----------------------- REVIEW 1 ---------------------

Overall evaluation: -1 (weak reject)

----------- Overall evaluation -----------

Summary:

This work aims at secure VM isolation against malicious hypervisor.

The proposed scheme, named HyperMI, hooks every interaction between the hypervisor and the VMs so that HyperMI manages the critical data and operations in a protected region.

HyperMI requires neither an additional hardware nor elevated privileges.

Pros:

The proposed scheme has implemented based on KVM.

It is analyzed against real-world threats.

Its performance overhead is measured using benchmark applications.

Cons:

In Section 6-D, the architectual difference between HyperMI and SKEE/SecPod/ED-Monitor is not clarified.

The quality of English writing is below the university level. There are too many (more than twenty; gave up to count) misuse of commas, sentences connected without conjunction, inconsitent tense, etc. The authors are strongly recommended to use a third-party proofreading service.

The implementation seems not downloadable.

----------------------- REVIEW 2 ---------------------

Overall evaluation: 1 (weak accept)

----------- Overall evaluation -----------

The paper proposes an approach for runtime protection of VMs without requiring additional hardware nor higher privileged levels than the hypervisor. Then paper reports a prototype implementation with 4k source lines of code.

The paper is well organized describing the threat model and assumptions, design and detailed aspects of the implementation. The experimentation conducts a security analysis and evaluates the overhead of the solution, which is found to be below 6% for the SPEC CPU2006 benchmark suite.

While the paper reports an implementation and prototype used for the evaluation, the implementation seems not to be made publicly available. From this perspective, the value of the contribution has some limitations with regards to the sharing of knowledge with the research community, since the community cannot easily reproduce and take-up the practical results.

----------------------- REVIEW 3 ---------------------

Overall evaluation: 2 (accept)

----------- Overall evaluation -----------

Paper summary

This paper proposes a mechanism that ensures and supports a secure isolated execution environment at the same privilege level with hypervisor. The proposed solutions is very interesting because does not need more permissions but, more importantly, additional hardware (can be adapted without constructing and changing the physical infrastructure).

Comments for author

The paper is well written and easy to understand.

The authors must have forgotten to write some references. An example of this: "However, a recent survey"; which survey?

Also I would suggest the increasing of the specification of the attacks on a Threat Model.

----------------------- REVIEW 4 ---------------------

Overall evaluation: 2 (accept)

----------- Overall evaluation -----------

This paper presents a software-based approach, called HyperMI, that can protect guest VMs from compromised hypervisors. The authors implement HyperMI into KVM by modifying part of the Hypervisor code and inserting hook functions for redirecting four types of operations from the Hypervisor to HyperMI.

In Sec IV A, since both the compromised hypervisor and HyperMI knows the address of switch gate, "the entrance address must be protected after switching to HyperMI.... introduced in section IV-B". As guaranteeing the security of HyperMI world is one of the fundamental requirements, it is not clear what the novelty of HyperMI is. That is, when reading through Sec IV-B, it seems like HyperMI is built on a number of recent same-privilege-level software based approaches, e.g, SKEE, SecPod. It would be helpful to explicitly point out the key differences (even after reading through Sec VI D). Also, "the system needs to run the switch process" - what is the system here referring to?

The performance evaluation is very limited. First, the experiment setup isn't really clear, e.g., how many guest VMs are configured to run on the physical server? Are the results measured in Figure 6 from running SPEC CPU2006 and Bonnie++ inside one VM while all other VMs are idle or running the same workload? How many times the switches between the normal world and HyperMI world happens during the benchmark duration? That said, some microbenchmarks such as breaking the overhead of switching to HyperMI, impacts on server boot up time, VM creation time etc will be helpful to really understand the performance of HyperMI. Moreover, is it possible to compare HyperMI to other software-based approaches and put the 6% performance overhead in context?

----------------------- REVIEW 5 ---------------------

Overall evaluation: 0 (borderline paper)

----------- Overall evaluation -----------

The authors presented a novel solution with aims to reduce the vulnerabilities of VM-based environments. The solutions provides isolation and monitoring with an insignificant overhead.

I miss a direct comparison with the the approaches enumerated in the introduction section.

Authors should describe a little bit the cited codes of vulnerabilities. For example, using Google, I found that vulnerability CVE-2018-1087 was discovered for kernel version 4.13. However, as far I found, reports do not include kernel version 4.15. I am not an expert in the field, but looks like a simple update can fix the vulnerability. Authors should provide more insights about the contribution of the research.

Other point, is the vulnerability CVE-2017-8106, evaluated in the paper. Again, this vulnerability looks that affects old versions of kernels.

----------------------- REVIEW 6 ---------------------

Overall evaluation: -1 (weak reject)

----------- Overall evaluation -----------

The authors have proposed HyperMI which is an addendum to existing hypervisors for addressing the security and vulnerability issues associated with the hypervisors. The proposed HyperMI world provides tasks including monitoring, isolation, content management. The authors implement HyperMI in x86 hypervisor and evaluate it with a set of standard benchmarks. Overall the proposed HyperMI results in acceptable overhead.

The primary problem with the paper is the writing. At many points in the paper, the statements made by the authors are unclear. While a software-based solution to handling some of the issues is a good idea, the overall performance penalty may not be acceptable to an end-user. There are several reasons why implementing additional functionality in software in the hypervisor is a bad idea. These include the HyperMI itself can be hacked, especially since it is a software solution. The other major problem with the paper is its overhead evaluation. Even if the software idea actually reduces the cost of the solution, the performance penalty is high and the insights in the performance evaluation are weak.

----------------------- REVIEW 7 ---------------------

Overall evaluation: -2 (reject)

----------- Overall evaluation -----------

This paper presents HyperMI, a protection mechanism for virtual machines, that lives at the same privilege level as the hypervisor and protects VMs from the hypervisor being compromised.

The idea behind HyperMI is interesting because it does not require extra hardware and reduces the Trusted Code Base to a small amount of lines of code. However this idea is not really new since it was already applied to operating system kernels in SKEE (NDSS 2016), acknowledged by the authors, and earlier in Nested Kernel (ASPLOS 2015) which the authors seem to ignore. The originality of HyperMI is actually not well explained in the paper, neither in comparison to SKEE, nor in comparison to related work for same-privilege-level isolation in hypervisors like ED-Monitor (although ED-Monitor is cited). Moreover the way HyperMI works with respect to the hypervisor could be considered as similar to para-virtualization with operating system kernels, which could be discussed in related work as well. Finally, the authors ignore recent mainstream processors extensions to secure VM memory, like AMD Secure Encrypted Virtualization (although the first generation has some security flaws)!

, which was advertized by Microsoft as used in its Azure platform.

The paper has unfortunately other important weaknesses, especially about the presentation, the clarity (thus maybe soundness) of the security design and the performance evaluation which make it too far from being acceptable for publication.

The presentation suffers a lot from many language mistakes, which makes think that a thorough proof-reading phase is missing. This becomes a real issue at several places where the reader just fails to understand what the authors mean. Examples:

- IV.C (last lines) "especially, to cut down the performance overload, HyperMI set VM exit to configure of conditional exit events as non-running and reduce the occurrence number of exit events."

- IV.D (first paragraph) "If the page is remapped or double mapped, the system causes a page fault." It took a while before understanding that this was the beginning of a description of the series of actions taken after "double page mapping" or "page remapping" attacks.

- IV.D (last sentence) "This solution is the same as the treatment method of the pointer for the purpose of security protection.": Which pointer?

At least Subsection IV.D should definitely be rewritten. Moreover the design section (Section III) is very short and many design-level details (like the actual attacks to protect from and the strategies to prevent them) actually appear in Section IV (Implementation). It is actually hard to see how Section IV is more detailed about the implementation than Section III. Perhaps Section III should be renamed "Architecture overview" and Section IV renamed "Design"?

Security-wise, first the threat model should be more detailed. It is assumed that the hardware is trusted but writing so the authors seem to consider the platform firmware, BIOS or UEFI, and thus the code running in SMM (having higher privileges than the hypervisor) as part of the hardware. Although this assumption is conventional, it would more clear to write it explicitly. However the worst issue with the threat model is that it is unclear which kind of hypervisor corruption can happen and how they are made benign or just prevented by HyperMI. The authors almost always refer to the hypervisor as "compromised hypervisor" but HyperMI works only if this corruption is kept within some bounds and if HyperMI is able to keep corruption within these bounds. These bounds are not clearly presented.

This leads to the second security issue in this paper. The protection mechanisms used to protect the HyperMI world from the (compromised) hypervisor are not clearly presented and not all well justified. The authors seem to refer to SKEE for these mechanisms (only a sentence in the security analysis of the evaluation section seems to mean that! This should have figured in Section III "Design"!) but the presented mechanisms in Section IV are not detailed enough to be sure of how privileged instructions are kept out of the (compromised) hypervisor code, even in case of code reuse attacks that jump in the middle of multi-byte instructions. With this respect, the Nested Kernel paper (which, contrary to SKEE, addresses the x86 architecture) is very clear. Moreover to protect a VM from malicious accesses to its VMCS and EPT structures, it is written that first the addresses of these structures are hidden from the hypervisor (does it mean security-by-obscurity??), second the memory !

hosting these structures is not mapped in the hypervisor address space, and third all specific instructions (VMREAD and VMWRITE) required to access the VMCS structures in the hypervisor code are replaced by hooks to HyperMI. Assuming that HyperMI's mechanisms to first intercept the hypervisor changes to its page table and second prevent the hypervisor from executing directly VMREAD/VMWRITE instructions can not be bypassed, what is the point of hiding VMCS and EPT addresses?

The evaluation section is unfortunately poor. The security analysis part is half done, since the paragraph about IOMMU shows no argument. The performance evaluation misses details about the experiments (e.g. the Linux kernel version is not enough to describe an hypervisor based on KVM), only runs system-wide peformance benchmarks while the detailed overheads of interception mechanisms would be interesting to know, and lacks analysis of the experiment results. For instance, the analysis of Bonnie++ results just says that they show a low overhead altough they show different overheads depending on the access pattern. Some analysis are just unclear because we miss explanations. For instance:

- (about SPEC CPU2006 results) "This can incur worlds switching which involves fewer register access and fewer TLB flushes with PCID technique" -> It reads like a comparison with another approach. Which one?

- (about Bonnie++) "HyperMI has no extra memory operations for I/O data" -> Does it mean that no VMCS access and no EPT accesses are required for I/O?

A few other minor remarks and questions, all about Section IV.D:

- It is said that the VMID is based on a hash of the image of a VM: what about several read-only (aka disposable) VMs based on the same image and run concurrently?

- VM marking: it would be more clear to present this as a shadow structure mapping VMs to EPTP values, as the hypervisor must in some way maintain this information.

- page marking: in short page marking is a structure that records for each page which context (a VM or the hypervisor) maps it. Instead of maintaining such a (simplified) reverse mapping, was page coloring considered?

- Instead of "VT-d", the authors certainly meant "VT-x" (VT-d is Intel's name for IOMMU).