

VirtAV: an Agentless Antivirus System based on In-Memory Signature Scanning for Virtual Machine

Hongwei Tang^{1,2,3}, Shengzhong Feng^{1,3}, Xiaofang Zhao^{2,3}, Yan Jin^{2,3}

¹Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055, China

²Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China

³University of Chinese Academy of Sciences, Beijing 100049, China

tanghongwei@ict.ac.cn, zhaoxf@ict.ac.cn, sz.feng@siat.ac.cn, jinyan@ncic.ac.cn

Abstract—Antivirus is an important issue to the security of virtual machine (VM). According to where the antivirus system resides, the existing approaches can be categorized into three classes: internal approach, external approach and hybrid approach. However, for the internal approach, it is susceptible to attacks and may cause antivirus storm and rollback vulnerability problems. On the other hand, for the external approach, the antivirus systems built upon VMI technology cannot find and prohibit viruses promptly. Although the hybrid approach performs virus scanning out of the virtual machine, it is still vulnerable to attacks since it completely depends on the agent and hooks to deliver events in the guest operating system. To solve the aforementioned problems, based on in-memory signature scanning, we propose an agentless antivirus system VirtAV. VirtAV can monitor the specific event of the guest VM that is defined as the first instruction-fetch operation on a newly updated host memory page frame, and can scan virus in the page when the event occurs. As an external approach, VirtAV doesn't rely on any event or agent in the guest OS, so it guarantees the security of itself to the greatest extent. In addition, it provides full life cycle protection for VMs, no matter which state (running, paused, resumed or migrated) they are in. We implemented a prototype by extending Qemu/KVM hypervisor. Experimental result demonstrates that the function of VirtAV is verified (by finding 100% of the 3546 sample viruses) and the overhead of VirtAV on guest performance is acceptable. Especially, VirtAV has little impact on the performance of common desktop applications, such as video playing, web browsing and Microsoft Office series.

Keywords—agentless, antivirus, antivirus storm, virtual machine, virus signature

I. INTRODUCTION

ANTIVIRUS protection is necessary for modern computers. As virtualization technology has been extensively adopted to enhance the effectiveness of resource utilization, more and more applications are deployed in VMs. Currently three antivirus approaches, i.e., internal approach, external approach and hybrid approach, are provided in the industry and academia.

In the **internal approach**, antivirus software is installed in the VM and it often uses signature-matching [1][2] technology to search viruses in files [3][4]. When all guest VMs on a host machine schedule virus scanning or signature database updating simultaneously, the host will be overloaded and corresponding performance of VMs will be degraded dramatically. As a result, the 'antivirus storm' problem [5] is inevitable because of a large number of concurrent resource-intensive operations stressing on both computing resource and I/O resource. On the other hand, as a useful approach that facilitates VM fault tolerance and system maintenance, snapshot & rollback also has been applied. However, when the VM rollbacks to a snapshot previously saved, the signature database installed inside may be out of date [6][7]. In this case, antivirus software may be unable to detect newer viruses timely, which leads to security vulnerabilities of antivirus peculiar to virtualization environment. We call it as a 'rollback vulnerability' problem. Moreover, antivirus software is highly susceptible to attacks when exposed in the VM under protection [8].

The **external approach** is inspired by the feature that the VMM has supervisory privilege on guest VMs. Antivirus software is totally moved out of the VM [8][9][10], and gets necessary information from the guest OS using VMI [11] techniques. The existing work following this approach can only detect virus in guest files or memory, and can't prohibit it properly.

The **hybrid approach** is usually composed of two parts: hooks on the guest OS to monitor events and capture information, and scanning engine deployed in a dedicated VM [5] [12] [13]. The most widely used antivirus product from VMware follows this approach [13]. Although this approach can avoid the antivirus storm problem and the rollback

Manuscript received November 30, 2015. This work was supported by the National Natural Science Foundation of China under Grant No.61402444, No.61202413 and No.61202061.

H. Tang is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190 China. (phone: +86-153-2175-6132; fax: +86-10-6260-0276; e-mail: tanghongwei@ict.ac.cn).

S. Feng is with the Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences, Shenzhen 518055 China. (e-mail: sz.feng@siat.ac.cn).

X. Zhao is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190 China. (e-mail: zhaoxf@ict.ac.cn).

Y. Jin is with the Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190 China. (e-mail: jinyan@ncic.ac.cn).

vulnerability problem, the hooks or agents in the guest OS are still vulnerable to attacks.

Motivated by the aforementioned observations, an agentless antivirus system VirtAV that is based on in-memory signature scanning is proposed in this paper. Basically, it can be classified into external approach since signature scanning and database updating are performed completely outside of guest VMs under protection. The main contributions are listed as follows.

- We designed an agentless approach for antivirus protection on VMs, which can actively monitor the guest VMs on the VMM layer, transparently perform virus detection and timely prohibit virus found when the guest VM is running. Such an approach guarantees the security of antivirus system to the greatest extent. Meanwhile, it also provides full life cycle protection for VMs.
- We proposed memory signature of virus, variant of file signature widely used in traditional antivirus software, which can be used to uniquely identify virus when it has been loaded or even partially loaded into memory. Such an approach helps antivirus system detect and prohibit virus in memory before it is activated.
- We implemented a prototype system based on Qemu/KVM hypervisor. To verify the functions provided by VirtAV, We performed tests on 3546 sample viruses, and the result shows that 100% of the sample viruses can be found. Moreover, performance benchmarks for Windows booting, video playing and synthetic office workload, which are run on single VM and multiple concurrently VMs, show that the VirtAV overhead is reasonable and acceptable. Furthermore, the antivirus storm and rollback vulnerability can be eliminated by VirtAV.

The rest of the paper is organized in the following manner: Section 2 gives the main design principles on VirtAV. Section 3 gives the implementation details of VirtAV. Section 4 presents the experimental results from the viewpoints of function and performance. Section 5 gives a brief review of related works on virtual machine antivirus. Finally Section 6 concludes the paper and mentions the future work.

II. GOALS AND CHALLENGES

The overall goal of VirtAV is to provide antivirus protection for VMs without introducing the antivirus storm problem and the rollback vulnerability problem firstly. In addition, as the antivirus agent in the guest OS leads to vulnerabilities and weakness points on security, VirtAV is expected to implement an agentless solution. Furthermore, for the purpose of VirtAV to be a practical solution, the negative performance impacts on the guest VMs should be as low as possible. To achieve the goals above, the key challenges that we will investigate are highlighted as follows.

A. Virus detection and prevention in memory

In traditional ways, file signature is used to distinguish specific virus from other files. Periodically file system scanning and specific file scanning on critical file operations are two common policies in commodity antivirus products. However, file signature based virus detection needs understanding of OS level semantics which can't be obtained by the VMM. As we all known, memory is the only way that must be passed for the execution of any binary codes, which means we can detect and prevent virus in memory before it is executed by CPU in theory. However, it is more difficult to detect virus in memory than in files. The reason is that binary codes are loaded into memory on demand in the unit of page only when the code is fetched by CPU. As a result, we usually get partial view of the executable in memory even when it is being executed. That is, we can no longer rely on file signature to detect virus in memory. Subsection 3.2 provides the details to solve this problem.

B. Virus scanning trigger strategy

The time instance of triggering virus scanning in memory must be before that of executing binary codes of the executable. However, the VMM cannot intercept guest OS level events. Instead, it can only capture hardware level events. Therefore, the strategy regarding when and how to trigger virus scanning is another problem. Subsection 3.3 describes how VirtAV utilizes memory virtualization to solve this problem.

C. Performance guarantee

To detect and prohibit virus promptly, VirtAV scans virus on the critical path of binary code execution in the guest VM. Moreover, the execution is paused when VirtAV is scanning virus, which means the performance overhead is inevitable. To be a practical antivirus solution, VirtAV should reduce the overhead and provide acceptable performance to guest VMs. The solution is provided in subsection 3.4.

III. DESIGN AND IMPLEMENTATION

Basically, VirtAV is an agentless antivirus system for VMs which provides antivirus protection by extending KVM and Qemu. It identifies virus by memory signature and dynamically scans binary codes in host memory to detect viruses. It interposes synchronous virus scanning operations on the critical path of program executions in guest VM and it ensures that any code executed by vCPU has been examined. It is transparent to the guest OS and requires no modifications or hooks on the guest OS. The design and implementation is discussed in the following subsections.

A. Architecture

The overall architecture of VirtAV is shown in Fig. 1. VirtAV is built upon KVM/Qemu hypervisor and is composed of four parts: 1) VirtAV-engine embedded in Qemu, 2) a centralized signature database deployed on the host Linux, 3) VirtAV-stub in KVM, and 4) VirtAV-cleaner on host Linux.

VirtAV-engine: We integrated the scan engine of ClamAV [3] with Qemu and designated it as VirtAV-engine. It runs in

the same context with Qemu's vCPU thread. It scans host memory footprints of executables in guest VMs to search memory signatures of virus. Furthermore, the vCPU thread will be paused when it scans host memory. At the core of VirtAV-engine, different matching algorithms can be employed. In current implementation, we use the default algorithms adopted in ClamAV including AC and BM [3].

Signature Database: VirtAV stores memory signatures of viruses in a centralized database deployed on the host Linux. The database is shared by all the VirtAV-engines on the host, and updating for it would not cause storm. We proposed memory signature to identify virus. The definition of memory signatures are presented in subsection 3.2.

VirtAV-stub: To trigger virus scanning at the right time, we extended the memory virtualization module in KVM to trap specific memory access events. Once trapped, the vCPU is paused and VirtAV-stub transfers the guest memory page holding binary codes to user-level VirtAV-engine for scanning. When virus is detected, VirtAV will take actions according to predefined policies, e.g., killing the corresponding process and isolating/removing the executable by default. To kill the process, it manipulates the guest page table entry of the process to revoke executable right on the guest page frame using VMI techniques. After the vCPU resumes, the process is automatically killed by the guest OS as general protection fault.

VirtAV-cleaner: It is an auxiliary utility for virus treatment. It locates the executable of the virus according to information provided by VirtAV-stub, and takes actions to isolate or remove the virus. It is built upon libguestfs [14] which provides interfaces to access and modify guest file-system from outside of the VM.

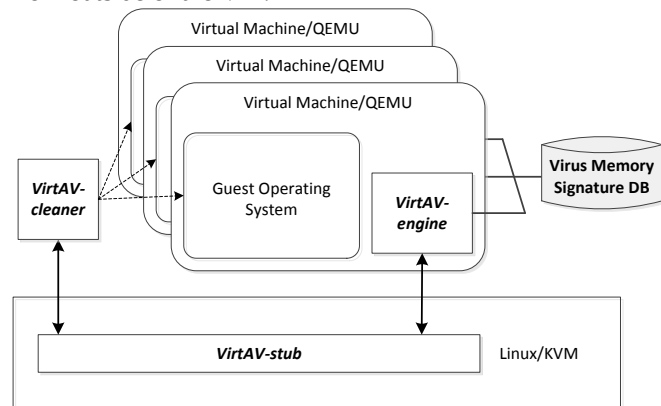


Fig.1. Overall architecture of VirtAV

B. Memory Signature of Virus

To find virus accurately and timely in memory, VirtAV identifies virus using memory signature which is slightly different from file signatures. File signature is usually picked up from the text section of virus executable. However, the way to generate memory signature is different from that of file signature. The reason can be explained in detail as follows.

In modern operating systems, such as Windows and Linux, memory management is based on paging which divides the

continuous system memory into fixed-sized page frames. Page frames are assigned and mapped to processes' virtual address spaces on demand. When a process is created, the OS establishes mappings between the process' virtual memory areas and sections of executable and libraries. By that time, page frames are not assigned to the process, and sections are not loaded into memory either. When the process starts to run, especially the time instance that the CPU attempts to fetch instructions from the virtual memory area mapped to the text section, page fault will be generated as page frame is not present. Then the OS will assign a page frame to the process and eventually load a block of contents from the executable. The text section may reside in more than one page frames and should be loaded multiple times. So we cannot rely on scanning the whole executable to detect virus because part of the virus could have been executed and may have infected the system before the file signature is loaded. Take Fig. 2 as an example. Before the file signature is loaded into memory, the codes in page frames previously loaded may have been already executed and malicious activities may have taken place yet.

Memory signature of virus is defined as a binary or hex string, which can be used to identify the virus uniquely when the virus is in memory. The memory signature can only be picked out from the block of text section that will be mapped into the same memory page frame as the entry point of the executable. Fig. 2 shows comparison about picking out file signature and memory signature from a windows virus file in PE format [15].

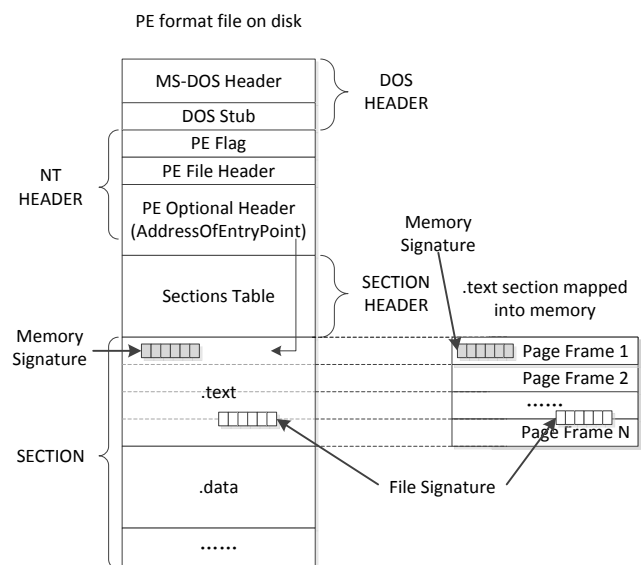


Fig. 2. Memory signature vs. file signature of virus

VirtAV confines that memory signature can only be picked out from the specific page-sized block of the text section. This restriction may not bring significant challenges on the generation of signature. On the one hand, most viruses are small in size, and even the size of text section is usually smaller than page size (typically 4096Bytes). For these viruses, we can take the file signatures as the memory signatures. On

the other hand, we can use longer signature string or multiple signature strings to identify virus uniquely.

C. Virus Scanning in Memory

Generally, on-demand scanning is widely adopted in commodity antivirus products. For example, scan operations can be triggered when opening an executable to execute, downloading files from the Internet and so on. It is required that the on-demand way completely understands OS-level semantics and even hooks on system calls. However, the VMM has no knowledge of what happened in guest OS and can merely capture hardware level events, such as device I/O operations, EPT [16] violations and etc. As a consequence of the semantic gap problem [10], previously proposed external solutions have fundamental shortcomings on the instantaneity and effectiveness.

To get around this problem, VirtAV adopts a different way that scans virus in host memory page frames which are mapped to guest memory space. The idea is based on the fact that executables to be executed in the guest VM should be firstly loaded into host page frames, which can be viewed by the VMM. To trigger scan operation, we take a specific hardware event which is defined as the **first instruction-fetch operation on a newly updated host memory page frame**, abbreviated to first fetch event. When a host page frame is updated, it is possible that virus has already sneaked into the page frame. In this case, we need to examine each newly updated page frame before fetching instructions by vCPU. To catch the event efficiently, we program the EPT entry bits which control access permissions on host page frames. The basic logic is that write-enable flag bit and fetch-enable flag bit should not be set at the same time. When the write-enable flag bit is set, the guest OS is allowed to load executables into the page frame. When the fetch-enable flag bit is set, the vCPU is allowed to fetch instructions from the page frame. When the event occurs, VMEXIT is generated and the vCPU accessing the page frame will be trapped and paused. To handle the event, VirtAV will scan the page frame to detect memory signatures, and prevent the virus from executing if found. Otherwise, after scanning, the vCPU will be resumed and allowed to fetch instructions from the page frame. The flow path from loading a program to executing it in the guest VM is depicted in Fig. 3.

As for a host page frame, it may be in one of the following four states before loading binary codes of the guest program into it.

State I: The host page frame has not been allocated and mapped to the guest yet.

State II: The host page frame has been allocated and mapped, and the access permission specified in EPT entry is read-only.

State III: The host page frame has been allocated and mapped, and the access permission specified in EPT entry is fetch-enable but not write-enable.

State IV: The host page frame has been allocated and mapped, and the access permission specified in EPT entry is write-enable but not fetch-enable.

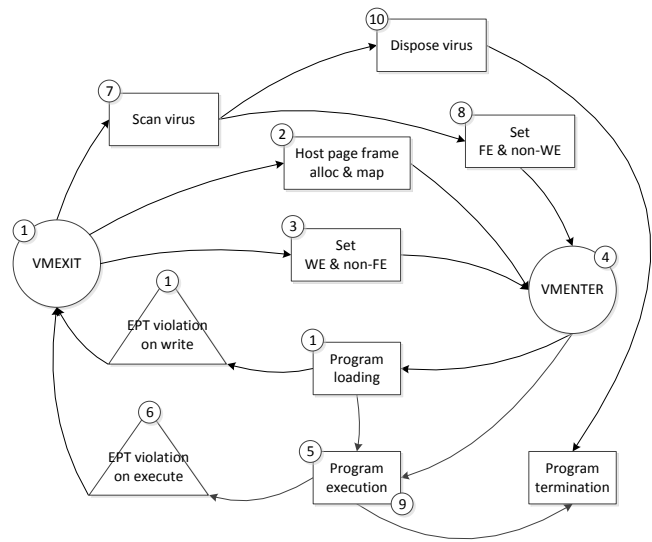


Fig. 3. Flow path of program execution in guest VM

① When loading binary codes of the guest program, VMEXIT is generated as EPT violation occurs if the host page frame is in state I, II or III. ② For state I, the VMM will allocate a host page frame, establish the EPT mapping and set the write-enable flag bit in the EPT entry. ③ For state II and III, the VMM will simply set the write-enable bit and clear the fetch-enable bit in the corresponding EPT entry. ④ After the EPT violation is handled, the VMM will resume the vCPU by generating VMENTER. ⑤ In the guest VM, after successfully loading the binary codes, the vCPU will attempt to fetch instructions from the host page frame. ⑥ Then, another VMEXIT will be generated as the fetch-enable flag bit of the host page frame is cleared. That is the very first fetch event defined above. ⑦ By having captured the event, the VMM invokes virus scanning. ⑧ If no virus is found, the VMM will simply set the fetch-enable bit and clear the write-enable bit in the EPT entry and resume the vCPU. ⑨ Finally, the program successfully executes in the guest VM. ⑩ If any virus is found, actions will be taken to deal with that virus.

D. Performance Considerations

As is discussed in previous subsections, virus scanning is on the critical path of program execution in guest VMs, which means performance and efficiency are very important for VirtAV to be a practical solution. In this subsection, we will introduce some fundamental design considerations focusing on performance and efficiency.

1) Scan text section of the executable only

Generally speaking, the time consumed for virus detection is proportional to the size of the data to be scanned. Commodity antivirus products scan all sections of the executable, while VirtAV can reduce the scan duration by scanning the text section of the executable only.

2) Scan each host page frame loaded with binary codes once

VirtAV may degrade the performance of guest VMs due to the overhead introduced in the path of program execution. The overhead mainly comes from page scanning and context switches between the VMM and guest VMs. To reduce the overhead, VirtAV scans each host page frame loaded with binary codes only once, provided that the content of the page has not be changed since it is scanned. As the write-enable flag bit in EPT entry for the page frame is cleared, the page content can't be modified, so that there is no need to rescan the page frame despite the binary codes in the page frame may execute multiple times.

Moreover, in mainstream operating systems, executables are usually cached in system memory, such as system file cache in Windows and page cache in Linux, and shared by different address spaces. VirtAV detects virus on the view of host memory, which corresponds to system memory of the guest VM, so the host page frames loaded with binary codes need to be scanned only once.

3) Scan for each vCPU independently

VirtAV-stub monitors and traps the first fetch events on each vCPU. When the event is captured, the vCPU generating the event is paused and trapped into VMM by VMEXIT. The other vCPUs in the same guest VM keep on running without any disturbances. If multiple events on the same page frame concurrently happens on more than one vCPUs, VirtAV will scan the page frame only once just for the vCPU on which the event firstly occurs. In this case, VirtAV-stub sets the flag of the page frame to PG_SCANNING to avoid duplicated in-flight scans, and just simply makes other vCPUs to wait for the result.

4) Zero-copy for kernel-to-user data transfer

VirtAV-stub captures the events in KVM context in kernel level, while VirtAV-engine works in Qemu context in user level. So it needs to transfer page content from kernel level to user level for scanning. To improve performance, we use zero-copy to transfer data by dynamically mapping the host page frames in Qemu address space that can be directly accessed by VirtAV-engine.

E. Virus Treatment

If there is matched memory signature found in host page frame, VirtAV should dispose the virus according to predefined policies, such as killing the corresponding process and removing the virus file by default. The sketchy procedure is described as follows.

In general, VirtAV-engine, VirtAV-stub and VirtAV-cleaner cooperate to deal with virus. Firstly, VirtAV-engine notifies VirtAV-stub that it has found virus in the host page frame. Under the help of VMI tools, VirtAV-stub locates the page global directory for the current process running on the vCPU in guest VM, and walk through the guest page tables to find the entry referencing the guest physical page frame which holds the virus. Then, VirtAV-stub updates the flag bit of the entry to reclaim execute-permission on the guest page frame, resumes the vCPU and lets the current process to go. The current process will be killed immediately by the guest OS due to general protection fault. In the meanwhile, VirtAV-stub also notifies VirtAV-cleaner the location of the executable

corresponding to the virus. Finally, VirtAV-cleaner isolates or removes the executable accordingly.

IV. EXPERIMENTAL EVALUATION

A. Environment Setup

The hardware platform is composed of a virtualization host and a NFS server backed by a fiber channel disk array. The virtualization host includes dual Intel Xeon E5620 processors with 4 CPU cores (@2.40GHz) each, 24GB system memory and a 300GB local disk. The NFS server includes dual Intel Xeon E5620 processors, 16GB system memory and a 300GB local disk. The fiber channel disk array is organized in RAID5 using ten disks (Seagate ST3300657SS, SAS, 300GB). As VirtAV is built upon the KVM module in Linux kernel version 3.17.4, Qemu version 2.1.2 and ClamAV version 0.98.6, we run 64-bit CentOS release 6.5 with 3.17.4 kernel and Qemu-2.1.2 on the host hardware platform to facilitate performance comparison. Guest VMs are configured as 4 vCPUs (2 sockets with dual-core), 3GB memory, and 50GB disk space. We run 32-bit Windows 7 SP1 in guest VMs.

B. Functional verification

To verify the validity of our approach, we use an open virus package for Windows platform downloaded from the Internet that totally contains 3546 viruses [17]. To generate the memory signature database, we manually pick out memory signatures for the sample viruses. There are more than 3000 viruses with code section size smaller than 4096 bytes. For these viruses, we simply take the file signatures as the memory signatures.

We setup the Windows guest with clean state and take a snapshot for it. Before performing test on a sample virus, the Windows guest is rolled back to the snapshot to avoid influences among different viruses. In the end, the results show that VirtAV can find out all the sample viruses accurately.

C. Performance Evaluation

1) Single VM

We use a Windows guest VM to benchmark the performance of VirtAV on four dimensions, including 1) booting Windows, 2) launching applications, 3) playing video, and 4) synthetic workload generated by PCMark [18] test suite. We compare the performance of VMs running with and without VirtAV (as called Native below).

a) Performance of booting Windows

The boot time of Windows guest VM is shown in Table 1. Although it is slower with VirtAV than the native case, the total boot time is still acceptable from an overall perspective.

TABLE 1. WINDOWS GUEST BOOT TIME

	VirtAV (seconds)	Native (seconds)
Boot Time	34.8	27.5

b) Performance of playing video

We take the build-in sample video in Windows 7 for test. As a reference, the pure time duration for playing the video is 30 seconds.

For the first time to play the video, VirtAV would scan virus on the page frames holding the executable of the media player and the libraries that it depends on. While this may be omitted in the following invocations of the media player. So we compare the first and following runs of the test, and the results are shown in Table 2. We run 10 times for each test case and take the average time. As can be seen from the table, the overhead of VirtAV is negligible when playing video.

TABLE 2. TIME DURATION FOR PLAYING THE SAMPLE VIDEO

	VirtAV (seconds)	Native (seconds)
First Play(Average)	33.6	32.4
Following Play(Average)	33.0	32.0

c) PCMark tests

We take PCMark (version 8) as the synthetic benchmarking workload. PCMark is a famous tool for testing the performance of PC. The benchmark produces a score as well as detailed results for comparison with other machines. As the guest VMs in our test-bed have no physical support on GPU, some tests cannot run. So we only compare the performance results of web browsing, writing and spreadsheet. The results are shown in Table 3. The performance is represented as time used to complete the corresponding basic operations, so shorter is better. It is clear that the performance difference is small between the two cases which means VirtAV introduces little overhead on these workloads.

TABLE 3. PCMARK TESTS

Workload	VirtAV (seconds)	Native (seconds)
Web Browsing-JunglePin	0.400	0.390
Web Browsing-Amaonia	0.130	0.130
Writing	4.130	3.660
Spreadsheet (Libre Office Calc Conventional)	23.600	22.525

2) Multiple VMs

We create seven VMs for benchmarking concurrent performance. All the virtual disks share a common base image file which is an advanced feature supported by Qcow2. On the whole, the performance overhead on concurrent VMs is reasonable, and antivirus storm problem can be avoided in VirtAV.

a) Performance of booting Windows

We power on the 7 VMs at the same time, and take the time that the last one takes to complete the booting process as the overall boot time. As is shown in Table 4, with VirtAV, the

boot time of concurrent VMs increases significantly, but still remains acceptable.

TABLE 4. CONCURRENT WINDOWS GUESTS' BOOT TIME

	VirtAV (seconds)	Native (seconds)
Boot Time	40.2	29.3

b) Performance of playing video

We play the sample video in the 7 VMs concurrently, and the results are shown in Table 5. The overhead is constant and low even in the multi-VM case.

TABLE 5. CONCURRENTLY PLAYING THE SAMPLE VIDEO IN 7 VMs

	VirtAV (seconds)	Native (seconds)
First Play(Average)	34.4	33.4
Following Play(Average)	33.9	32.7

c) PCMark tests

We run PCMark tests in the 7 VMs concurrently, and the results are shown in Table 6. As can be seen, the penalty on the application performance remains small in the multi-VM case.

TABLE 6. CONCURRENT PCMARK TESTS

Workload	VirtAV (seconds)	Native (seconds)
Web Browsing-JunglePin	0.430	0.416
Web Browsing-Amaonia	0.133	0.133
Writing	4.533	4.137
Spreadsheet (Libre Office Calc Conventional)	23.964	23.013

3) Performance of human-computer interactions

User experience is critical for desktop computer which is a typical use case of virtual machine. To evaluate the impact of VirtAV on interactive applications, we select the most commonly used softwares on Windows, including IExplorer (version 11), Microsoft Office (version 2010) Word, PowerPoint and Excel. Moreover, to quantify and measure the performance, we split the execution of these applications into two phases: launching phase and operating phase, and evaluate the two phases respectively. The launching phase starts from the time at which the user clicks on the icon to the time at which the application can accept inputs. The operating phase starts just after the launching phase ends, until to the termination of the application.

Performance evaluation of the launching phase is simple because we can take the time duration to launch the application as the indicator. The results are shown in Fig. 4, in which shorter means better. As the executables can be cached in system memory, virus scan may be avoided on the following invocations of the applications. Just as seen in the figure, the time for the first launch is longer than that for the following launches. Although it is obviously slower under

VirtAV than in the native case, the experience is still acceptable.

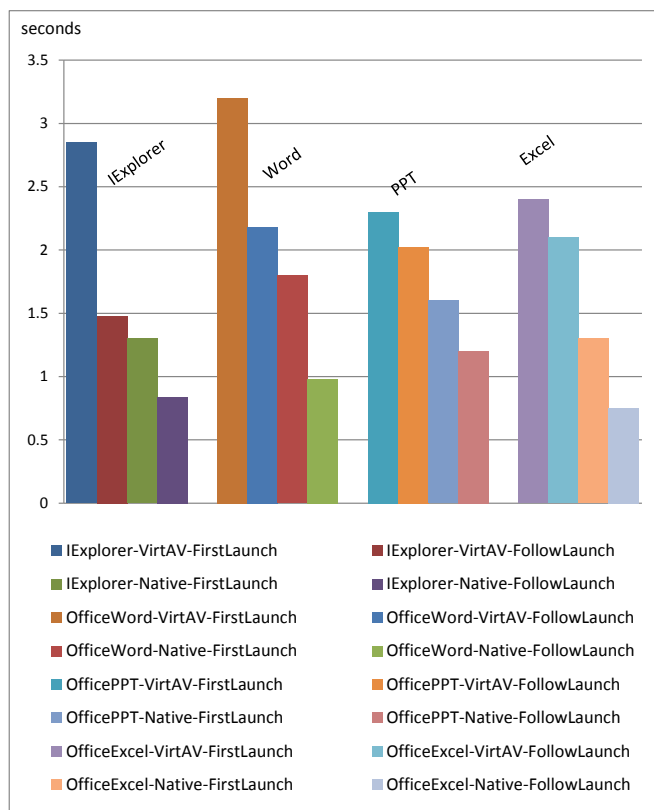


Fig. 4. Time of the launching phases

Evaluation of user experience on the operating phase is qualitative and we only focus on whether user operations are as smooth as in the native case. We perform basic operations on the applications, including typing letters in the input area, clicking on the menus, and scrolling pages. In the experiment, we found that only IEplorer can operate smoothly just as in the native case, while Microsoft Office series go through a period during which they are not responsive as in the native case. We call this period as unsmooth period. After the unsmooth period, the performance of the applications is boosted and user operations become smooth as in the native case. The lengths of the unsmooth periods are compared in Fig. 5.

To find out the reasons behind it, we profile memory access behaviors of the applications (MS Office Word, PPT and Excel) in the guest VM, and use SysInternalsSuite [19] tools to collect guest OS-level information. From the tracing logs in KVM, we found that the guest VM generates thousands of EPT violations on some specific page frames repeatedly during the tests. The violations are caused by execute faults and write faults alternately, which means that the vCPU updates and fetches instructions from the same page frame. Moreover, the tools (including Procmon, Rammap and Vmmap) show that these page frames are mapped as private in the application process's address space, and the protection

flags on the page frames are set to PAGE_EXECUTE_READWRITE.

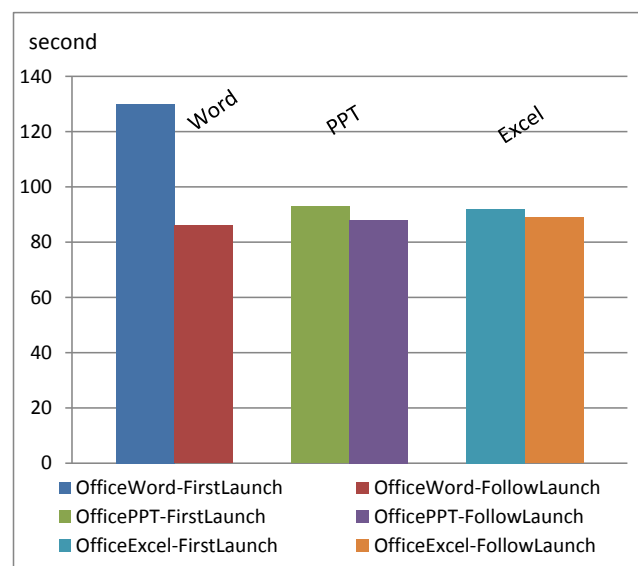


Fig. 5. Lengths of the unsmooth periods

Based on the above observation, we perform the following optimization to improve user experience. The basic idea is to detect repeated and alternant write-execute EPT violations on a specific page frame, scan virus on the first execute fault and skip subsequent execute faults. We use the triples <HPFN, GVFN, GPFN> in KVM to identify a mapping from guest virtual page to host page frame. We can get GVFN (Guest Virtual Page Frame Number) and GPFN (Guest Physical Page Frame Number) by reading from the VMCS structure [16]. When handling EPT violation caused by write fault or execute fault, we keep track of the corresponding GVFN and GPFN, and record the counts of write fault violation and execute fault violation in the host page descriptor, respectively. If the current GVFN and GPFN are not the same as the last couple recorded in the host page descriptor, we reset the count of write fault or execute fault to 1 and perform virus scanning on the page if it is a execute fault. Otherwise, the count of write fault or execute fault is increased by 1. If the count of execute fault is larger than 1, virus scanning is skipped.

After optimization, the unsmooth periods are eliminated, and the lengths of the launching phases are shortened slightly. Fig. 6 a) and b) show the comparisons on the performance of the first launch and subsequent launches respectively.

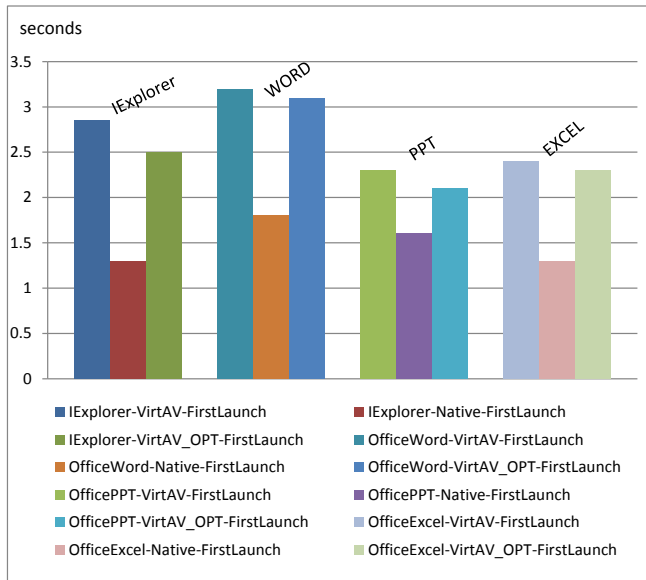


Fig. 6. a) Performance comparison on the first launches

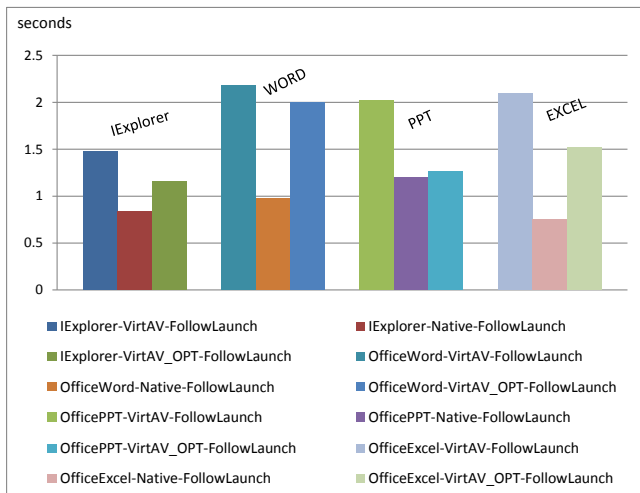


Fig. 6. b) Performance comparison on the subsequent launches

V. RELATED WORK

Virtualization technology is recently used to improve the anti-attack capability of intrusion or malware detection tools [8][26]. Security tools can be installed outside of the VM that need to be protected, and monitors the guest via special interfaces. As is isolated from the VM, the security tools are immune to attacks even if the VM is compromised by attackers.

As lacking of guest OS-level semantic (known as the semantic gap), the accuracy and effectiveness of the security tools outside the VM have to be sacrificed. Livewire [8] is a general IDS framework adopting VMI technology, which uses the priori knowledge about the guest OS data structures to interpret the hardware-level view in OS-level semantics. Especially, to detect signatures of malicious program, it performs a full scan of all the host memory. However, this

methodology cannot timely find out virus and prohibit them. VMwatcher [9] reconstructs semantic views of guest OS in a non-intrusive manner, and provides objects with the guest OS view to external malware detection tools. Just like Livewire, VMwatcher only provides malwares detection and cannot intervene execution of the guest. [12] proposed a hybrid approach enabling security tools to perform active monitoring while pertaining isolation to ensure security. It hooks the guest OS to monitor interested events and handle the events in an isolated VM. It solves the semantic gap problem in a simple manner, but it heavily depends on integrity of the guest OS. [20] described a framework that enables security monitoring applications to be placed in the untrusted guest VM without sacrificing the security guarantees. The monitoring application itself is protected by the VMM and should be self-contained to ensure its integrity. This method imposes restrictions on the monitoring applications and even requires reconstruction of commodity tools to adapt to this framework.

Antivirus storm problem is observed and highlighted by industrial community. To solve this problem, VMware & TrendTech [5][13] proposed a so-called agentless antivirus framework – vShield Endpoint. In that framework, the primary causes of storm including signature scanning and database updating are offloaded to a dedicated VM, called secure virtual appliance (SVA). Moreover, there is a lightweight agent (called ‘EPsec Thin Agent’) deployed in guest OS whose main duties are monitoring file operations in guest OS and forwarding files to SVA to be scanned. However, the framework is not truly agentless, and indeed, the agent is exposed to attacks from malicious users or malware. Moreover, the agent depends on the guest OS. When the agent is not running, for example in booting phase or shutting down phase of the guest OS, the guest VM loses protection of antivirus.

There are also some researches [21] [22] [23] focusing on malware analysis with the help of virtualization techniques. [24] presented an external approach to malware analysis which can hide the analyzer from the target. With the help of hardware virtualization technology such as Intel VT, it can offer both instruction level and system call level tracing of the target in guest VM. [25] described a Qemu-based system that can dynamically analyze Windows kernel-level code and extract malicious behaviors from rookits.

VI. CONCLUSION AND FUTURE WORK

In this paper, we presented an agentless antivirus solution VirtAV which relies on in-memory signature scanning. To catch virus accurately and promptly, VirtAV utilizes the memory signature to identify virus and captures the first fetch event to trigger virus scanning which can be observed from the hypervisor. Event monitoring and virus detection are offloaded to the VMM, so that attacks in the guest VM cannot reach the antivirus system. Furthermore, any modification is not required on the guest OS, which can simplify implementation of the system. Antivirus storm and rollback vulnerability issues are eliminated in VirtAV. In addition,

VirtAV can provide full life cycle antivirus protection for the guest VM no matter what state it is in. Functionality experiments of VirtAV show that the detection rate can achieve to 100% based on 3546 sample viruses from an open virus package. Performance experiments of VirtAV show that the acceptable overhead is introduced to the guest VM. For common desktop applications on Windows platform, performance degradation is negligible, and in general, the user experience on these applications is close to the native case without VirtAV.

Our ongoing and future work is to further minimize the amount of data to be scanned and reduce the time duration needed to scan virus. The directions include 1) recognizing pages with the same content among different VMs to reduce the number of page frames to be scanned, and 2) reconstructing VirtAV-engine as a host Linux kernel module to eliminate the overhead of context switches between Qemu and KVM.

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers and editors for their valuable suggestions and comments to improve the quality of this paper.

REFERENCES

- [1] J. O. Kephart and W. C. Arnold, "Automatic extraction of computer virus signatures," in *Proc. 4th Virus Bulletin International Conference*, 1994, pp. 178-184.
- [2] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333-340, 1975.
- [3] The ClamAV website. [Online]. Available: <http://www.clamav.net>.
- [4] The McAfee Antivirus website. [Online]. Available: <http://www.mcafee.com/>.
- [5] Trend Micro White Paper. "Changing the game for antivirus in the virtual datacenter," Sep. 2010.
- [6] Y. Xia, Y. Liu, H. Chen and B. Zang, "Defending against VM rollback attack," in *Proc. of Dependable Systems and Networks Workshops (DSN-W)*, 2012, pp. 1-5.
- [7] T. Garfinkel and M. Rosenblum, "When virtual is harder than real: security challenges in virtual machine based computing environments," in *Proc. of 10th Workshop on Hot Topics in Operating Systems*, 2005, pp. 20-20.
- [8] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proc. of the 2003 Network and Distributed System Security Symposium*, 2003, pp. 191-206.
- [9] X. Jiang, X. Wang and D. Xu, "Stealthy malware detection through VMM-based 'out-of-the-box' semantic view reconstruction," in *Proc. of 14th ACM Conf. on Computer and Communications Security*, 2007, pp. 128-138.
- [10] P. M. Chen and B. D. Noble, "When virtual is better than real," in *Proc. of 8th Workshop on Hot Topics in Operating Systems (HOTOS'01)*, 2001, pp. 133-138.
- [11] H. Xiong, Z. Liu, W. Xu and S. Jiao, "Libvmm: A library for bridging the semantic gap between guest OS and VMM," in *Proc. of 12th International Conference on Computer and Information Technology (CIT)*, 2012, pp. 549-556.
- [12] B. D. Payne, M. Carbone, M. Sharif and W. Lee, "Lares: an architecture for secure active monitoring using virtualization," in *Proc. of 29th IEEE Symposium on Security and Privacy*, 2008, pp. 233-247.
- [13] VMware vShield Endpoint. [Online]. Available: <http://www.vmware.com/products/vsphere/features/endpoint.html>.
- [14] Libguestfs website. [Online]. Available: <http://libguestfs.org/>.
- [15] Microsoft PE and COFF Specification. [Online]. Available: <https://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx>.
- [16] "Intel 64 and IA-32 Architectures Software Developer's Manual, Volume 3B: System Programming Guide Part 2." [Online]. Available: <http://www.intel.com/content/www/us/en/processors/architectures-software-developer-manuals.html>
- [17] atozvirus.rar. [Online]. Available: <http://yun.baidu.com/wap/link?uk=2852875414&shareid=3677790463&third=0>.
- [18] PCMark website. [Online]. Available: <http://cn.futuremark.com/benchmarks/pcmark>.
- [19] SysInternalsSuite website. [Online]. Available: <https://technet.microsoft.com/en-us/sysinternals/bb842062.aspx>.
- [20] M. Sharif, W. Lee, W. Cui and A. Lanzi, "Secure in-VM monitoring using hardware virtualization," in *Proc. of 16th ACM Conference on Computer and Communications Security*, 2009, pp. 477-487.
- [21] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai and P. M. Chen, "ReVirt: enabling intrusion analysis through virtual-machine logging and replay," *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 211-224, 2002.
- [22] J. R. Crandall, G. Wassermann, D. A. Oliveira, Z. Su, S. F. Wu and F. T. Chong, "Temporal search: detecting hidden malware timebombs with virtual machines," *ACM SIGOPS Operating Systems Review*, vol. 34, no. 5, pp. 25-36, 2006.
- [23] Y. Wang, D. Beck, X. Jiang and R. Roussev, "Automated web patrol with strider HoneyMonkeys: finding web sites that exploit browser vulnerabilities," in *Proc. of 13th Network and Distributed Systems Security Symposium*, 2006, pp. 1-15.
- [24] A. Dinaburg, P. Royal, M. Sharif and W. Lee, "Ether: malware analysis via hardware virtualization extensions," in *Proc. of 15th ACM Conference on Computer and Communications Security*, 2008, pp. 51-62.
- [25] M. Andreas, K. Christopher and K. Engin, "Exploring multiple execution paths for malware analysis," in *Proc. of 28th IEEE Symp. on Security and Privacy*, 2007, pp. 231-245.
- [26] G. Xiang, H. Jin, D. Zou and X. Chen, "Virtualization based security monitoring," *Journal of Software*, vol. 23, no. 8, pp. 2173-2187, 2012.



Hongwei Tang is a PhD candidate at the University of Chinese Academy of Sciences. He received his B.S. degree from Nankai University (China) in 2006 and M.S. degree from University of Chinese Academy of Sciences in 2009. Currently, he is a senior member of China Computer Federation (CCF) and a research associate at Institute of Computing Technology of Chinese Academy of Sciences (ICT, CAS). His research interests include cloud computing, virtual machine and operating system.



Shengzhong Feng received his Ph.D. degrees in computer science from Beijing Institute of Technology in 1997. He was employed by Institute of Computing Technology, Chinese Academy of Sciences. From 2005 to 2007, he investigated gene chip algorithm design as a visiting professor in University of Toronto, Canada. Currently, he is the executive director of the China Mathematics Software Association and a committee member of High Performance Computing Association.

He has published over 30 research papers, most of them are indexed by SCI/EI. His research interests include high performance computing, grid computing and bioinformatics.



Xiaofang Zhao received her Ph.D. degrees in computer science from Institute of Computing Technology, Chinese Academy of Sciences. Currently, she is the director of computer application research center in Institute of Computing Technology, Chinese Academy of Sciences. She is a senior member of China Computer Federation (CCF) and a member of Engineering and Technology Committee. Her research interests include computer architecture of next generation, security of

computer system and information security.



Yan Jin received the B.S., M.S., and Ph.D. degrees in computer science from Harbin Institute of Technology, Harbin, China, in 2001, 2003, and 2008, respectively. He was a research fellow in department of Electrical and Computer Engineering, University of Nevada, Las Vegas (UNLV) from 2008 to 2011. Since 2012, he has been an associate professor in Institute of Computing Technology, Chinese Academy of Sciences. His research interests include network security, cloud computing and cloud security.