Universiteit van Amsterdam

Research Project 1

Hyper-V Harmony: Exploring Cross-VM Communication (IPC)

Authors
Yassir Laâouissi
(yassir.laaouissi@os3.nl)
Vlad Arsene
(vlad.arsene@os3.nl)

Supervisor
Kyrian Maat
(k.maat@uva.nl)

April 10, 2024

Abstract

This study delves into the isolation provided by mechanisms such as Inter-Partition Communication (IPC) in Hyper-V, a virtualization technology within the Microsoft Windows realm. The main objective is to explore how IPC is used within a Hyper-V architecture to determine the possibilities for compromising the isolation layer of machines. The research examines the aspects of Hyper-V architecture, categorizes cross-VM IPC mechanisms, and investigates how these communication channels can be exploited to breach virtual machine isolation. One of the results of this research is the ability to send arbitrary messages over IPC using partially undocumented structures in a storage driver called SCSI Request Block (SRB). The findings enhance virtualization security by highlighting risks and providing insights, on how to safeguard Hyper-V environments against emerging threats. Furthermore, this research concludes the ability to fuzz previously undocumented structures used in Hyper-V.

Contents

1 Introduction

In the domain of computing, solutions like Hyper-V have spearheaded a revolution, characterized by enhancements in efficiency and regulation of resources through the use of virtual machines. On the flip side of these prevalent technological strides, however, there is a need to examine the potential for security breaches [7, 6], particularly in functions such as Inter-Partition Communication (IPC). This study entails an in-depth analysis of the use of this functionality within the Hyper-V platform used for virtualization. At the heart of this evaluation, the goal is to map how communication channels, such as VMBus, might serve as pathways to break through the isolation layers that form the relevancy of a virtual machine, thereby compromising its security.

1.1 Research questions

The primary objective of this research is to establish a deconstruction of specific IPC workings used by Hyper-V. For this objective to be completed, the following research questions have been established:

Main question: What Cross-VM/Inter-Partition Communication is present in between Hyper-V guest virtual machines, and can be leveraged for breaching the virtual machine isolation layers?

Sub questions

- How does the Hyper-V architecture function at a foundational level?
  - To what extent does the Hyper-V design facilitate communication between virtual machines?
- Which Cross-VM IPC are present in Hyper-V?
  - How can hypercalls passing through VMBus and hypervisor be mapped?
- How can the found IPC be exploited?

With the myriad of challenges virtual platforms such as Hyper-V face, this study eventually stands as a reminder to enlist strong security measures. This is established in this study by addressing channel-based communication between partitions in between Hyper-V guest virtual machines, and its corresponding potential to breach isolation.

2 Related work

Previous research into Hyper-V and the usage of IPC reveals limited insight into the closed-source nature of the infrastructure in Hyper-V. Even if the underlying mechanism is constantly being changed. The information derived from

old sources is still of value for making hypotheses during research, and thus can potentially lead to renewed insights.

Inter-Process Communication represents the collection of techniques for data sharing among different processes. It can be accomplished by either memory sharing or message passing [4]. In Cross-VM IPC, which in Hyper-V is also called Inter-Partition Communication [12], the memory spaces of each IPC session are isolated between the sender and receiver.

Furthermore, in a virtualized environment, the communication procedures can be divided into VM-Host and VM-VM IPCs [9]. The former is used in any OS-level virtualization technology (such as Hyper-V) and allows access to different system services and resources. The latter aims to facilitate the interaction between applications on different machines while preserving most of the inherent isolation between them.

To ensure isolation, Hyper-V [13] hosts operating systems inside different logical units called partitions. Due to security, performance, and compatibility reasons, three types of partitions are supported [10] by the hypervisor: root, enlightened, and unenlightened. The root partition runs the Hyper-V Management Services and allows access to the communication mechanism used by the host and virtual machines. The other partitions are used as regular virtual machines, with the enlightened ones being virtualization-aware. This means they can access the data bus directly, thus bypassing the entire device emulation layer.

The primary mechanism for discovering security and reliability issues in modern software is fuzzing [1]. A fuzzer would execute a program with randomly generated input and adapt that input in an attempt to force an invalid operation. AFL++ [3] or WinALF (the Windows-adapted variant) can take advantage of the state-of-the-art fuzzing techniques to converge to a potential error-inducing input. Different flaws in the implementation of the para-virtualized drivers used by Hyper-V [5, 8], have been found using such approaches. Therefore there is a great interest in their further research and development.

3 Method

The goal of this study is to map the functionality of Hyper-V's Inter-Partition Communication and explore the potential for breaching the isolation layer through the technologies that Inter-Partition Communication serves. To do this, it is required to debug components of Hyper-V, to see their respective behavior in real-time. However, analyzing and debugging hypervisor components is a meticulous process and is prone to regular system crashes. Therefore the methods and environments used to research a hypervisor, such as Hyper-V, need to be thoroughly thought through. In this chapter, the lab environment and planning of conducted experiments will be discussed.

3.1 Lab setup

The lab environment consists of a couple of elements, as illustrated in Figure 1.

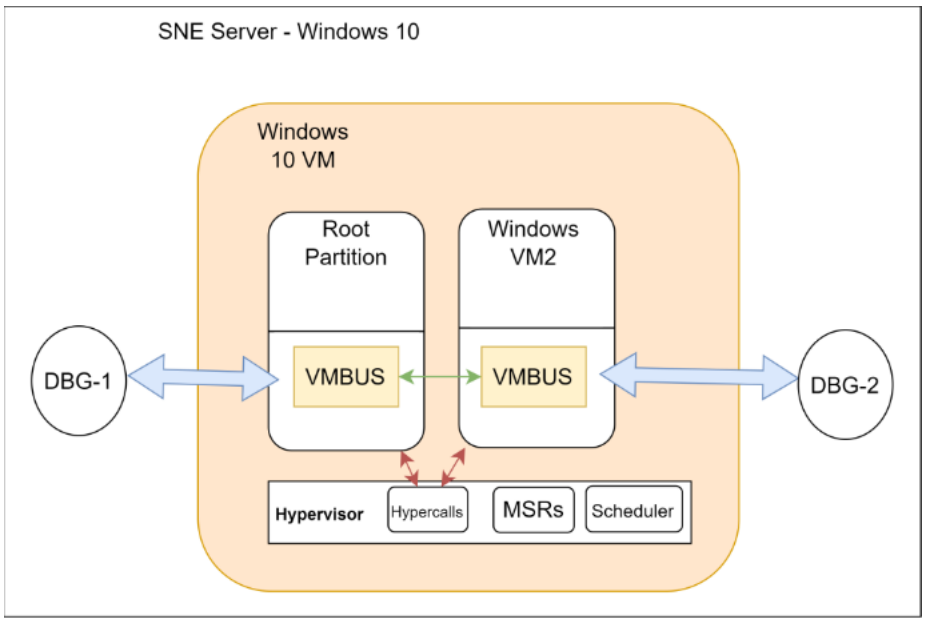

Figure 1: Visualisation of the isolated VM architecture

The first element is the Windows 10 server. This is a Dell Server, where Windows 10 has been flashed directly on the hardware through the Dell Remote Access Controller (DRAC). On this Windows 10 host, later referred to as Layer-0, a virtual machine was made with Hyper-V. This Virtual Machine, also known as Layer-1, can not be directly debugged for breaking the isolation layer using IPC. Despite Layer-1 being a Hyper-V-based VM, the risk of debugging layer-1 is crashing the root partition of the Layer-1 VM, which is situated on the Layer-0

machine. To recover the root partition to a functioning state, rebooting Layer-0 is required. Restarting an entire Dell server takes more time than restarting a Hyper-V VM.

Therefore a third layer, called Layer-2 was introduced. This layer is a nested VM (Layer-2 inside Layer-1). Layer-2 is the actual VM and root partition that have been researched. A series of debuggers have been attached to the kernel of both the Layer-2 root partition and the kernel of the Layer-2 guest partition (Windows 10 virtual machine). The debugging sessions were established over the network by modifying two entries in the Boot Configuration Data (BCD) store for each partition (debug on boot and network debugging).

3.2 Experiments

Using the aforementioned lab environment, a series of experiments have been conducted to uncover the answers to the research questions. Below a brief description of the experiments and corresponding literature can be found.

As will later be described in section 4.1 of this paper, Hyper-V partitions communicate through either a mechanism called synthetic interrupts, or through channel-based communication, such as VMbus. In 4.2 we described the limited use of synthetic interrupts. Therefore we sided with the more often used form of inter-partition communication in Hyper-V called VMBus. This mechanism utilizes a module called the Virtualization Service Provider/Client in combination with VMBus channels to communicate between respective partitions about resource allocation concerning storage, video, PCI, and networking[10].

To utilize this same mechanism in later experiments, one has to have an understanding of the layout of the channel and the messages being sent over it. This experiment was dedicated towards mapping the layout of this channel, and the traffic going over the channel. In doing so, a couple of methods were used:

- **Hyper-V Sockets**: The Windows 10 Anniversary Update introduced a new socket address family [2] made specifically for enabling communication between the Hyper-V host and its guests. It creates a VMBus channel through which user-mode applications can send arbitrary data. A connection between the host and guest could not be established, thus this approach was completely disregarded.
- **Creating custom VMBus channel**: A second attempt was to create a custom VMBus channel, and send messages through it by reversing the libraries responsible for sending messages and replicating the steps. However, this idea was promptly abandoned as it is impossible to create a VMBus channel from a guest. The `vmbuskernelmodeclientlibapi.h` header explains that it can only be created from the root partition and therefore defies the whole purpose of setting up such a channel to break isolation.

- **Hijacking an existing VMBus channel**: The last, and most successful, approach was to utilize an already existing VMBus channel between the guest and root in Layer-2 through a custom driver. This implies determining the structure of the messages sent over the VMBus channel.

Once a functioning VMBus channel can be utilized, sending an arbitrary message is the logical next step to take to start fuzzing and potentially break the isolation. This is done so that in a later stage fuzzing can be made possible from the guest towards the root partition.

4 Results

4.1 Hyper-V architecture

Hyper-V is a hypervisor, most commonly known as a translation layer between hardware and virtual machines as illustrated in high level as can be seen in figure 2. Note that both the hypervisor and virtual machines consist of multiple mechanisms that have been omitted from the figure due to irrelevancy at this stage.

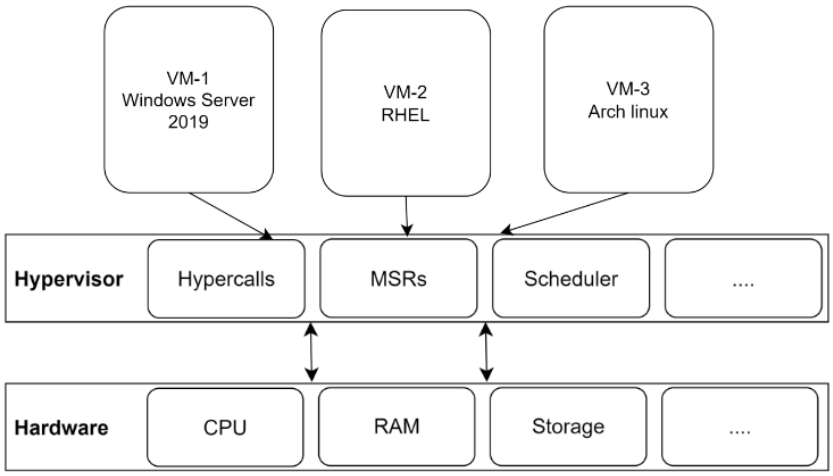

Figure 2: High-level visualization of Hyper-V

However, in reality, each virtual machine lies within a so-called partition. All of these VM partitions are orchestrated by a root partition. Microsoft describes the root partition to be an orchestration partition for all other partitions. It is the only partition with direct hardware access, and therefore crucial for guest

partitions to operate. However, when no guest partitions are present, the root partition is running. Furthermore, the root partition is started when Hyper-V is started, which under default conditions happens during the boot-time of the guest OS. The root partition was found to be running due to the modules "winhvr.sys", "vmbkmclr.sys" and "vmbusr.sys" being loaded without a guest being present.
They communicate through VMBus and synthetic interrupts. An illustration in figure 3 has been made based on Microsoft's vendor documentation [13]. This illustration has been reduced to the relevant elements of their documentation. The red arrows indicate synthetic interrupts, and the green arrows indicate VMBus channels. We refer the reader to [13] for more information on the aforementioned out-of-scope elements of the hypervisor and partitions.

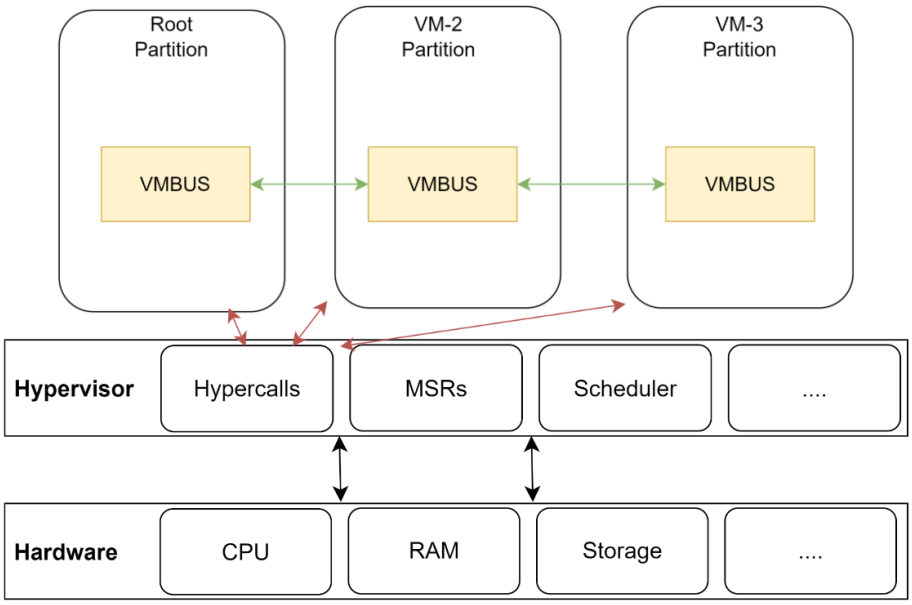

Figure 3: Low-level visualization of Hyper-V

4.2 IPC Flavours in Hyper-V

As briefly discussed earlier, there are two types of IPC present in Hyper-V:

- **Synthetic interrupts**: Synthetic interrupts [11] are IPCs that have to go through the hypervisor to reach other guests. This type of IPC is used for establishing a VMBus channel for example.
- **VMBus**: This type of IPC goes directly from partition to partition, without an intermediary such as a hypervisor. This software construct utilizes

functionality from native kernel drivers to communicate storage, network, visual, and PCI-related resource usage [10].

The choice of exploring the VMBus channel was made due to a couple of reasons:

1. There is no intermediary, hence debugging, reversing, and translating findings served less complexity.
2. The exact inner workings of storage-related Inter-Partition Communication over VMbus have not been described in depth publicly to our knowledge.

4.3 Controlling the VMBus

4.3.1 Talking to the root

The different attempts (as listed in section 3.2) to send arbitrary data to the root partition indicated that hijacking an already existing channel could prove to be the best approach for breaching the virtualization layer. This research focuses on the interplay between the `storVSP.sys` and `vmbkmcl.sys` drivers and attempts to take over the storage communication channel. This is achieved through a custom driver (shown in figure 4) that performs the following steps:

1. Determine the base address of `vmbkmcl.sys`
2. Find the `StorVSC` device
3. Determine the addresses of the packet manipulation functions (e.g. `VmbPacketSend()`)
4. Send arbitrary packets to the root partition
5. Create an I/O Queue to receive data from a user-mode application (used for fuzzing)

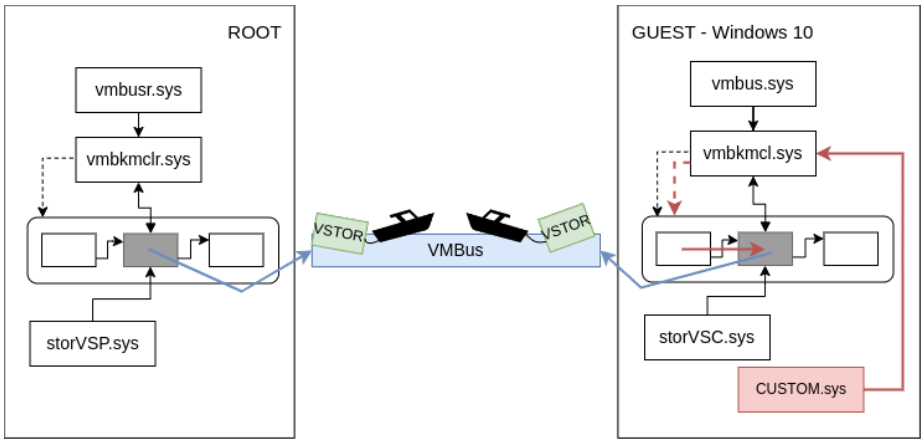

Figure 4: StorVSC channel communication

The driver uses the Mimikatz kernel modules to enumerate the installed drivers and find their base addresses. The storage channel can be found inside a linked list which is managed by the `vmbkmcl.sys` module. Since the `VMBCHANNEL` data structure is unknown, parsing such a list presents a significant challenge. The driver dynamically determines the top of the data structure since it starts with a pointer to self. A `VMBCHANNEL` also contains the GUID of the installed device. In the debugger, all communication channels, with their associated GUIDs can be listed and the offset to the respective field can be determined. Figure 5 shows the automatically determined fields in the data structure.

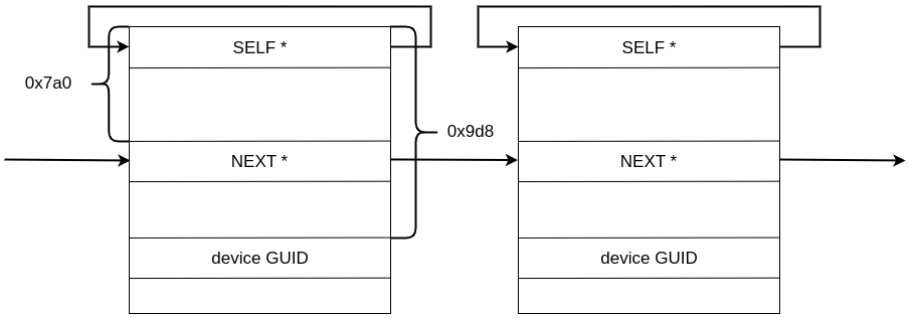

Figure 5: Illustration of VMBCHANNEL data structure

After determining the address of the storage channel, arbitrary SRB packets can be crafted and sent to the root partition via the `VmbPacketSend()` function which returns a `STATUS_SUCCESS` code.

The source code of the driver can be found in the appendix A.

4.3.2 Vulnerabilities, Exploitation, and Fuzzing

The custom driver built represents a fuzzing harness for the storage channel. With some additional components, specific packets can be crafted and an attempt to crash the root partition can be made.

To achieve this, some additional components (figure 6) are required. On an outside control machine, the main fuzzing process runs. A custom user mode executable will craft and send the packets to the hijacking driver via IOCTLs. Finally, on the root partition, a monitoring driver will wait for `storVSP.sys` to crash and send a memory dump back to the control machine once this happens.

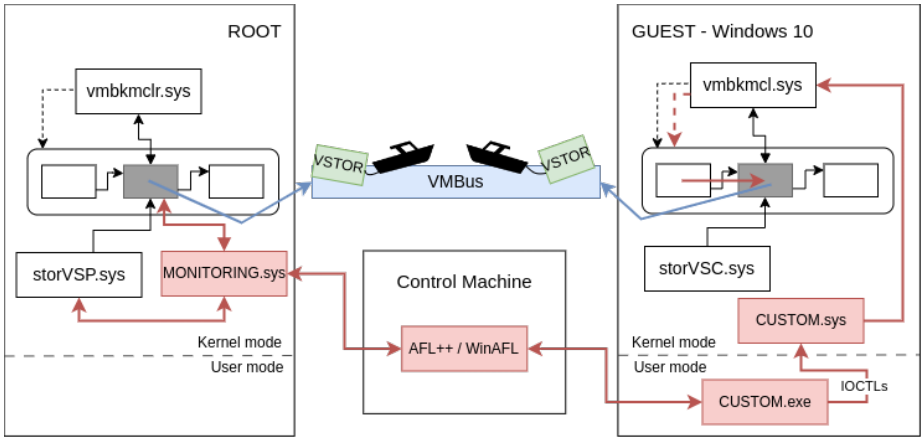

Figure 6: Fuzzing setup

If the root partition crashes or issues an error at any point, then a vulnerability might be present within the Hyper-V's infrastructure. A root partition crash is, in fact, a denial of service for all the guests running on top of the hypervisor.

5 Conclusion

5.1 Root partition

The root partition is present at all times when Hyper-V is running. Even if no guest partition is running. As far as this research goes, no documentation has been located indicating a reason for the root partition to be running, without a guest partition. Therefore one can conclude that the root partition is potentially creating a resource creep, by allocating more memory than required for

the functional operation of Hyper-V.

5.2 Hijacking existing VMBus channel

Attempts have been made to create IPC on demand. These attempts have failed, which prompted us to resort to hijacking the already present IPC. We can conclude that this has been deemed possible. Furthermore, this has enabled us to send arbitrary messages over VMBus through SCSI request blocks (SRBs). This can be concluded due to the client-side confirmation of the custom messages being sent successfully.

5.3 Fuzzing

A partial fuzzing harness has been set up, though it currently has limitations that will be discussed in 6.1. The current harness can send arbitrary messages to the root through a custom kernel mode driver, by utilizing a partially undocumented VMBus-related process. We have therefore shown/proven that fuzzing is possible for storage-related IPC through VMBus via our preliminary set-up

6 Future work

6.1 Full-fletched fuzzing

We succeeded in sending an SRB message through VMBus, which resulted in a successful client status code. To establish a more complete fuzzing harness, a confirmation of receiving the message from the root is required. As this would confirm that the root partition is processing our sent request. Furthermore, a monitoring driver on the root partition needs to be established to monitor whether the root partition is still functional after an arbitrary message has been sent to the root. Moreover, a userland executable in the guest partition needs to be created to craft and send the arbitrary messages. This currently is accomplished directly by the custom kernel land driver. This poses a risk of crashing the guest in unforeseen circumstances, a user-land executable performing IOCTLs to the kernel land driver eliminates some of the risks in doing so.

6.2 Exploring other IPC

The scope of our project was limited to VMBus and storage-related IPC. The realm of IPC also includes synthetic interrupts, and IPC over VMbus other than storage drivers. Future work in researching fuzzing harnesses for other IPC flavors is a viable route to take to establish the same goal as has been central in this research project as well.

References

[1] D. Babić, S. Bucur, Y. Chen, F. Ivančić, T. King, M. Kusano, C. Lemieux, L. Szekeres, and W. Wang. Fudge: fuzz driver generation at scale. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 975–985, 2019.

[2] M. H. et al. Make your own integration services. `https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/user-guide/make-integration-service`, 2023. [Online].

[3] A. Fioraldi, D. Maier, H. Eißfeldt, and M. Heuse. {AFL++}: Combining incremental steps of fuzzing research. In *14th USENIX Workshop on Offensive Technologies (WOOT 20)*, 2020.

[4] F. Johansson and C. Lindström. Inter-process communication in a virtualized environment, 2018.

[5] J. Kääp. Hyper-v vmbus based traffic interception and fuzzing, 2020.

[6] MSRC. CVE-2020-1040. `https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2020-1040`, 2020. [Online].

[7] MSRC. CVE-2021-38672. `https://msrc.microsoft.com/update-guide/en-US/advisory/CVE-2021-38672`, 2021. [Online].

[8] P. H. Ophir Harpaz. hAFL1 – Our Journey of Fuzzing Hyper-V and Discovering a Critical 0-Day. `https://www.akamai.com/blog/security/discovering-a-critical-0-day`, 2021. [Online].

[9] Z. Shan, X. Wang, T.-c. Chiueh, and X. Meng. Facilitating inter-application interactions for os-level virtualization. In *Proceedings of the 8th ACM SIGPLAN/SIGOPS conference on Virtual Execution Environments*, pages 75–86, 2012.

[10] swiat. First Steps in Hyper-V Research . `https://msrc.microsoft.com/blog/2018/12/first-steps-in-hyper-v-research/`, 2018. [Online].

[11] swiat. Fuzzing para-virtualized devices in Hyper-V . `https://msrc.microsoft.com/blog/2019/01/fuzzing-para-virtualized-devices-in-hyper-v/`, 2019. [Online].

[12] v-thepet et al. Inter-Partition Communication. `https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/tlfs/inter-partition-communication`, 2021. [Online].

[13] v-thepet et al. Hyper-V Architecture. `https://learn.microsoft.com/en-us/virtualization/hyper-v-on-windows/reference/hyper-v-architecture`, 2022. [Online].

A CUSTOM.sys project

https://gitlab.os3.nl/varsene/hyper-v-storvsc