'à ce que l'outil soit berné. Ces tests se sont déroulés principalement sur Linux (distribution Arch Linux) et Windows 10. Tous ces processus nous amènent à une machine virtuelle (VM : Virtual Machine) qui se veut la plus proche possible d'une machine physique, du point de vue de l'*Operating System (OS) Guest*. Un script a été conçu afin d'automatiser la récupération des informations nécessaires à la VM.

## C Mise en contexte

Dans le cadre de mon projet de Bachelor, je voulais créer une machine virtuelle indétectable en tant que telle afin de pouvoir exécuter certaines applications dans un environnement virtualisé, alors que ces dernières ne

le permettent pas.

Ce désir a commencé par une recherche personnelle afin de mettre en place un dispositif de ce type. Il m'a été proposé d'en faire mon projet de semestre, et c'est exactement ce qui a été fait.

## D Méthodologie de travail

La première étape du projet est la recherche d'outils permettant la détection d'un OS virtualisé.

Le but de cette recherche est de lister les différents tests à passer par notre VM. Cela nous permet de connaître les différents mécanismes déployés aux différents niveaux de l'OS.

La deuxième étape consiste à examiner le code source des outils afin de comprendre en détail ce qui est fait pour détecter la virtualisation d'un système.

Ensuite, il faut rechercher comment contourner la technique de détection, via des documentations ou des options natives à l'hyperviseur. Si certaines techniques ne sont pas modifiables via la ligne de commande, il faut trouver des alternatives ou modifier directement QEMU afin de parer à la détection.

L'étape finale est de vérifier que la technique pour laquelle nous avons appliqué une parade ne détecte plus la virtualisation de l'*OS guest*, et de répéter cela pour tous les outils énumérés jusqu'à ce que la machine soit considérée comme physique.

## E Structure du rapport

Le rapport commencera par justifier le choix de l'hyperviseur choisi pour ce projet.

Ensuite, le rapport donnera la liste des outils avec lesquels les tests de détection de virtualisation ont été réalisés. En plus de cela, il y aura une énumération des différentes techniques à parer, toutes séparées en trois sous-chapitres :

- Explication des aspects techniques de la méthode de détection de virtualisation.

- Explication de la méthode en elle-même.
- Les parades appliquées ou possibles afin de contourner la détection.

L'implémentation des différentes techniques qui ont été possibles sera documentée, ainsi que le script permettant d'automatiser l'exécution de la VM.

Pour finir, il y aura une présentation des résultats ainsi qu'une discussion sur ces derniers.

## II État de l'Art

Le projet utilise un hyperviseur de type 1. Cela signifie que l'hyperviseur a un accès direct aux ressources de la machine physique.[\[1\]](#)

D'autres hyperviseurs de ce type sont :

- Microsoft Hyper-V
- VMware ESXi
- Xen

À la différence des hyperviseurs de type 2, qui passent d'abord par l'hôte de la machine physique pour pouvoir accéder à des ressources matérielles.

Des hyperviseurs de type 2 sont :

- Oracle VirtualBox
- VMware Workstation

La machine virtuelle fait du *full system emulation*. En d'autres termes, l'hyperviseur émule un système entier, y compris le matériel, à la différence de l'*application emulation*, qui n'émule que l'*Application Programming Interface* (API).[\[2\]](#)

Pour des raisons de performance de la machine virtuelle, la technique de virtualisation du CPU utilisée est la *hardware-assisted virtualization*.

Cela signifie que la virtualisation du CPU est assistée par le processeur physique à l'aide d'instructions dédiées à la virtualisation, améliorant grandement les performances de la machine virtuelle par rapport à d'autres techniques telles que la *dynamic binary translation* ou la paravirtualisation.



Le CPU est séparé en deux modes :[3]

- non-root :

L'*OS guest* tourne dans ce mode. Si ce dernier ne fait pas d'appel requérant des privilèges, il accède directement au matériel.

- root :

Le *virtual machine manager* (VMM) tourne dans ce mode. Si l'OS guest fait un appel demandant des privilèges, il demande un changement de mode (appelé VMExit), qui fait passer l'appel en mode root et, si ce dernier est valide, permet l'exécution de l'instruction privilégiée via la VMM.

Jusqu'à présent, la manière la plus répandue de cacher une VM est de passer par *libvirt* et *virt-manager*.

Cela est fait grâce à un fichier XML contenant les différents composants à émuler et leur mode d'émulation, avant que *libvirt* génère une commande QEMU/KVM correspondante.

Les informations permettant de générer une VM indétectable étaient dispersées et difficiles à trouver.

Récemment, [un projet Git](#) très intéressant propose lui aussi une automatisation des patches et des techniques décrites dans ce document, à la différence qu'il utilise *libvirt* et *virt-manager*, avec un *template* XML des options à ajouter pour parer à la détection.

Ce document cherche à expliquer plus en détail les techniques de détection et à justifier pourquoi ces options ou ces patches sont nécessaires afin d'éviter la détection de virtualisation.

Les outils qui détectent la virtualisation :

- **Linux :**
  - [systemd-detect-virt](#)

Le programme Systemd est un ensemble de blocs qui forment la

fondation d'un système Linux. C'est un gestionnaire de système et de services ayant le *Process Identification* (PID) 1.[4]

Dans cette suite de programmes offerte par Systemd, nous avons *systemd-detect-virt*, un outil de détection de virtualisation sous Linux, écrit en C.

- [virt-what](#)

*virt-what* est un *shell script* qui peut être utilisé afin de détecter si le programme tourne dans une VM.[5]

- **Windows :**

- [Al-Khaser](#)

*Al-Khaser* est un *proof of concept malware* qui cherche à faire réagir les antivirus présents sur le système.[6]

Parmi les techniques utilisées par le programme, certaines visent directement les VMs et cherchent à les détecter (de manière générique et spécifique).

Écrit en C, C++ et C#.

- [Paranoid Fish \(Pafish\)](#)

*Paranoid Fish* (Pafish) est un outil de détection de VM et d'analyse d'environnement *malware*. [7]

Écrit en C.

- **Multi-plateforme :**

- [VMAware](#)

*VMAware* (VM + Aware) est une bibliothèque *cross-platform* écrite en C++ pour la détection de machines virtuelles.[8]

Écrit en C++.

### III Pourquoi QEMU/KVM ?

L'hyperviseur de choix pour ce projet est QEMU avec KVM.

Le choix de KVM est justifié pour plusieurs raisons :

- KVM a été intégré dans le noyau Linux depuis 2007 et se trouve dans toutes les implémentations du noyau officiel Linux.
- L'utilisation de KVM permet de passer d'un hyperviseur de type 2 à un type 1, apportant une amélioration non négligeable des performances dans les VMs.
- Il permet d'utiliser la *hardware-assisted virtualization* du CPU, ainsi qu'une option importante qui permet de contourner une technique de détection de virtualisation.
- Comparé à d'autres hyperviseurs de type 1 :
  - KVM ne requiert aucune installation et peu de configuration sur une grande partie des distributions Linux, comparé à Xen ou VirtualBox.
  - Il ne requiert pas d'abonnement, contrairement à VMware ESXi.
  - Il est open-source (toujours comparé à VMware ESXi), ce qui nous permet de modifier le code source en cas de besoin ou de l'inspecter pour effectuer des recherches.

QEMU en ligne de commande permet un contrôle total de chaque composant de la machine virtuelle.

Si *libvirt* et *virt-manager* permettent d'exécuter des VMs de manière automatique, ce n'est pas toujours clair ce qui est généré en arrière-plan pour l'exécution de la VM.

Ainsi, en restant en ligne de commande, il est possible d'avoir une vue constante sur chaque changement effectué avec l'ajout ou la suppression des éléments de la machine.

QEMU est un hyperviseur simple à utiliser et permet une flexibilité indispensable pour le projet.

## IV Méthodes de détection

### A Nom des composants émulés

#### I Explication

Dans un hôte physique, les noms des composants sont généralement formés à l'aide de la marque du produit ainsi que du modèle.

Cela est dû au fait qu'il y a un marché et une compétition entre plusieurs entités.

Dans un système virtualisé, les marques sont remplacées par les noms des hyperviseurs, alors que le modèle n'importe peu dans le nom.

#### II Méthode de détection

Un programme est capable de détecter si un système est virtualisé si les noms des composants contiennent des mots tels que "QEMU", "KVM" ou "Virtual".

Voici un exemple dans la figure 1 :

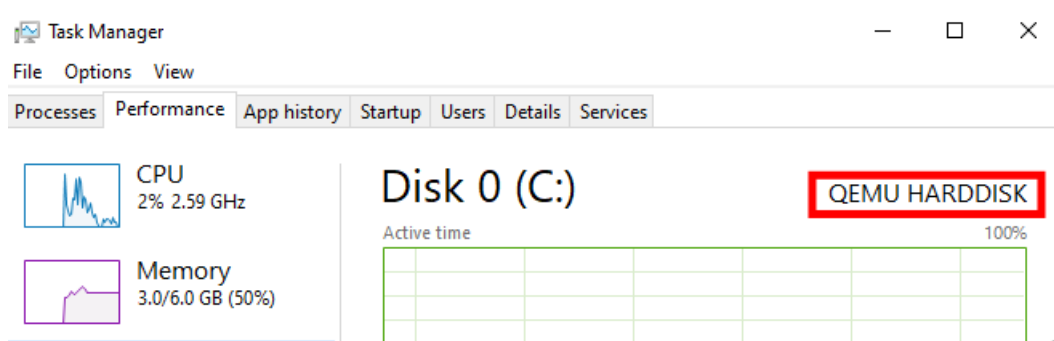


Figure 1: Screenshot d'une VM Windows 10 dont le nom de disque contient le mot "QEMU"

Réalisé par DOS SANTOS Ricardo

### III Parades

Ces noms sont hard-codés dans le code source de QEMU. Afin de pouvoir modifier l'exemple de la figure 1, il faudra modifier le fichier *hw/ide/core.c* :

```
strcpy(s->drive_model_str, "QEMU HARDDISK");
```

Le choix du nom est libre, mais il est recommandé d'utiliser des noms de véritables composants physiques.

Par exemple :[9]

```
strcpy(s->drive_model_str, "Samsung SSD 980 500GB");
```

Tout modifier à la main est très fatigant et peu pratique, donc il est conseillé d'utiliser le patch git donné par [le projet git de Scrut1ny](#).

## B Table DMI + System Management BIOS Byte 2

### I Explication

La *Desktop Management Interface* (DMI) était un framework qui servait de premier standard de management de bureau (*Desktop Management*). Cela permettait de traquer les composants de nos ordinateurs.[10]

Le développement actif de la DMI s'est arrêté le 31 décembre 2003 et la fin de vie du produit a été conclue définitivement en mars 2005.[11]

Depuis, le nouveau standard est System Management BIOS (SMBIOS), repris par DMTF (développeurs de la DMI) afin de permettre une communication simplifiée des composants matériels aux plateformes qui souhaitent récupérer les informations de ces derniers.[12]

### II Méthode de détection

Lorsqu'un système est virtualisé, si aucune option n'est spécifiée, alors QEMU charge le fichier binaire `/usr/share/qemu/bios-256k.bin` qui est une implémentation de *Legacy BIOS*.

Sous Linux, il est possible de détecter la virtualisation d'un OS en scanant des fichiers exposés dans le dossier `/sys/class/dmi/id`, correspondant à certaines informations contenues dans les tables DMI/SMBIOS :

- Nom de produit, `/sys/class/dmi/id/product_name`
- Vendeur du système, `/sys/class/dmi/id/sys_vendor`
- Vendeur du châssis, `/sys/class/dmi/id/board_vendor`

- Vendeur du BIOS, `/sys/class/dmi/id/bios_vendor`
- Spécifique aux VMs Hyper-V[13], la version du produit `/sys/class/dmi/id/product_version`
- Modalias, un formatage des informations système données précédemment `/sys/class/dmi/id/modalias`

Chacun de ces fichiers contiendra des chaînes spécifiant la VMM utilisée, permettant la détection de virtualisation.

En plus de cela, dans la spécification de la SMBIOS, dans la table d'extension des caractéristiques byte 2, le 4ème bit signifie que le système est virtualisé, mais toujours dans cette même spécification, le bit n'étant pas mis à 1 ne doit rien inférer sur la virtualisation du système.

Il est aussi possible de vérifier le nombre de tables SMBIOS présentes dans la machine. Si ce dernier est plus petit que le nombre minimal de tables présentes dans une machine physique, alors nous pouvons inférer que nous sommes dans une VM.

Voici la figure 2 avec un exemple de table DMI/SMBIOS qui n'est pas modifiée par QEMU:

```
Handle 0x0100, DMI type 1, 27 bytes
System Information
  Manufacturer: QEMU
  Product Name: Standard PC (Q35 + ICH9, 2009)
  Version: pc-q35-9.2
  Serial Number: Not Specified
  UUID: Not Settable
  Wake-up Type: Power Switch
  SKU Number: Not Specified
  Family: Not Specified
```

Figure 2: Screenshot d'une table DMI/SMBIOS classique sur QEMU  
Réalisé par: DOS SANTOS Ricardo

### III Parades

Il est possible de spécifier les valeurs à exposer au guest OS via le format suivant :[14]

```
qemu-system-x86_64 [...] -smbios type=<type>,field=<value>[,  
...]
```

Nous pouvons alors remplacer chaque information comme dans l'exemple ci-dessous :

```
qemu-system-x86_64 [...] -smbios type=0,vendor=AMI [...]
```

Ci-dessus, la valeur du vendeur BIOS est changée par le string "AMI". Pour trouver toutes les informations des tables DMI/SMBIOS afin de les passer à QEMU, nous pouvons utiliser l'outil *dmidecode*.

Pour ce qui est de la table d'extension des caractéristiques byte 2, nous avons deux solutions possibles pour contrer le 4ème bit de la table :

- Sans aucune option BIOS passée à QEMU, l'implémentation par défaut du *legacy BIOS* ne met pas ce bit à 1.
- Si nous souhaitons utiliser UEFI, alors nous devons passer un fichier de BIOS à QEMU. Cependant, aucune option ne permet explicitement de ne pas mettre ce bit à 1.

Alors il est possible de modifier le code source de QEMU dans le fichier *hw/smbios/smbios.c*. Dans ce fichier, nous retrouvons le code suivant pour la construction de la table SMBIOS (type 0) :

Dans la fonction *smbios\_build\_type\_0\_table*, nous pouvons retrouver la valeur suivante :

```
t->bios_characteristics_extension_bytes[1] = 0x14;
```

Nous pouvons manuellement changer la valeur à *0x04* (ou *0x0F*<sup>[9]</sup>) afin que la virtualisation ne soit plus annoncée par SMBIOS (que ce soit UEFI ou Legacy BIOS).

Voici la même information que la figure 2 après parade:

```
Handle 0x0000, DMI type 0, 24 bytes
BIOS Information
    Vendor: AMI
    Version: F.45
    Release Date: 10/29/2024
    Address: 0xE8000
    Runtime Size: 96 kB
    ROM Size: 64 kB
    Characteristics:
        BIOS boot specification is supported
        Function key-initiated network boot is supported
        Targeted content distribution is supported
        UEFI is supported
    BIOS Revision: 15.45
```

Figure 3: Screenshot de la table DMI/SMBIOS après modification de QEMU  
Réalisé par DOS SANTOS Ricardo

## C CPUID (x86\_64)

### I Explication

CPUID est une instruction qui permet de recevoir les informations des CPUs x86[15].

Ceci est fait en appelant la fonction assembleur *CPUID* en utilisant les registres *eax*, *ebx*, *ecx* et *edx*.

La valeur contenue dans *eax* (et dans certains cas, *ecx* aussi), sert de paramètre pour savoir quelle catégorie d'information nous souhaitons recevoir, appelée communément *leaf*.

Il y a deux types d'informations : *basic function* et *extended function*. [15]  
Les *basic functions* acceptent les valeurs *eax* suivantes :

- *0x0* à *0xB*, *0xD* et *0xF*
- *0x10*, *0x12*, de *0x14* à *0x1A* et *0x1F*

Les *extended functions*, quant à elles, acceptent les valeurs *eax* allant de *0x80000000* à *0x80000008*.

Chacune de ces *leaves* donne des informations relatives aux capacités des CPUs x86. Les valeurs et les explications de ces dernières sont documentées dans les manuels CPU Intel[15] et AMD[16].



En temps normal, les valeurs  $0x40000000$  à  $0xFFFFFFFF$  ne sont pas considérées comme implémentées.[15]

Les hyperviseurs utilisent les *leaves*  $0x40000000$  à  $0x400000FF$  afin de transmettre des informations sur l'hyperviseur au guest OS via CPUID.[16] Nous retrouvons notamment les informations suivantes[17] :

- La signature de l'hyperviseur (ici, QEMU/KVM) retournée dans les registres *ebx* à *edx*, avec *eax*= $0x40000000$
- Les *features* de KVM, avec *eax* =  $0x40000001$

## II Méthodes de détection

Une première méthode de détection de virtualisation est de vérifier l'output des registres *ebx*, *ecx* et *edx* de la *leaf* 0.

Ces registres contiennent le *vendor\_id*[15], qui est modifié par les hyperviseurs avec leur ID.

Une deuxième méthode est de vérifier le bit 31 du registre *ecx*, qui correspond à la présence d'un hyperviseur dans la *leaf*  $0x1$ .[18].

En temps normal, ce bit n'est pas utilisé par les CPUs physiques (bit toujours à 0)[15], AMD le documente comme réservé, étant utilisé par les hyperviseurs pour annoncer leur présence dans les CPUs virtuels.

Une troisième méthode est de regarder les données contenues dans la *leaf*  $0x40000000$ , utilisée pour retourner la signature de KVM dans les registres *ebx* à *edx*.

## III Parades

Afin d'avoir un *vendor\_id* qui ne sera pas détecté comme virtualisé, il suffit de passer le CPU en mode **host-passthrough** avec l'argument QEMU suivant :

```
qemu-system-x86_64 [...] -cpu host [...]
```

Pour contrer la détection du bit 31 du registre *ecx*, *leaf*  $0x1$ , nous pouvons

spécifier à QEMU/KVM de ne pas mettre ce bit à 1 en désactivant la *feature* CPU :

```
qemu-system-x86_64 [...] -cpu host,hypervisor=off [...]
```

La *feature* **hypervisor=off** fait que KVM ne s'annoncera plus à travers le bit 31 du registre *ecx*.

Pour parer aux *leaves* *0x40000000-0x400000FF*, nous pouvons rajouter **kvm=off** :

```
qemu-system-x86_64 [...] -cpu host,hypervisor=off,kvm=off [...]
```

Ceci désactive la signature retournée par KVM dans la *leaf* *0x40000000* et met les valeurs de ces *leaves* à 0.

Il ne faut pas confondre l'activation de KVM (via l'option **-enable-kvm**) et la suppression de sa signature dans CPUID (**-cpu host,kvm=off**) qui sont deux options **totalelement différentes**.

## D Température

### I Explication

Dans un système physique, les composants chauffent lorsqu'un système tourne. Il y a un besoin de logger ces températures afin de pouvoir prendre des mesures protectrices si un système surchauffe.

Dans un système virtualisé, il n'y a pas de capteur de température, donc le guest OS ne charge pas les fichiers en lien avec la température.

### II Méthode de détection

Certains outils vérifient l'existence d'un capteur de température ou des chemins créés par leur présence (*/sys/class/thermal/thermal\_zoneX*).

Une méthode de détection est donc la vérification des chemins, ou simplement si un capteur existe.

### III Parades

QEMU ne propose que des capteurs I2C, n'ajoutant pas de *thermal\_zone* dans */sys/class/thermal/thermal\_zoneX* ou étant découvert par les autres outils de détection.

Dans ce cas, il faut écrire un code en C afin de rajouter un composant à QEMU qui gère les températures, reconnu par sysfs par les différents outils.

## E Taille de Disque

### I Explication

Dans les ordinateurs physiques modernes, la taille minimale des disques est de 500 Go.

### II Méthode de détection

Selon les programmes, si la taille du disque et la taille d'espace restant dans le disque sont trop petites, il est possible d'inférer une virtualisation.

### III Parade

Avec l'outil *qemu-img*, il est possible de créer des images de disques, il suffit d'en allouer assez (selon la machine et ses besoins).

Une valeur recommandée généralement est 128 GiB.

Un exemple serait le suivant :

```
qemu-img create --format raw windobe.img +128G
```

## F Voltage

### I Explication

Les machines physiques utilisent de l'électricité pour fonctionner. Ceci implique forcément un voltage, qui correspond à la force d'un courant[\[19\]](#).

### II Méthode de détection

Dans une machine virtuelle, la notion de voltage n'est pas nécessaire, au même titre que la température.

Tout comme la température, il faut ajouter un composant à QEMU, écrit en C selon les structures données afin de pouvoir émuler un voltage.

## G Hostname + Username

### I Explication

Un hostname est un nom unique dans un réseau local donné, et le nom d'utilisateur est un nom unique au sein d'un système.

Cela permet d'identifier une machine sur un réseau et un utilisateur respectivement.

### II Méthode de détection

Si ces noms sont très explicites (ex: QEMU-user1, qemuMachine, etc.), alors un programme peut analyser ces derniers et tenter de détecter la virtualisation d'une machine.

### III Parades

Il suffit de ne pas utiliser de noms d'utilisateurs et des hostnames qui n'ont aucun nom de VM ou des noms trop génériques (ex: Utilisateur "user").

## H Ventilateurs CPU

### I Explication

Comme les besoins de vérifier les températures, et la notion de surchauffe n'existent pas dans un environnement virtualisé, le besoin d'avoir des ventilateurs de CPU/GPU ou autre n'existe pas non plus.

### II Méthode de détection

Un CPU physique requiert un ventilateur ou un moyen de refroidissement.

L'absence de tels moyens permet d'affirmer fortement que l'OS est virtualisé.

## I RDTSC force VM exit

### I Explication

RDTSC est une instruction qu'un programme peut utiliser pour faire un *benchmark* des temps de calculs de CPU.

Dans un environnement virtualisé, ces temps sont potentiellement très lents

pour les standards modernes.

Depuis un OS guest, il est possible de faire appel à des instructions afin de forcer un *VM\_exit*.

Un changement de contexte (donc un *VM\_exit*) est très coûteux pour un CPU, et demande un certain temps pour changer de contexte (passer de CPU non-root à root).

## II Méthode de détection

Dû au coût d'un changement de contexte, utiliser l'instruction *RDTSC* couplée à une action qui garantit un *VM\_exit*, alors les résultats de l'instruction peuvent inférer la virtualisation d'un système.

## III Parade

Dans le [projet git](#), le Kernel peut être modifié afin de contrôler la sortie de l'instruction *RDTSC*.

Il est proposé de récupérer la valeur *RDTSC* de l'hôte et de construire une fausse valeur dans les registres des CPUs virtuels afin de tromper le *benchmark*.

Cela requiert la recompilation du kernel.

## J Adresse MAC

### I Explication

Une adresse MAC est un identifiant unique dans un réseau local qui permet d'identifier les interfaces de la carte réseau d'une machine.

### II Méthode de détection

De manière générale, certains hyperviseurs utilisent des préfixes connus pour leurs adresses MAC.

La présence de ces préfixes permet d'affirmer la virtualisation via l'hyperviseur utilisant ce préfixe.

### III Parades

Il est possible dans QEMU de spécifier l'adresse MAC des interfaces réseau de la VM.

Pour ce faire, voici un exemple de commande suivante:

```
qemu-system-x86_64 [...] -device e1000,mac=AA:BB:CC:DD:EE:FF
```

Ci-dessus, le modèle de carte réseau n'est qu'un exemple (e1000).

## K Tables ACPI

### I Explication

Advanced Configuration and Power Interface (ACPI) est un *framework* développé dans le but d'être une interface de gestion de l'alimentation d'une machine ainsi qu'un *framework* de configuration en tant que sous-système dans la machine hôte.[\[20\]](#)

En plus de cela, l'ACPI résout un problème de dépendance d'architecture.[\[20\]](#)

### II Méthode de détection

Dans les tables de l'ACPI, les composants qui forment les tables ont des noms pour les identifier.

Ces derniers sont générés par l'hyperviseur et contiennent le nom de l'hyperviseur (Ex: QEMU002).

### III Parades

Nous pouvons utiliser le patch de QEMU, qui change au niveau du code source les noms générés pour les tables ACPI.

Un exemple de changement apporté au niveau du code source via un git diff du fichier `hw/i386/fw_cfg.c`:

```
-    aml_append(dev, aml_name_decl("_HID",  
    aml_string("QEMU0002")));  
+    aml_append(dev, aml_name_decl("_HID",  
    aml_string("UEFI0002")));
```

## L Dmesg

### I Explication

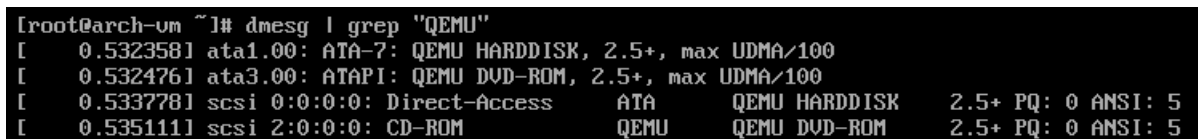
Cette méthode est propre à Linux.

*Dmesg* est un outil qui imprime et contrôle le *kernel ring buffer*. En d'autres termes, *dmesg* permet de lire les messages au niveau kernel ainsi que de contrôler le tampon dans lequel ces messages s'installent.

### II Méthode de détection

Des informations au niveau kernel sur les composants peuvent apparaître dans les résultats de la commande *dmesg*.

Voici un exemple dans la figure 2:



```
[root@arch-vm ~]# dmesg | grep "QEMU"
[ 0.532358] ata1.00: ATA-7: QEMU HARDDISK, 2.5+, max UDMA/100
[ 0.532476] ata3.00: ATAPI: QEMU DVD-ROM, 2.5+, max UDMA/100
[ 0.533778] scsi 0:0:0:0: Direct-Access ATA QEMU HARDDISK 2.5+ PQ: 0 ANSI: 5
[ 0.535111] scsi 2:0:0:0: CD-ROM QEMU QEMU DVD-ROM 2.5+ PQ: 0 ANSI: 5
```

Figure 4: Screenshot de la VM sur Arch Linux, exécutant la commande *dmesg*  
Réalisé par DOS SANTOS Ricardo

### III Parades

Changer les noms des composants dans QEMU permet d'éviter que des noms d'hyperviseur n'apparaissent dans *dmesg*.

À nouveau, l'excellent travail de [Scrut1ny](#) permet de le faire via un patch utilisant *git diff*.

## V Création d'une VM et exécution des techniques d'évasions

### A Pré-requis

Pour pouvoir répliquer ce projet, il faut avoir activé dans les options BIOS Intel-VT<sub>x</sub> ou AMD-V.

Il est possible de vérifier si le CPU a la capacité activée via la commande suivante[3]:

```
lscpu | grep Flags | grep "vmx\|svm"
```

*Vmx* ou *svm* devrait apparaître dans la liste des *flags* CPU.

Ceci est requis pour pouvoir utiliser KVM et faire de l'*hardware-assisted emulation*.

Selon les distributions Linux, il faut aussi que l'utilisateur appartienne au groupe KVM. Si le module n'est pas chargé, il faut également le faire[3]:

```
sudo modprobe kvm
```

## I Paquets à installer

- Debian:

```
sudo apt install uuid-dev iasl git gcc-5 nasm  
python3-distutils qemu-utils dmidecode
```

- Arch-Linux:

```
sudo pacman -S base-devel patch git qemu-img subversion  
glibc iasl dmidecode
```

## II Patching de QEMU

Afin de patcher QEMU, il faut tout d'abord cloner [cloner le repo git](#).

Ensuite, il suffit d'être dans le dossier du repo git QEMU et d'exécuter la commande suivante:

```
patch -fsp1 < qemu.patch
```

Où *qemu.patch* est le nom du fichier utilisé et fourni dans le [git du projet](#). À noter que le patch reprend celui de [Scrut1ny](#) avec des modifications supplémentaires.

Pour *build QEMU*, utilisez la commande suivante:

```
./configure && make -j8
```

Où l'option *-j8* est une option qui permet de faire tourner plusieurs jobs simultanément. En d'autres termes, cela permet de paralléliser la commande *make*.

Les binaires QEMU se trouveront dans le dossier *build* du repo.



### III Patching de EDKII

Les instructions pour *EDKII* sont similaires:

- Cloner le [repo](#) EDKII
- Utiliser le patch du via la commande:

```
patch -fspi < edkii.patch
```

- *Build EDKII* avec:

```
make -C BaseTools  
cd OvmfPkg && ./build.sh
```

Le fichier *OVMF.fd* se trouvera dans *Build/OvmfX64/DEBUG\_GCC5/FV/OVMF.fd*

### B Mise en place des différentes parades

Pour commencer, il faut choisir notre binaire de QEMU patché.

Ensuite, nous pouvons appliquer les options QEMU pour chaque parade:

- **KVM:** *-enable-kvm*.  
Comme son nom l'indique, cela nous permet d'activer l'utilisation de KVM par QEMU.
- **Carte Mère:** *-device e1000e,mac=AA:11:BB:22:CC:33*.  
La valeur de l'adresse MAC peut être générée aléatoirement au préalable.
- **BIOS:** *-drive if=pflash,format=raw,file=OVMF\_patched.fd*.  
*OVMF\_patched.fd* est le nom du binaire patché de EDKII.
- **CPU:** *-cpu host,hypervisor=off,kvm=off -smp 6,sockets=1,threads=1,cores=6*.  
Ceci permet de donner 6 vCPU à la VM, où 6 est un choix arbitraire.  
Il est possible d'augmenter le nombre de cœurs attribués à la VM.

Pour ce qui est du **SMBIOS**, il faut donner tous les types et valeurs des tables disponibles via le format d'option:

```
qemu-system-x86_64 -smbios type=x,field=y
```

Dû à la longueur des options à donner, voici un extrait d'un script bash qui les récupère automatiquement via *dmidecode*:

```
dmi_get ()
{
    echo \$(sudo dmidecode -t $1 | grep -e ".$$2.*:" | cut -d ":"
        -f2 | xargs)
}

declare -a SMBIOS=(
-smbios type=0,vendor="\$(dmi_get 0 Vendor)",date="\$(dmi_get 0
    Date)",version="\$(dmi_get 0 Version)",release="\$(dmi_get 0
    BIOS)",uefi=on
-smbios type=1,manufacturer="\$(dmi_get 1
    Manufacturer)",product="\$(dmi_get 1
    Product)",version="\$(dmi_get 1 Version)",serial="\$(dmi_get 1
    Serial)",uuid="\$(dmi_get 1 UUID)",sku="\$(dmi_get 1
    SKU)",family="\$(dmi_get 1 Family)"
-smbios type=2,manufacturer="\$(dmi_get 2
    Manufacturer)",product="\$(dmi_get 2
    Product)",version="\$(dmi_get 2 Version)",serial="\$(dmi_get 2
    Serial)",asset="\$(dmi_get 2 Asset)",location="\$(dmi_get 2
    Location)"
-smbios type=3,manufacturer="\$(dmi_get 3
    Manufacturer)",version="\$(dmi_get 3
    Version)",serial="\$(dmi_get 3 Serial)",asset="\$(dmi_get 3
    Asset)",sku="\$(dmi_get 3 SKU)"
-smbios type=4,sock_pfx="\$(dmi_get 4
    Sock)",manufacturer="\$(dmi_get 4
    Manufacturer)",version="\$(dmi_get 4
    Version)",serial="\$(dmi_get 4 Serial)",asset="\$(dmi_get 4
    Asset)",part="\$(dmi_get 4 Part)"
-smbios type=11,value="\$(dmi_get 11 1)",value="\$(dmi_get 11
    2)",value="\$(dmi_get 11 3)",value="\$(dmi_get 11
    4)",value="\$(dmi_get 11 5)",value="\$(dmi_get 11 6)"
-smbios type=17,loc_pfx="\$(dmi_get 17 Locator | cut -d ' '
    -f1-5)",bank="\$(dmi_get 17 Locator | cut -d ' '
    -f1-2)",manufacturer="\$(dmi_get 17 Manufacturer | cut -d ' '
    -f1)",serial="\$(dmi_get 17 Serial | cut -d ' '
    -f1)",asset="\$(dmi_get 17 Asset | cut -d ' '
    -f1)",part="\$(dmi_get 17 Part | cut -d ' '
    -f1)",speed="\$(dmi_get 17 Speed | cut -d ' ' -f1)
)
```

Il suffit ensuite d'appeler le tableau de la manière suivante:

```
qemu-system-x86_64 "${SMBIOS[@]}"
```

Le reste des options sont recommandées, et les options de mon choix lors de création de VMs (elles sont donc arbitraires):

- *-M q35*.  
Passer le type de machine à *q35*.
- *-m 12G*.  
Spécification de la mémoire RAM à QEMU.
- *-b menu=on*.  
Cette option permet d'entrer dans la configuration BIOS de la VM avant que cette dernière n'exécute les options de démarrage.

En [git du projet](#), il y aura un script qui automatise l'exécution de la VM. Ce dernier est celui qui est utilisé lorsque la VM est exécutée dans ce document.

## VI Résultats & discussion

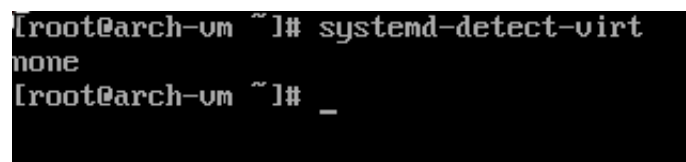
### A Systemd-detect-virt

#### I Résultats

*Systemd-detect-virt* est le premier programme testé, car ce dernier était simple mais présentait des techniques de détection qui se retrouvent dans tous les autres outils.

Avec les options données, il est possible d'esquiver toute détection employée par le programme. Ce dernier ne fait plus la différence entre une machine physique réelle et la VM.

Voici la figure 5 des résultats:



```
[root@arch-vm ~]# systemd-detect-virt
none
[root@arch-vm ~]# _
```

Figure 5: Screenshot des résultats de Systemd-detect-virt  
Réalisé par DOS SANTOS Ricardo

## II Discussions

Lors de la phase d'étude du programme et de CPUID, il y a eu un moment de confusion sur ce qui était vérifié par *systemd*, notamment par rapport au 31<sup>e</sup> bit du registre *ecx*, *leaf 0x1*.

En effet, les manuels Intel indiquaient ce bit comme étant toujours mis à 0. Heureusement, les manuels AMD ont fini par éclaircir les points qui n'étaient pas mentionnés par Intel.

Après avoir compris le mécanisme de CPUID, et comment les hyperviseurs utilisaient les espaces non alloués par les CPUs physiques, les vérifications des tables SMBIOS furent un nouveau blocage.

QEMU ne proposant pas de désactivation du quatrième bit via des options, il a fallu appliquer un premier patch de QEMU pour pouvoir contrer chaque technique de détection employée.

## B VMAware

### I Résultat

VMAware est un programme cross-plateform beaucoup plus complet que les autres outils utilisés.

Les tests ont été faits principalement sur Linux, mais aussi sur Windows.

Les techniques de détection qui sont employées par VMAware réussissent à 99%.

Il n'existe qu'une seule technique qui n'a pas été contournée pour ce programme : la détection d'un capteur de température.

L'impossibilité de faire monter un composant en */sys/class/thermal* avec ceux proposés par QEMU est le seul obstacle à franchir pour contourner VMAware. Voici la figure 6 des résultats de VMAware:

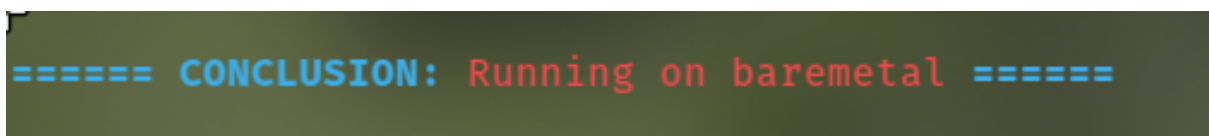


Figure 6: Screenshot des résultats de VMAware  
Réalisé par DOS SANTOS Ricardo

## II Discussion

Pouvoir monter un capteur thermique via I2C semblait être une bonne avancée, mais malgré la détection du capteur, ce dernier ne crée pas de dossier dans `/sys/class/thermal`.

La seule solution est d'écrire un nouveau composant et de l'ajouter à QEMU, puis de le recompiler. Malheureusement, il n'était pas possible, avec les contraintes de temps, de pouvoir le faire, rendant le programme encore à contourner à 100%.

## C Al-khaser

### I Résultats

Al-khaser est un outil d'analyse d'environnement *malware*. Ce programme ne peut être exécuté que sur Windows.

Les tests de ce programme sont divisés en deux sections pour les VMs:

- Des tests de détection de VMs génériques. Ces tests ne tentent pas de détecter un hyperviseur spécifique.
- Des tests spécifiques à des hyperviseurs. Ici, ce qui nous intéresse sont ceux pour QEMU/KVM.

Pour ce qui est des tests génériques, nous avons des résultats à hauteur de 70% de réussite.

La majorité des tests qui ne réussissent pas sont liés à l'absence d'éléments physiques comme le voltage et la température.

D'autres tests sont liés à SMBIOS:

Si ceux sur la mémoire de SMBIOS passent, une vérification du nombre de tables ne passe pas.

Cependant, chaque test spécifique à QEMU passe. Les résultats ne sont pas parfaits, mais ils restent significatifs.

Voici la figure 7 des résultats pour QEMU

```
-----[QEMU Detection]-----
[*] Checking reg key HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0 [ GOOD ]
[*] Checking reg key HARDWARE\Description\System [ GOOD ]
[*] Checking qemu processes qemu-ga.exe [ GOOD ]
[*] Checking qemu processes vdagnt.exe [ GOOD ]
[*] Checking qemu processes vdservice.exe [ GOOD ]
[*] Checking QEMU directory C:\Program Files\qemu-ga [ GOOD ]
[*] Checking QEMU directory C:\Program Files\SPICE Guest Tools [ GOOD ]
[*] Checking SMBIOS firmware [ GOOD ]
[*] Checking ACPI tables [ GOOD ]
```

Figure 7: Screenshot des résultats de Al-khaser pour QEMU  
Réalisé par DOS SANTOS Ricardo

## II Discussion

Ce qu'il faut principalement retenir du point de vue d'une VM avec Al-khaser, c'est le besoin de rajouter des capteurs (de température ou de voltage) émulsés.

Le but serait d'avoir des composants, qui n'alourdissent pas l'hyperviseur mais qui permettent d'exposer des valeurs fictives, soit dans le code du composant, soit via une interface dans l'hôte (potentiellement lire les valeurs de l'hôte, comme un passthrough).

## D Pafish

### I Résultats

Les tests sur Pafish sont réussis à 99%.

Le seul test qui ne passe pas est causé par le fait qu'un patch n'est pas appliqué sur le kernel.

Ce document n'a pas appliqué le patch kernel, par manque de temps pour tester et utiliser un kernel modifié. Cependant, le patch est disponible et applicable via le projet [git de Scrut1ny](#).

## II Discussion

Au-delà du patch non appliqué sur le kernel, Pafish est un autre outil qui peine à détecter les VMs.

Cependant, à l'instar des autres outils de détection de virtualisation, ce dernier pousse l'utilisateur à exposer de fausses valeurs au niveau kernel afin de passer le test de détection.

Cela reste une méthode vue nulle part ailleurs.

## E Anti-cheats & Autres

Certains jeux vidéo ont des anti-cheats, des programmes qui tentent de protéger le programme (ici le jeu) de toute manipulation de mémoire ou du client de jeu en lui-même.

Certains de ces jeux refusent de tourner dans une VM, car ces derniers ne peuvent pas se protéger face au contrôle d'un hôte de l'environnement virtualisé, qui facilite la manipulation des données de l'OS invité.

Malheureusement, ces anti-cheats ne sont souvent pas open-source, donc des tests effectués avec des jeux sont difficiles à évaluer.

De manière générale, si nous prenons un jeu avec un anti-cheat ayant pour réputation de refuser une VM, ce dernier ne nous permettra pas de l'exécuter. Cependant, il est difficile de dire si cela est dû à un manque de performance ou si nous sommes détectés comme virtualisés.

## F Limitations de ce projet

Dans un premier temps, la première limitation du projet est QEMU lui-même.

Si QEMU est un excellent hyperviseur couplé à KVM, ce dernier n'implémente pas tous les détails qui permettent à une VM de se faire passer pour une machine physique. Cela est normal, un hyperviseur ne cherche pas à se faire passer pour une machine réelle, c'est généralement l'inverse.

## VII Conclusion

Le but de ce projet est de rendre une VM aussi indétectable que possible. À cette fin, j'ai utilisé QEMU couplé à KVM avec les diverses options de QEMU pour modifier ma VM, et la rendre aussi proche de ma machine physique que possible.

Dans ce but, il a fallu aussi modifier QEMU et EDKII pour l'image BIOS/UEFI, ce qui nous amène à plusieurs outils qui ont échoué à détecter la VM.

Les outils qui nous ont détectés présentaient des techniques de détection que QEMU/KVM ne pouvait pas gérer (manque de capteurs de température ou voltage), et qui n'ont pas pu être implémentées dans le temps imparti.

Si une VM indétectable n'a pas pu être un objectif atteint, cet exercice reste largement satisfaisant, car cela montre les capacités actuelles qui nous permettent d'atteindre un tel objectif, avec peu d'obstacles qui nous permettent vraiment de l'atteindre.

Malheureusement, les informations qui permettent de mettre tout cela en œuvre ne sont pas assez expliquées ou sont juste difficiles à trouver, amenant à un véritable défi sur comment rendre un environnement virtualisé comparable à un environnement physique.

Les performances de la VM restent un problème qui n'a pas été abordé dans ce document, cependant l'utilisation des connaissances acquises couplée à des outils comme libvirt pourrait pallier ce problème.



## Références documentaires

- [1] Redhat. *What is a hypervisor?* URL: <https://www.redhat.com/en/topics/virtualization/what-is-a-hypervisor>. Accessed: 26 March 2025.
- [2] Florent Glück. *Introduction to Virtualisation*. Feb. 25, 2024. Accessed: 10 December 2025.
- [3] Florent Glück. *Platform Virtualisation*. Feb. 25, 2024. Accessed: 10 December 2025.
- [4] *System and Service Manager*. URL: <https://systemd.io>. Accessed: 26 March 2025.
- [5] R. Jones. *Virt-what - detect if we are running in a virtual machine*. URL: <https://people.redhat.com/~rjones/virt-what/>. Accessed: 26 March 2025.
- [6] Ayoub Faouzi. *Al-Khaser*. URL: <https://github.com/ayoubfaouzi/al-khaser>. Accessed: 18 December 2024.
- [7] Alberto Ortega. *Pafish*. URL: <https://github.com/aOrtega/pafish>. Accessed: 18 December 2024.
- [8] *VMware*. URL: <https://github.com/kernelwernel/VMware>. Accessed: 18 December 2024.
- [9] Scrutiny. *Qemu patches*. URL: <https://github.com/Scrutiny/Hypervisor-Phantom/blob/main/Hypervisor-Phantom/patches/QEMU/intel-qemu-9.2.0.patch>. Accessed: 15 Février 2025.
- [10] *Desktop Management Interface*. URL: <https://www.dmtf.org/standards/dmi>. Accessed: 27 November 2024.
- [11] *DMTF announces End of Life for DMI*. DMTF. URL: <https://www.dmtf.org/node/22>. Accessed: 18 December 2024.
- [12] *SMBIOS Specification*. Version 3.8.0. DMTF. Aug. 5, 2024. URL: [https://www.dmtf.org/sites/default/files/standards/documents/DSP0134\\_3.8.0.pdf](https://www.dmtf.org/sites/default/files/standards/documents/DSP0134_3.8.0.pdf). Accessed: 18 December 2024.
- [13] *Source code comments in /src/basic/virt.c*. Systemd. Accessed: 23.10.2024.
- [14] *Mappings for DMI/SMBIOS to Linux and dmidecode*. URL: <https://gist.github.com/smouser/290f74c256c89cb3f3bd434a27b9f64c>. Accessed: 10 December 2024.
- [15] *Intel® 64 and IA-32 Architectures Software Developer's Manual*. Nov. 2020. Accessed: 27 November 2024.
- [16] *AMD64 Architecture Programmer's Manual Volume 3: General-Purpose and System Instructions*. Mar. 2024. Accessed: 10 December 2024.
- [17] *Documentation of the Linux Kernel Source Code*. Accessed: 10 December 2024.
- [18] *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Mar. 2024. Accessed: 27 November 2024.
- [19] Cambridge University. *Voltage*. URL: <https://dictionary.cambridge.org/dictionary/english/voltage>. Accessed: 26 March 2025.
- [20] *Advanced Configuration and Power Interface (ACPI) Introduction and Overview*. Apr. 26, 2016. Accessed: 26 March 2025.