Consistent Offline Update of Suspended Virtual Machines in Clouds

Consistent Offline Update of Suspended Virtual Machines in Clouds

Kenichi Kourai
Department of Computer Science and Networks
Kyushu Institute of Technology
Fukuoka, Japan
Email: kourai@ksl.ci.kyutech.ac.jp

Yuji Shiota
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
Email: rabbits114@ksl.ci.kyutech.ac.jp

***Abstract*—In Infrastructure-as-a-Service clouds, there exist many virtual machines (VMs) that are not used for a long time. For such VMs, many vulnerabilities are often found in installed software while VMs are suspended. If security updates are applied to such VMs after the VMs are resumed, the VMs easily suffer from attacks via the Internet. To solve this problem, offline update of VMs has been proposed, but some approaches have to permit cloud administrators to resume users' VMs. The others are applicable only to completely stopped VMs and often corrupt virtual disks if they are applied to suspended VMs. In addition, it is sometimes difficult to accurately emulate security updates offline. In this paper, we propose *OUassister*, which enables consistent offline update of *suspended* VMs. OUassister emulates security updates of VMs offline in a non-intrusive manner and applies the emulation results to the VMs online. This separation prevents virtual disks of even suspended VMs from being corrupted. For more accurate emulation of security updates, OUassister provides an emulation environment using a technique called *VM introspection*. Using this environment, it automatically extracts updated files and executed scripts. We have implemented OUassister in Xen and confirmed that the time for critical online update was largely reduced.**

***Keywords*-security updates, virtual machines, VM introspection, clouds**

I. Introduction

Infrastructure-as-a-Service (IaaS) clouds provide users with virtual machines (VMs). Users can create their VMs and install various software such as favorite operating systems and servers. While they can easily create VMs as necessary, many VMs are not used for a long time [1]. Unused VMs are usually stopped or suspended because only running VMs are charged in IaaS clouds. When a VM is stopped, the system running in the VM is shut down. In contrast, when a VM is suspended, its current state is saved to storage. This is analogous to hibernation of physical machines. Since it takes time to shut down the systems and boot stopped VMs and various caches for performance improvement are lost in VMs, it is desirable to suspend VMs if possible and resume them as fast as possible when necessary.

In either case, while VMs are not running, many vulnerabilities are often found in installed software. If such VMs start running after a long time, they easily suffer from attacks via the Internet. In traditional systems, security updates are applied after VMs are booted or resumed. However, this *online update* is at high risk because VMs are attacked while security updates are being applied online. Security updates are accumulated in proportion to a not-running period and it takes a long time to apply many security updates. To solve this problem, *offline update* of VMs has been proposed [1]–[3], but some of the approaches have to permit cloud administrators to temporarily boot or resume users' VMs to apply security updates [3]. This is often undesirable in public clouds. The other approaches are applicable only to *stopped* VMs [1], [2]. If they are applied to suspended VMs, virtual disks are often corrupted. In addition, it is sometime difficult to accurately emulate security updates offline only by mounting virtual disks of VMs [1].

In this paper, we propose *OUassister* for enabling consistent offline update of *suspended* VMs. OUassister emulates security updates of VMs offline in a non-intrusive manner, whereas it applies the emulation results to the VMs online. This separation between offline and online tasks prevents virtual disks of even suspended VMs from being corrupted and keeps the integrity of the virtual disks. Since this online task is minimum, the online update time can be much shorter than that in traditional online update. To apply security updates offline more accurately, OUassister provides an emulation environment using a technique called *VM introspection* [4]. Using VM introspection, it emulates a special filesystem providing system information and several system calls. Then, OUassister automatically extracts updated files by using the *union filesystem* [5] and executed scripts by hooking system calls. When VMs are resumed and become online, OUassister updates the virtual disks and executes extracted scripts *inside* the VM.

We have implemented OUassister in Xen [6]. OUassister constructs an emulation environment with Transcall [7], which we have developed for monitoring VMs from the outside using VM introspection. Transcall was extended so as to obtain information inside suspended VMs and save package scripts to be executed. Moreover, OUassister extracts updated files with aufs [8], which is one implementation of the union filesystem. Using OUassister, we confirmed that the apt command in Ubuntu could be executed offline and several packages could be updated correctly. In addition, we

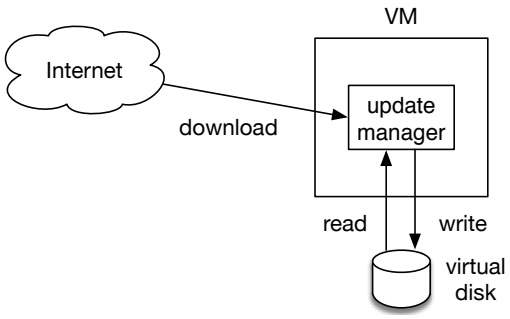

Figure 1. Online update.

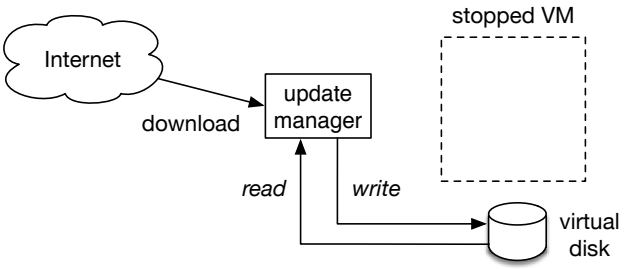

Figure 2. Traditional offline update.

showed that the online update time after a VM was resumed became much shorter.

The rest of this paper is organized as follows. Section II describes the issues in security updates of VMs and Section III proposes OUassister. Section V reports experimental results using OUassister. Section VI discusses related work and Section VII concludes the paper.

II. Security Updates of VMs

Security updates of the systems in VMs consist of two phases, as illustrated in Fig. 1. First, the update manager running in VMs downloads security updates, i.e., software packages, from the Internet. If one package depends on others, the update manager downloads those packages as well. Second, it extracts files from the packages and locates them in virtual disks of VMs. If some of the already installed packages are uninstalled, the update manager removes files that have been installed by those packages. In addition, it executes package scripts before and after the installation or removal of files if necessary. For example, the scripts start and stop servers and update databases. Since security updates are performed inside running VMs, this is called *online update*.

Unfortunately, there exist many unused VMs in IaaS clouds [1]. If unused VMs are running, the update manager has a chance to apply security updates. However, unused VMs are often stopped or suspended because only running VMs are charged in most of the IaaS clouds. The difference between stopped and suspended VMs is whether the state of a VM is saved or not. If a VM is stopped, the system inside the VM is shut down and the state of the VM is abandoned. In contrast, if a VM is suspended, its state such as CPUs, memory, and devices is saved to storage. The saved state is restored when a VM is resumed. Suspending VMs is more desirable than stopping them because suspended VMs can become online more quickly and keep peak performance just after resumption by reusing various caches such as the file cache inside VMs.

When VMs are booted or resumed after a long time, performing online update is at high risk because many vulnerabilities are usually found in software installed in the VMs. If attacks against found vulnerabilities are widely spread, the VMs suffer from the attacks immediately after they are connected to the Internet. Since VMs have to connect to the Internet and download security updates, it is difficult to prevent attacks via the Internet. If there are many security updates, it takes a long time to apply them and the probability of attacks becomes higher.

To solve this problem of online update, *offline update* of VMs has been proposed [1]–[3]. One approach is to update VMs in an execution environment isolated for security updates [3]. It downloads security updates to a local server in advance, boots or resumes VMs in the execution environment, and applies security updates by using the update managers running in the VMs. It enables secure updates because VMs are not connected to the Internet. In addition, it is applicable not only stopped VMs but also suspended VMs. However, users have to permit cloud administrators to boot or resume VMs and apply security updates. This is often undesirable in public IaaS clouds because users and cloud administrators belong to different organizations. Also, it is costly to construct and maintain the dedicated execution environment for security updates.

The other approach is to directly modify virtual disks of stopped VMs [1], [2], as illustrated in Fig. 2. This approach downloads security updates outside VMs *offline*. One tool [2] stores the downloaded security updates in the virtual disks. Then, it applies the stored updates just after VMs are booted and become online. This tool enables offline update of *stopped* VMs, but it is not applicable to *suspended* VMs. If it modifies virtual disks of suspended VMs, the disks are often corrupted. This is because the guest operating system in VMs caches filesystem state read from virtual disks. Unlike stopped VMs, such caches in memory are saved on VM suspension and restored on the resumption. Therefore, the integrity cannot be kept between the old caches and the modified virtual disks after VMs are resumed.

Another tool [1] not only downloads security updates but also applies them to the virtual disks offline as much as possible. Like the above tool, this tool cannot be used for suspended VMs because it can corrupt the virtual disks. In addition, this tool runs the update manager offline in an emulated environment, but the emulation is incomplete.

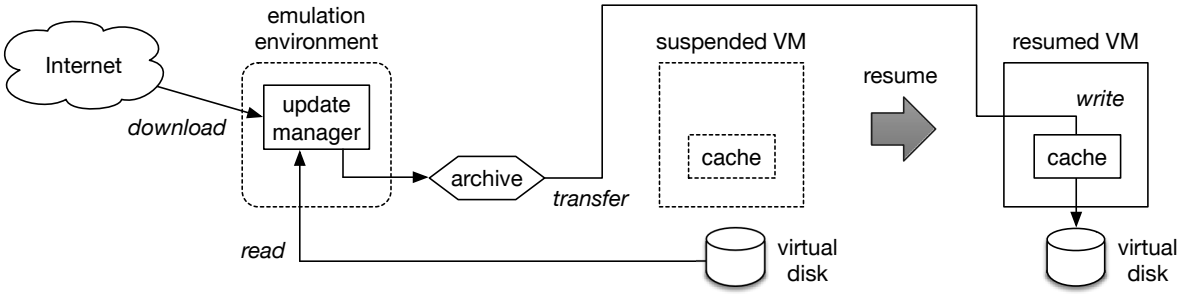

Figure 3. Offline update in OUassister.

It is reported that the tool failed to apply several security updates because the proc filesystem used in the emulation environment was different from that in VMs. The proc filesystem is dynamically generated by the operating system and returns system information.

III. OUASSISTER

To enable consistent offline update of *suspended* VMs, this paper proposes *OUassister*. Fig. 3 illustrates the procedure of security updates in OUassister. OUassister runs the update manager in its emulation environment *outside* a VM. It can run the update manager securely because the provided emulation environment is well maintained outside vulnerable VMs. The update manager first downloads security updates from the Internet and then applies those security updates *offline*. At this time, OUassister emulates security updates to a VM, instead of directly applying them. It records information on files updated by the update manager. Since it prevents the update manager from modifying a virtual disk of the VM, the integrity of the virtual disk is kept even if the VM is suspended and various caches are saved. Even if this offline task takes time, the risk of the updated VM does not increase because the VM are kept offline during the update emulation.

When the VM is resumed and becomes *online* later, OUassister just applies the emulation results to the VM. It first transfers information on updated files to the VM. Then, it creates, modifies, and removes files in the virtual disk *inside* the VM. Since the VM itself updates its virtual disk using various internal caches, the integrity of the virtual disk is kept. If security updates contain scripts to be executed before or after updating files, OUassister executes these scripts in an appropriate timing inside the VM. The scripts are recorded when the update manager applies security updates in the emulation environment offline. Since the update manager does not run online in the VM, this online task is much less than traditional online update.

Using *VM introspection* [4], OUassister provides a more accurate emulation environment for security updates of VMs offline. VM introspection is a technique for securely accessing the internal state of VMs from the outside. This emulation environment provides the same filesystems including the proc filesystem as well as regular files and directories. The proc filesystem is a pseudo filesystem dynamically generated by the operating system kernel and strongly depends on the system running in a VM. For example, the proc filesystem contains information on the configuration of the operating system and running processes. Therefore, OUassister dynamically creates the proc filesystem by analyzing data structure in the kernel memory of a VM. In addition, OUassister emulates several system calls executed by the update manager to return system information in a VM.

OUassister automatically extracts files updated by the update manager using the *union filesystem* [5] in the emulation environment. The union filesystem can create a layered filesystem by stacking multiple filesystems. OUassister stacks an empty filesystem on top of the filesystem used in a VM. Then, it records created, modified, and removed files and directories in the upper layer. In addition, OUassister automatically extracts scripts executed by the update manager by hooking system calls. Since the update manager issues a system call when it executes a script contained in security updates, OUassister saves a script file specified by the system call.

As such, OUassister can achieve mostly offline update of suspended VMs by combining offline emulation with online application. This idea is similar to the previous work [1], which updates virtual disks of VMs offline and executes only scripts online. One difference is that the offline task in OUassister is *not intrusive* to VMs at all. OUassister does not update any files in virtual disks from the outside of VMs to prevent virtual disks of suspended VMs from being corrupted. Also, it does not have to permit cloud administrators to update users' VMs although it needs the permission only for reading the storage and memory of the VMs. If it is not desirable to grant even this permission, remote VM introspection [9] can be applied to run the update manager in trusted remote hosts. The other difference is that the execution environment emulated in OUassister is more similar to that inside VMs. This can reduce the possibility that the update manager fails to apply security updates by the environmental differences.

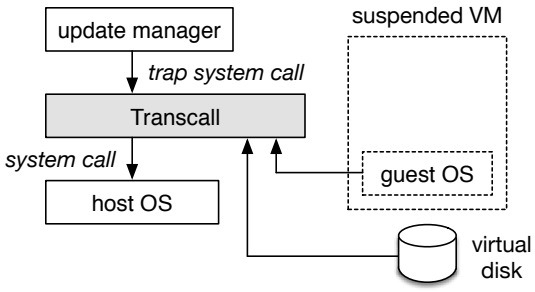

Figure 4. Update emulation using Transcall.

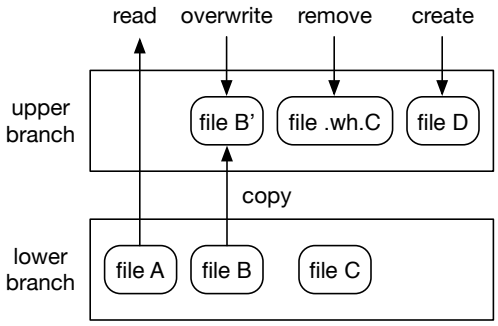

Figure 5. Extracting files from the upper branch.

IV. Implementation

We have implemented OUassister for Ubuntu Linux running on top of Xen. We ran the update manager in the privileged VM called Dom0 using the emulation environment provided by OUassister. We assumed the apt command in Ubuntu as an update manager, but we believe that OUassister can apply the other update managers, e.g., yum in CentOS.

A. Accurate Emulation Environment

OUassister constructs an execution environment for accurately emulating security updates using *Transcall* [7]. Transcall is a tool that we have developed for monitoring VMs outside them using VM introspection, as shown in Fig. 4. It provides the *shadow filesystem* for transparent access to the virtual disk of a VM. Using the shadow filesystem, the update manager can access files and directories in the VM as if it ran inside the VM. In addition, Transcall provides the *shadow proc filesystem* for accessing system information on the guest operating system in the VM, e.g., processes and networks. The shadow proc filesystem is created by analyzing the kernel memory of the VM and obtaining necessary data in the guest operating system. This filesystem is implemented using FUSE [10]. Since Transcall does not support suspended VMs, we run Transcall after resuming a VM but before restarting it. When the emulation of security updates is completed, that almost resumed VM is simply destroyed.

Also, Transcall emulates some of the system calls issued by the update manager running in the emulation environment. For example, when the update manager issues the uname system call for obtaining information on the guest operating system, Transcall obtains data in the guest operating system and returns it. For most of the system calls, e.g., related to filesystems and networks, Transcall just redirects the invocation to the host operating system in Dom0 on which Transcall is running.

B. Extracting Updated Files

To extract files and directories updated by the update manager, OUassister uses *aufs* [8]. Aufs enables multiple directories called branches to be overlaid transparently. It merges files and directories of branches and provides files of an upper branch in a higher priority. OUassister mounts the virtual disk of a VM to a directory and configures that branch to read-only. In contrast, it configures a branch for maintaining updated files to read-write. Then, it gives a higher priority to the read-write branch. As a result, the read-only branch becomes lower and the read-write one becomes upper.

When the update manager writes data to the directory mounted with aufs, the data is always written to the upper branch, as shown in Fig. 5. If a target file does not exist, aufs creates the file in the upper branch. If the file exists in the lower branch, aufs copies the original file to the upper branch. In any case, the lower branch is not modified and the integrity of the virtual disk is kept. When the update manager reads a file, aufs provides a file in the upper branch if an updated file exists. Otherwise, aufs provides a file in the lower branch. As such, OUassister can give an illusion of updating the virtual disk to the update manager.

When the update manager removes a file or directories in the lower branch, aufs creates a special file called a *whiteout* file in the upper branch. A whiteout file indicates that the corresponding file in the lower branch is removed. The name of a whiteout file is `.wh.`<*filename*> where <*filename*> is the name of a removed file. If there is a whiteout file in the upper branch, the update manager cannot access the corresponding file or directory in the lower branch. As such, OUassister can give an illusion of removing files or directories in the virtual disk. Since files or directories in the lower branch are not really removed, the integrity of the virtual disk is kept.

After security updates are applied to the directory mounted with aufs, updated files and directories are extracted in the upper branch. First, OUassister searches all the directories used for the upper branch and obtains the paths of whiteout files. Then, it creates a list of removed files and directories and removes the whiteout files. Next, OUassister creates an archive of updated files and directories and compresses it.

C. Extracting Scripts

During update emulation, OUassister also extracts executed scripts, e.g., maintainer scripts in Ubuntu packages and package-specific scriptlets in RPM packages. Such package scripts are executed before or after installation and uninstallation. OUassister does not execute package scripts offline but saves them. To achieve this, it hooks the execve system call issued by the update manager using Transcall and obtains the path in which a script is located and the arguments to the script. If the path is for a package script, OUassister saves that script file with its arguments. Then, it rewrites the path in the execve system call to a dummy program, i.e., `/bin/true`, to prevent the script from being executed offline.

This script extraction is a task specific to the package management system. For Ubuntu packages, OUassister extracts four types of scripts: pre-installation and pre-removal scripts and post-installation and post-removal scripts. The path of a script is <*package_name*>.<*script_type*> in the package directory or <*script_type*> in the temporary directory. <*script_type*> is `preinst`, `prerm`, `postinst`, or `postrm`.

D. Applying Emulation Results

OUassister applies the results of offline emulation to the virtual disk of a target VM just after the VM is resumed and becomes online. First, OUassister transfers the archive of updated files and directories to the VM using the local network. Then, it decompresses the archive, extracts files, and copies them to the virtual disk inside the VM. Second, OUassister transfers the list of removed files and directories to the VM. In the VM, it removes files and directories from the virtual disk on the basis of the list.

Also, OUassister executes saved package scripts while applying emulation results. The scripts are executed with saved arguments in a saved order. Before applying the updates in files and directories, OUassister executes saved pre-installation and pre-removal scripts. For example, these scripts stop servers and unload kernel modules. After applying the updates in files and directories, OUassister executes saved post-installation and post-removal scripts, which start servers and load kernel modules, for example.

To prevent the resumed VM from being attacked while applying emulation results, OUassister temporarily disconnects the VM from the Internet. It configures the local firewall of the host operating system in Dom0 so that the VM cannot communicate with the Internet, as illustrated in Fig. 6. Note that it allows communication only with OUassister because OUassister needs to transfer emulation results to the VM. After applying emulation results, OUassister removes the added firewall rules. As a result, the VM cannot communicate with the Internet for a while just after resumed, but it is acceptable if the time of applying emulation results is short enough.

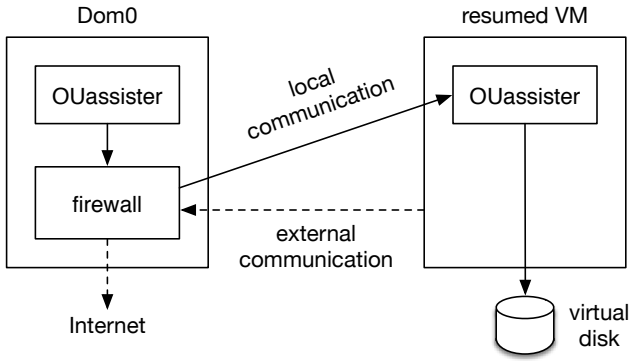

Figure 6. Online application in an isolated network.

V. Experiments

We examined the effectiveness of offline update in OUassister. For comparison, we executed traditional online update inside a VM, which we call *in-VM update*. In the experiments, we used a PC with an Intel Xeon E5630 processor, 6 GB of memory, 250 GB of HDD, gigabit Ethernet. We ran Xen 4.1.3 and Ubuntu 12.04 LTS in Xen's Dom0. We used a VM with one virtual CPU, 512 MB of memory, and 4 GB of virtual disk. We also ran Ubuntu 12.04 LTS in the VM. During the offline task in OUassister, we suspended the VM.

We executed four types of software updates using the apt command: (1) installing the nginx web/proxy server, (2) uninstalling it, (3) updating the OpenSSL library, and (4) updating the package list. Since the nginx-full package depended on five packages, its installation and uninstallation caused those of six packages in total, respectively.

A. Update Time

First, we measured the online update time, which is the time needed for applying software updates inside a running VM. For OUassister, the online update time is the time for only the online task performed after a VM is resumed. For traditional in-VM update, it is the time for the entire security updates including package download. As shown in Fig. 7, OUassister could reduce the online update time successfully. In-VM update took 10–23 seconds to apply each software update, while OUassister required only 2.5–5.9 seconds. The online update time in OUassister was 11–57% of in-VM update.

Next, we measured the total update time, which is the sum of the offline and online update time, when we used OUassister. The time with its breakdown is shown in Fig. 8. Emulating software updates and archiving updated files were done offline, while applying emulation results was done online. The emulation time is the time for emulating software update offline and it was 7.7–12 seconds. One reason why the time was shorter only in uninstalling nginx is that uninstallation did not need to download packages

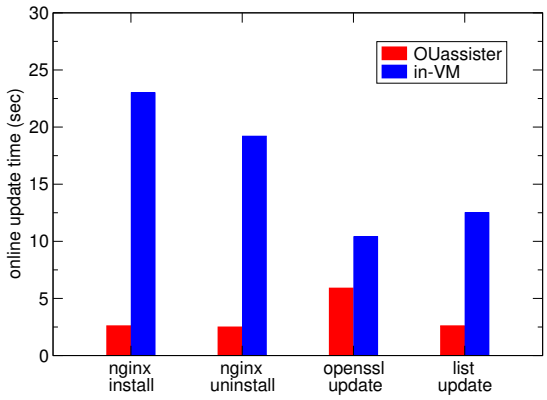

Figure 7. The online update time.

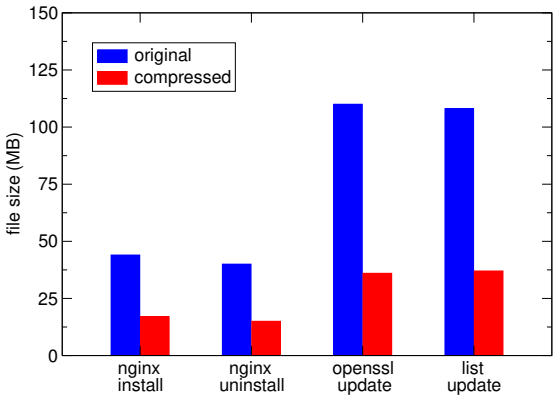

Figure 9. The size of updated files and directories.

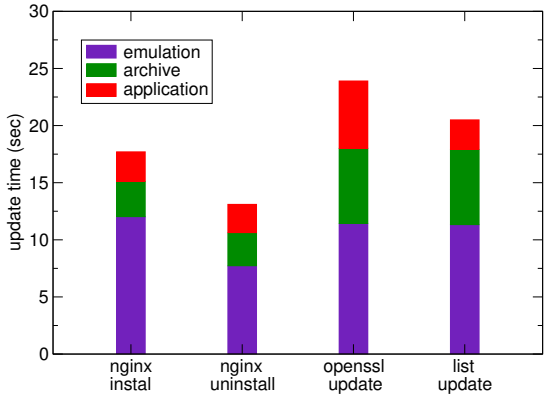

Figure 8. The breakdown of the total update time in OUassister.

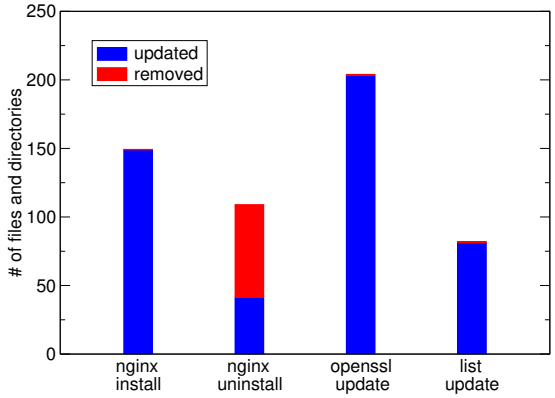

Figure 10. The number of updated files.

from the Internet. The archive time is the time for creating a compressed archive of files and directories updated by the update manager and it was proportional to the file size. As shown in Fig. 9, the size of files modified by installing and uninstalling nginx was much smaller than that of files modified by updating openssl and the package list.

Compared with Fig. 1, the total update time in OUassister became shorter than that in in-VM update when we installed and uninstalled nginx. This is because applying software updates inside the VM suffered from virtualization overhead, particularly in terms of its disk and network. In contrast, the total time was longer in OUassister when we updated openssl and the package list. Possible reasons are the overhead of handling larger archives and larger emulation overhead.

B. Extracted Files and Scripts

We examined the files and scripts extracted after the results of update emulation in OUassister. Fig. 10 shows the number of updated files and directories. Updated files included the data, cache, and log of the package management system and the database for man as well as files contained in the updated packages. While no file was removed in the nginx installation, one file was removed in the openssl update. For nginx, the total number of updated and removed files in uninstallation was less than that in installation. Compared with Fig. 9, the total size of updated files did not depend on the number.

Fig. 11 shows the breakdown of scripts extracted when OUassister emulated software updates. The number of scripts depended on the number of updated packages. Six packages were installed and uninstalled for nginx, respectively, while only one package was updated for openssl. Interestingly, when nginx was uninstalled, not only pre- and post-removal scripts but also post-installation scripts were extracted. This is because the post-installation script of man-db was executed and its database was updated.

C. Accessed Files in the Proc Filesystem

To show that our accurate emulation environment for the update manager is necessary, we examined accessed files in the shadow proc filesystem, which provides dynamically generated system information in a VM. Table I shows the files accessed by the apt command. For the first two files and one directory, the contents depend on the kernel configuration. Since the kernel can be different between the inside and

Table I
FILES IN THE SHADOW PROC FILESYSTEM ACCESSED BY APT.

file	description
`/proc/filesystems`	a list of filesystems supported by the kernel
`/proc/sys/kernel/ngroups_max`	the maximum number of process's group memberships
`/proc/sys/net/ipv6/`	information on IPv6
`/proc/[pid]/stat`	status information on the process with pid
`/proc/1/root/`	a symbolic link to the init process's root directory
`/proc/self/`	information on the process accessing the proc filesystem

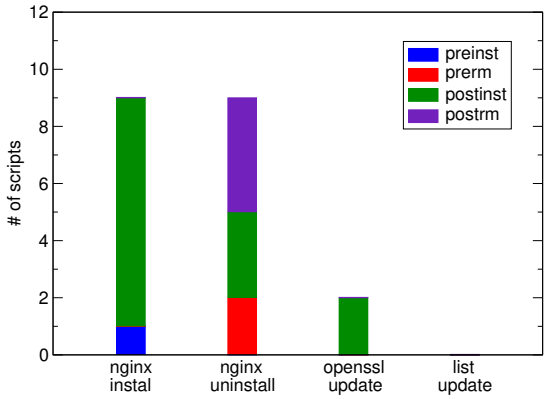

Figure 11. The breakdown of extracted scripts.

the outside of a VM, emulation is required for the correct behavior of the apt command. For the fourth file, it is also necessary that the apt command examines processes running inside a VM. In contrast, the remaining two directories do not require emulation. The root directory of the init process does usually not change. Since the apt process does not really run inside a VM, `/proc/self` should point to the process running outside a VM. In fact, Transcall creates a symbolic link of `/proc/self` to information on the apt process outside a VM.

VI. RELATED WORK

Ivanti Patch for Windows [2] (formerly known as Shavlik NetChk Protect) enables offline update of VMs. It directly writes security updates in the virtual disk of a VM. When the VM starts running, it automatically executes the security updates stored in the virtual disk. This tool requires applying security updates online, but it can download security updates offline. This leads to less online update time and decreases the attack surface during security updates. In addition, this tool can prevent attacks by disconnecting a vulnerable VM from the Internet during security updates. However, it is not applicable to suspended VMs because the integrity of the virtual disk cannot be kept.

Nüwa [1] is another tool for offline update of VMs. It directly updates the virtual disk of a VM by applying security updates outside the VM. In addition, it enables offline execution of the scripts contained in security updates as much as possible. First, it classifies scripts into ones that can be executed offline and the others. It rewrites scripts that cannot be executed offline and executes modified ones. If scripts cannot be rewritten appropriately, Nüwa executes them online. However, this tool has to give cloud administrators permissions for updating the virtual disks of VMs. Worse, it cannot be applied to suspended VMs.

Microsoft Virtual Machine Servicing Tool [3] runs VMs in a dedicated execution environment and applies security updates inside the VMs. In the execution environment, the local server downloads security updates whenever new ones are issued. When updating not-running VMs, the tool moves the VMs to the execution environment and boots or resumes them. The VMs download security updates securely from the local server and update themselves. After security updates are applied, the tool stops or suspends the VMs again and moves them back to the original host. This tool can be also applied to suspended VMs, but cloud administrators need permission for accessing users' VMs.

Aufs-based upgrade [11] in Ubuntu can simulate and test release upgrades. It creates a writable overlay on top of the root directory except for the home directory. If the upgrade does not work well, it is easily canceled by simply removing the upper branch of aufs. However, this tool does not have a mechanism for merging the upper and lower branches to eliminate the overhead of aufs after a successful upgrade. Solaris Live Upgrade [12] can minimize the downtime and risk when security updates are applied and the system is upgraded. It creates a copy of the root filesystem and modifies files in it. Using ZFS, the copy is performed only for modified files by the copy-on-write mechanism. If the system does not work well after updates, it can be reverted to the previous state immediately using the original root filesystem.

The overlay filesystem [13] is newer implementation of the union filesystem. Recently, it is used as a storage backend of Docker containers [14]. To simplify the implementation and optimize the performance, it limits the number of stacked layers only to two. Since OUassister requires only two layers, i.e., the filesystem used in a VM and that for updated files, it can use the overlay filesystem instead of aufs. Using the overlay filesystem can reduce the time needed for extracting files from security updates offline.

VM introspection was first proposed for intrusion detection in Livewire [4]. Livewire provides operating system

libraries to run intrusion detection systems customized for it. Many systems using VM introspection have been proposed, but only VMST [15] and our Transcall [7] can run existing tools without modification for monitoring a VM. VMST runs monitoring tools in one dedicated VM for each target VM and redirects introspection-related memory access inside the dedicated VM to the memory of the target VM. Therefore, it needs to prepare as many dedicated VMs as simultaneously updated VMs. In contrast, Transcall needs only one dedicated process for emulating trapped system calls and one FUSE process for the shadow proc filesystem. When many VMs are updated simultaneously, the architecture of Transcall is preferable.

VII. Conclusion

In this paper, we proposed OUassister for enabling consistent offline update of suspended VMs. OUassister emulates security updates of VMs offline more accurately using VM introspection. Then, it extracts updated files by using the union filesystem and executed scripts by hooking system calls. When VMs are resumed and become online, OUassister just applies the emulation results to the VMs. Since OUassister does not modify virtual disks of VMs at all while the VMs are suspended, the integrity of the disks is kept. We have implemented OUassister in Xen, Transcall, and aufs. We confirmed that OUassister enabled successful offline update of suspended VMs and achieved shorter online update time.

One of our future work is to examine the performance degradation of VMs due to the application of many security updates on VM resumption. We expect that the performance degradation is much smaller than online update. For this purpose, we need to investigate how many security updates are created in certain periods. Another direction is to apply OUassister to suspended VMs in other virtualized systems, e.g., KVM, vSphere, and Hyper-V. Since we have already developed Transcall for KVM [16], we could port OUassister to KVM easily. In addition, it is desirable to integrate the technique of script rewrite used in Nüwa [1] into OUassister. Such integration can reduce the online execution time of package scripts.

Acknowledgment

The research results have been achieved by the "Resilient Edge Cloud Designed Network (19304)," the Commissioned Research of National Institute of Information and Communications Technology (NICT), Japan.

References

[1] W. Zhou, P. Ning, X. Zhang, G. Ammons, R. Wang, and V. Bala, "Always Up-to-Date: Scalable Offline Patching of VM Images in a Compute Cloud," in *Proc. Annual Computer Security Applications Conf.*, 2010, pp. 377–386.

[2] Ivanti, "Patch for Windows Servers," https://www.ivanti.com/products/patch-management.

[3] Microsoft Corporation, "Virtual Machine Servicing Tool (VMST) 2012," https://technet.microsoft.com/en-us/library/jj149757.aspx.

[4] T. Garfinkel and M. Rosenblum, "A Virtual Machine Introspection Based Architecture for Intrusion Detection," in *Proc. Network and Distributed Systems Security Symp.*, 2003, pp. 191–206.

[5] J. Pendry and M. McKusick, "Union Mounts in 4.4BSD-lite," in *Proc. USENIX 1995 Technical Conf.*, 1995, pp. 25–33.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield, "Xen and the Art of Virtualization," in *Proc. Symp. Operating Systems Principles*, 2003, pp. 164–177.

[7] T. Iida and K. Kourai, "Transcall," http://www.ksl.ci.kyutech.ac.jp/oss/transcall/.

[8] J. Okajima, "Aufs5 – Advanced Multi Layered Unification Filesystem Version 5.x," http://aufs.sf.net.

[9] K. Kourai and K. Juda, "Secure Offloading of Legacy IDSes Using Remote VM Introspection in Semi-trusted Clouds," in *Proc. IEEE Int. Conf. Cloud Computing*, 2016, pp. 43–50.

[10] N. Rath, "libfuse: The Reference Implementation of the Linux FUSE (Filesystem in Userspace) Interface," https://github.com/libfuse/libfuse.

[11] M. Vogt, "AufsBasedUpgrades – Ubuntu Wiki," https://wiki.ubuntu.com/AufsBasedUpgrades.

[12] Oracle Corporation, "How to Upgrade and Patch with Oracle Solaris Live Upgrade," Oracle White Paper, 2010.

[13] M. Szeredi, "Overlay Filesystem," https://www.kernel.org/doc/Documentation/filesystems/overlayfs.txt.

[14] Docker, Inc., "Docker: Build, Ship, and Run Any App, Anywhere," https://www.docker.com/.

[15] Y. Fu and Z. Lin, "Space Traveling across VM: Automatically Bridging the Semantic Gap in Virtual Machine Introspection via Online Kernel Data Redirection," pp. 586–600, 2012.

[16] K. Kourai and K. Nakamura, "Efficient VM Introspection in KVM and Performance Comparison with Xen," in *Proc. Pacific Rim Int. Symp. Dependable Computing*, 2014, pp. 192–202.