An Empirical Study on the Software Integrity of Virtual Appliances: Are You Really Getting What You Paid For?

Jun Ho Huh
University of Illinois at Urbana-Champaign, USA
jhhuh@illinois.edu

Mirko Montanari
University of Illinois at Urbana-Champaign, USA
mmontan2@illinois.edu

Derek Dagit
University of Illinois at Urbana-Champaign, USA
dagit@illinois.edu

Rakesh B. Bobba
University of Illinois at Urbana-Champaign, USA
rbobba@illinois.edu

Dong Wook Kim
University of Illinois at Urbana-Champaign, USA
kim628@illinois.edu

Yoonjoo Choi
Dartmouth College, USA
yoonjoo@cs.dartmouth.edu

Roy Campbell
University of Illinois at Urbana-Champaign, USA
rhc@illinois.edu

ABSTRACT

Virtual appliances (VAs) are ready-to-use virtual machine images that are configured for specific purposes. For example, a virtual machine image that contains all the software necessary to develop and host a JSP-based website is typically available as a "Java Web Starter" VA. Currently there are many VA repositories from which users can download VAs and instantiate them on Infrastructure-as-a-Service (IaaS) clouds, allowing them to quickly launch their services. This marketplace, however, lacks adequate mechanisms that allow users to a priori assess whether a specific VA is really configured with the software that it is expected to be configured with. This paper evaluates the integrity of software packages installed on real-world VAs, through the use of a software whitelist-based framework, and finds that indeed there is a lot of variance in the software integrity of packages across VAs. Analysis of 151 Amazon VAs using this framework shows that about 9% of real-world VAs have significant numbers of software packages that contain unknown files, making them potentially untrusted. Virus scanners flagged just half of the VAs in that 9% as malicious, demonstrating that virus scanning alone is not sufficient to help users select a trustable VA and that a priori software integrity assessment has a role to play.

Categories and Subject Descriptors

D.4.6 [**Software**]: OPERATING SYSTEMS—*Security and Protection*

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ASIA CCS'13, May 8–10, 2013, Hangzhou, China.
Copyright 2013 ACM 978-1-4503-1767-2/13/05 ...$15.00.

General Terms

Security, Experimentation, Measurement, Verification

Keywords

IaaS, Virtual appliances, Whitelists, Software integrity

1. INTRODUCTION

Cloud computing has become popular over the years as organizations have tried to reduce the complexities and costs associated with deploying and maintaining an internal IT infrastructure. Infrastructure-as-a-Service (IaaS) cloud model [24], where users can build and configure virtual computing infrastructure by renting computing platform resources (e.g., [1, 6]) in the form of virtual machines, is a popular cloud computing service model. On these rented virtual machines, users have administrative privileges to install any piece of software they want and can configure them to meet their needs. Many services (e.g., [2, 10, 4, 5, 8]) that either provide preconfigured virtual machine disk images or enable sharing of such virtual machine disks have emerged to support the IaaS paradigm.

Virtual machine (VM) disk images that are preconfigured with the necessary software to support specific workflows of functions are known as *virtual appliances* (VAs), and services providing repositories of VAs are referred to as *appliance stores*. A VM image with a Tomcat web server fully configured to host Java web services and use the underlying MySQL database is an example of a VA. In the rest of the paper, VA and VM image will be used interchangeably. The benefits of this publisher-consumer model (see Figure 1) are clear: consumers enjoy the convenience of downloading a VA that suits their needs and launching a service quickly; and publishers can get paid for providing the VA[1]. This market-

[1]Other models exist in which publishers provide the virtual machine images to promote their software, or promote the use of their IaaS infrastructure, or simply share their images for the common good. All of these settings provide benefits, tangible or otherwise, to both consumers and publishers.

place, however, lacks adequate mechanisms that allow users to a priori assess whether a specific VA is correctly equipped with the software packages that it claims to contain. This paper sheds light on the integrity of software packages in real-world VAs, and demonstrates a clear need for a mechanism to assess their software integrity.

There are many well-known security challenges associated with the cloud computing paradigm (e.g., [19, 11, 27, 14, 29, 28]), and the security implications of the publisher-consumer model associated with VA stores (e.g., [32, 17]) are an important sub-class. A malicious publisher could install malware in a VA and attempt to read private information while a consumer uses the VA. A careless publisher might unintentionally publish a VA that is infected with a virus, is configured insecurely (e.g., running an unpatched version of an operating system, or with unintentional backdoors), or contains sensitive private information. Further, an irresponsible publisher could publish an incomplete VA that is missing integral software packages that it claims to have. Some software packages might be partially installed (i.e., missing critical files) or have files that are modified (i.e., different from the version provided by the software vendor). VAs with such partially installed or modified packages will not meet the expectations of VA consumers, who would expect the VAs to be configured with all the packages claimed by publishers, and that those packages are unmodified (unless otherwise noted). Currently, checking the integrity and identify of the installed software packages is a time-consuming process that must be performed manually by the consumer after the instantiation of the VA. Such information, if available during the selection process, could provide early insight into whether a VA is suitable and can be trusted. Some VAs come with build-scripts and logs, but without any guarantees that these logs are correct and accurate. The same is true for VA publishers and store providers: they lack mechanisms to verify the integrity of VA software before allowing them to be published, leading to security and quality assurance problems in appliance stores.

This paper presents an empirical study on the integrity of software packages in real-world VAs, assessed using a whitelist-based framework, showing that there is a significant variance in the software integrity of software packages across VAs. Our findings show that about 9% of the evaluated real-world VAs have significant software integrity issues and would probably not meet consumers' expectations. Surely, consumers should be informed about such VAs when selecting a VA. We would like to point out that while a well-configured VA with authentic software packages can still be susceptible to software vulnerabilities that are present in legitimate packages, the whitelisting approach can be used to avoid VAs with partially installed or modified packages[2] thus mitigating risk for consumers.

Whitelist-based approaches are hard to realize in general computing environments, given the variety and vast amount of existing software [20]. Our hypothesis is that in the appliance store model, a majority of consumers use fairly well-known, popular pieces of software, and thus the whitelist sizes will be more manageable. To test this hypothesis, we constructed a whitelist for our sample VAs and indeed found that the whitelist size growth decelerates.

[2]Note that there is a legitimate need for modified or customized software packages. Our focus is on VAs meant for well established workflows with standard software stacks. None of the VAs that we found to have "modified" software in our analysis claimed to be custom or modified VAs in their description.

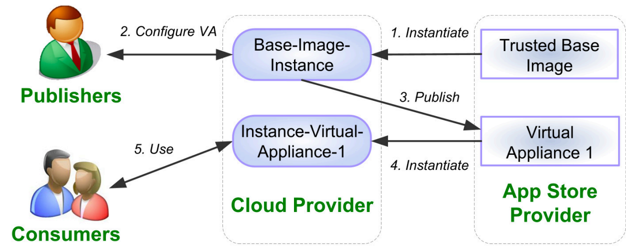

Figure 1: A typical virtual appliance store

Contributions: The key contributions of this paper are (1) an empirical evaluation of the software integrity in real-world VAs, showing that there is a *significant variation*, (2) demonstration of the *usefulness* of assessing the software integrity in VAs to help customers choose correctly configured VAs, and (3) demonstration of the *feasibility* and *scalability* of using whitelists for integrity assessment.

Organization: Section 2 covers related work. Section 3 provides an overview of the assessment framework used for analyzing real-world VAs in Section 4. Section 5 discusses the security implications. Our conclusions and future work are in Section 6.

2. RELATED WORK

Over the last few years, researchers and industry practitioners have been looking at challenges associated with cloud computing, including security and reliability challenges [19, 11, 26, 15, 27, 32, 14, 28, 29, 34, 16, 18, 33]. Here, we focus our discussion on past work [26, 32, 34, 17, 21] that addresses security risks and challenges associated with VA stores and with the management of VAs in general, which is most relevant to our work.

Reimer et al. [26] address the challenges posed by the sprawl of virtual machine images that need to be mounted for maintenance tasks. Specifically they propose Mirage Image Format (MIF), which simplifies the maintenance and administration process. To our knowledge, Wei et al. [32] are the first work to identify the security challenges and risks associated with the VA store model and with managing VAs in general. They proposed an image management framework with access control for VAs, filtering to remove unwanted information (e.g., browsing history), provenance tracking of images for accountability, and image maintenance (e.g., virus scanning), and argues for the benefits of such a system. In a later work [34], a scalable offline patching tool called *Nüwa*, which leverages MIF to significantly reduce the overhead of patching, was proposed. Accountability through provenance tracking, virus and malware scanning, and software patching are good security hygiene practices and do provide some assurances to the consumer regarding the security and trustworthiness of the VA. However, they alone are not sufficient. Accountability is after the fact and may come too late for some consumers, and the proportion of new or near zero-day malware that is detected through scanning is not very high [12]. In contrast, our approach provides more explicit information regarding a VA's software config-

uration to the consumer and is complementary to the aforementioned work.

Bugiel et al. [17] analyzed over a thousand Amazon Machine Images (VAs for the Amazon Elastic Compute Cloud) and found sensitive information (e.g., keys and credentials) and SSH backdoors inadvertently left in the VAs. To mitigate the risks to publishers and consumers they suggest countermeasures that include running filters and scanners on VAs and use of ratings based on certain verifiable properties and reputation, but they do not elaborate on the properties of interest or the rating system. There are in fact appliance stores that provide ratings for VAs [8]. However, the primary basis for the ratings seems to be user opinion, which, while quite useful, may not be objective, and may be based on many factors, of which software integrity and security are only a part. Furthermore, in ratings-based systems that are based on reputation and user feedback, it takes time for VAs to gather ratings and publishers to earn reputation, and it will be easy for brand-name providers to build their reputation even though many of the smaller players may be providing VAs that are as good or better than their well-known counterparts. In contrast, our software integrity assessment approach provides a means to rate VAs before publishing them, and it could be used either on its own or as part of a larger ratings ecosystem that also considers reputation of provider and user feedback.

Jayaram et al. [21] undertook an empirical study of similarity between VAs to enable efficient design of VA management systems. Their study considered block-level similarity in VAs to enable efficient design of de-duplication schemes, reduce storage, and improve image distribution, among other things. Our work focuses on use of similarity in VMs at the software or file level to gauge the size of whitelists that need to be maintained by a VA store provider. Moreover, to the best of our knowledge, we are the first group to analyze comprehensively the software integrity of VAs used in the real world.

Techniques based on file analysis similar to the one we used have been used previously in the context of operating systems. Seminal work from Kim et al. [23] and Vincenzetti et al. [31] introduced mechanisms for detecting changes in critical files in the system that might indicate the presence of rootkits or of other illicit modifications. These techniques are based on comparing files in the disk with a previously-generated hash of the file to detect changes. More recent work [25] extended such approaches to perform the analysis in a virtual machine environment, so that modifications to the files cannot be hidden even when the kernel is compromised. The goal of such techniques is to identify at *runtime* malware or malicious modifications to critical files by detecting changes from a trusted configuration (i.e., the initial configuration of the system). While we use similar file integrity based techniques, our focus is on analyzing the software integrity of a VA to assess the “initial” configuration. Further, our analysis discusses ratings for assessed software in contrast to earlier efforts that stop at detecting changes.

3. SOFTWARE INTEGRITY ASSESSMENT FRAMEWORK

To study the integrity of software packages in real-world VAs, we designed an integrity assessment framework based on software whitelisting techniques. At the heart of the framework is a Configuration Resolver (described in Section 3.1) that uses a VM Configuration Verification Tool (VM-CVT) to generate a “verification report” on a given VA (see Figure 2). Those tools use information provided by publishers and by software producers to verify the integrity of the software installed on the system, and to identify any modified or missing files. Based on this information, we introduce a simple rating system to score the installed software. The verification reports provide consumers with explicit information about the configuration of the system that they can use to gauge its suitability without having to download and instantiate the VA. Providers can use the verification reports to ensure that published VAs respect basic requirements by specifying policies. For example, a cloud provider could create a more trustable store by introducing a policy stating that all of the installed software and updates need to be fully verified. Such policies are discussed further in Section 5.3 as means to enhance VA security.

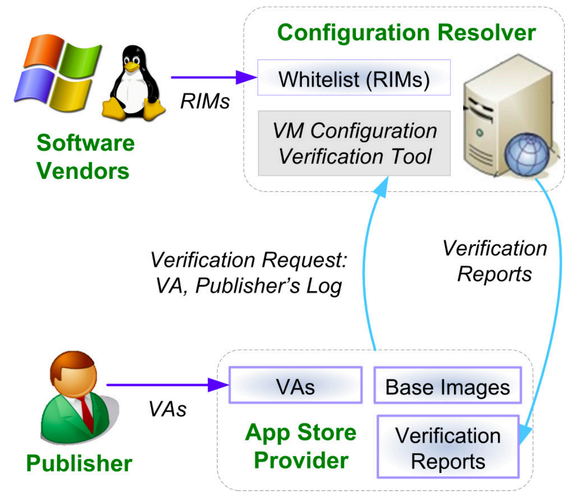

Figure 2: Software integrity assessment framework overview

Our integrity validation is based on reference integrity measurements (RIMs) [30] published by the vendors for their software. The RIMs contain the signed hash values of all the files in the software and metadata, including a description of each of them. This is a common software practice, and many software vendors already provide signed checksums for files (e.g., `rpm` packages contain MD5 checksums for the files), allowing users to verify the file integrity. In fact, the U.S. National Institute of Standards and Technology (NIST) maintains a software registry, called the National Software Reference Library (NSRL) [7], that provides RIMs for software. Bit9, a private security company, also maintains a massive software registry [3].

The framework can take advantage of additional information made available from publishers and from software vendors. In particular, we take advantage of the fact that VAs are often generated from trusted base images made available from the repository provider (e.g., Amazon Linux AMIs [1]) to speed up the analysis. This technique also allows a provider to whitelist implicitly proprietary software without having to register with third party Resolver. Moreover, a *log* describing the list of all software (and updates) installed on a VA, as well as the version information at the time of

VA publication, can be generated by the publishers and submitted with the VA to simplify the analysis. If publishers submit fabricated logs (e.g., lying about the installed packages), their VAs will get low integrity scores as the installed files will not match the RIMs and will appear unreliable.

Software vendors can provide additional metadata associated with the RIMs for identifying the file types (e.g., source code or image file) and their *criticality* for the software package (e.g., critical or non-critical). The criticality of a file may be determined by looking at how much of the expected software behavior can be affected when that file gets modified, replaced or deleted. Based on this guideline, a vendor, for instance, could treat executables and source files as critical, and compressed files and data files as non-critical. We use a similar rule to design the prototype (see Section 4.1.3).

3.1 Configuration Resolver

The Configuration Resolver communicates with the software vendors or third-party RIMs aggregators (e.g., NSRL or Bit9) to maintain an up-to-date database of the RIMs (the whitelist), also keeping track of the history of the RIMs, since not every VA will be equipped with the latest versions of software. It also maintains a list of RIMs that have expired or been revoked. Existing security services companies (e.g., anti-virus companies) are potential candidates for providing Configuration Resolver services to the cloud providers. The VMCVT is also managed by the Resolver and is explained next.

3.2 VM Configuration Verification Tool (VMCVT)

The VMCVT verifies the integrity of a VA by computing hashes for all of its files and comparing them with trusted hashes. Specifically, when a VA is created from a trusted base image, the tool mounts the filesystems of both the VA and base VM, hashes all of the files in both, and compares the absolute paths and hashes of the files to compile a list of files that have not changed from the base as well as the files that have been *added*, *modified*, or *deleted*. Files that are the same will be marked as "verified."

Files that have been added or modified are verified against the RIMs. The tool uses the publisher's log (which contains the exact software versions) to determine which sets of RIMs should be used to verify the software and updates installed on the VA. Added or modified files for which a corresponding RIM could not be identified are marked as "unverified." From the unverified files, the tool separates out the configuration files and performs a "diff" against the base image version (when available), or against the default version included in the software. In cases where a base VM is not available for the initial comparison, the VMCVT simply verifies all of the files in the VA against the RIMs. Although this would increase the number of files that need to be checked against the RIMs (which can be expensive if a third party service is utilized), the contents of the verification reports should not be affected.

The RIM's metadata is used to identify which files need to be present for a certain version of software to be considered complete and unmodified. For deleted files, the tool identifies the files that should not have been deleted based on that metadata. The tool also marks deleted base image files that belong to partially removed/uninstalled packages. Those files are all marked as "missing."; files that are part of completely uninstalled packages are not marked.

3.2.1 Verification report

Based on the verification results, the tool generates a digitally signed "verification report" that states the following:

- the list of installed software (and updates), including the version information;
- the list of verified, unverified, and missing files for each installed software; the file types, indicating whether a file is an executable, source file, web page, configuration file, image file, compressed file, or data file;
- the list of configuration files that have changed for each installed software and the content differences;
- the "integrity score" (explained in Section 3.2.2) for each installed software;
- the list of unverified files that are not part of the installed software;
- and the hash of the entire VA image.

The report provides consumers with an explicit list of the differences between the VA and the trusted software. Such a list provides a guide for inspecting changed files and configurations in the installed VA. For modified configuration files, in addition to the list, we provide consumers with the list of differences between the VA file and a trusted configuration file. There may also be unverified files that are not part of any installed software/update packages. All unverified files are grouped by their file types (see above), and a summary of the number of unverified files for each file type is shown in the report. The last element of the report is the hash of the entire VA image. This hash value allows the consumer to check the integrity of the VA (as it is described in the report) before launching it. The generated report is signed and published together with the VA.

The VMCVT can be made available to publishers so they may test and evaluate their VAs prior to publication. If the publishers believe that their legitimate software is failing the integrity checks, they may submit a report to the cloud provider (or a third party managing the Configuration Resolver), asking for a review and inclusion of their software. Such a practice will also help publishers get their logs correct. If their logs are listing wrong versions of software or are missing certain directory paths for a software package, their VAs will get low integrity scores; the verification reports will show which files have failed the integrity checks or which files are missing. It is the responsibility of publishers to use that information to correct their logs and add any missing directory paths before publishing their VAs.

3.2.2 Software integrity score and expected behavior

To summarize the analysis, the VMCVT tool considers each piece of software installed in a VA and assigns it a score on a scale of 1 to 3 using the rules described below. The score represents the *integrity* and *completeness* of the software as perceived by the VMCVT, indicating the extent to which a software will operate and *behave as expected*.

- integrity score ③—the software has *no* unverified or missing files, with the exception of configuration files;

- integrity score ②—the software has no unverified or missing critical files, but may have unverified or missing *non critical* files;

- integrity score ①—the software has unverified or missing *critical* files.

Installed software that has an integrity score of ③ is considered fully verified and integrity-protected, falling under the "clean or high-integrity installation" category. Clean installation indicates that a particular software package will behave as expected. A complete verification of such software requires only an inspection of the configuration files. Software that has a score of ② has only files deemed non critical missing or unverified; hence, the impact on expected behavior is considered to be limited. Such software is categorized as "partially clean or medium-integrity". Score of ① represents the "modified or low-integrity" category as such software may have critical files missing or unverified. It is highly likely that such low-integrity packages, which may have unverified or missing critical files, will not operate in an expected manner. These integrity scores allow the VA consumer to identify easily software packages that have a standard unmodified installation and those that have been modified or have a nonstandard installation.

4. ANALYSIS OF REAL-WORLD VIRTUAL APPLIANCES

In this section we shed light on the integrity of software packages in real-world VAs using the framework described in the previous section. By looking at the variance in the results, we gauge how useful the verification reports would be for consumers in selecting well installed VAs, and for providers in removing suspicious VAs from their stores. We generated verification reports for 151 randomly picked Amazon VAs and analyzed them.

4.1 Methodology and assumptions

4.1.1 Sampling method

We sampled the content of the Amazon market [2] by randomly selecting publicly available Amazon Machine Images (AMIs). We instantiated the selected images and obtained their disk content using a set of commands equivalent to the AMI tools provided by Amazon. As our prototype takes advantage of the Red Hat package manager (`rpm` database) in validating image content (in order to ease the verification process as explained further in Section 4.1.3), we focus on `rpm`-based distributions, which represent a significant portion of the Linux images available in the market.

Our random sample is composed of 151 images from an estimated pool of 2,300 valid `rpm`-based AMIs available in the Amazon US-east zone. We created the sample pool by acquiring the list of available AMIs through EC2 API calls. The call returned a list of 8,798 images that are available for instantiation. This list was reduced to 4,513 AMIs after filtering out Windows and `dpkg`-based Linux images. We randomly selected 300 images from this filtered list and tried instantiating them. About 50% of the selected images failed at boot due to errors, lack of user credentials or product codes, leaving us with a sample of 151 successfully instantiated VAs.

Table 1: Three randomly selected sub groups. The numbers shown here are average values.

	Group 1	*Group 2*	*Group 3*
Total number of files	52,886	51,909	52,786
Number of verified files	52,137	51,183	52,036
Number of software packages	427	427	426

4.1.2 Representativeness of the samples

We evaluated whether our sample size is sufficiently large to represent the entire sample pool (instantiable `rpm`-based AMIs estimated at 50% of the total 4,513). If our samples are a representative set, the randomly shuffled subgroups (partial sets) of the samples should also represent our samples (i.e., show similar characteristics) with respect to the properties of interest, which are the number of files, number of verified (and unverified) files and number of software packages (see Table 1). To verify whether they do, we randomly shuffled the 151 VAs and divided them into three subgroups, repeating this process 30 times and computing the average values. The three groups are hypothesized to share similar characteristics and closely represent each other. To test this hypothesis, we compare the properties of interest between the three subgroups. The results are shown in Table 1. There is no significant difference among the three randomly generated groups. Moreover, our statistical analysis did not reject the null hypothesis, showing $\chi^2 = 0.11$, $p = 0.99$, and degrees of freedom of 4.

4.1.3 The VMCVT prototype

Using a combination of shell and python scripts, we constructed a prototype implementation of the VMCVT tailored to the experiment at hand. Given a base image and a derived VA, it first generates the checksums by hashing all of the files in both images. It compares the hash values between the two images and creates a list of *added* files, *modified* files, and *deleted* files. The `rpm verify` command is then used to verify the integrity of the files that were added, modified, or deleted through `rpm`; `rpm verify` checks every file installed through `rpm` against its database of MD5 checksums and checks cryptographic signatures. The VMCVT keeps track of all the files that failed `rpm verify`, including the deleted files that should not have been deleted (these are what we refer to as the *missing* files). This reduces the effort needed to manually create the whitelist, since we are satisfied with the files that are verified through `rpm`. In a production implementation, though, each and every file in a VA should be checked against a common whitelist.

VMCVT then checks the remaining added, modified and deleted files (i.e., those that are unknown to `rpm`) against the whitelist that we created (see Section 4.1.4) and marks the files that are unverified or missing. To figure out whether an entire software package was removed from the base image (without using `rpm`), we inspected the differences in the file directories between the two images. If an entire directory for a software package was removed from the base, we assumed that this software package was completely uninstalled. Deleted files that were part of this directory were ignored and did not affect the software integrity scores.

The above generated unverified/missing list, together with the list that failed `rpm verify`, contains the complete list of unverified added and modified files as well as the missing files

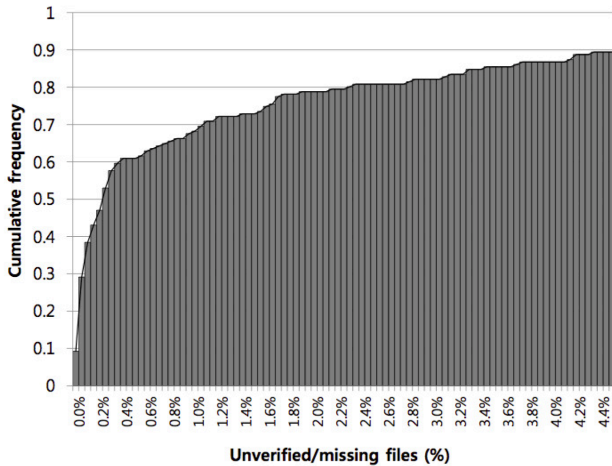

Figure 3: Cumulative frequency of VAs vs. percentage of unverified/missing files

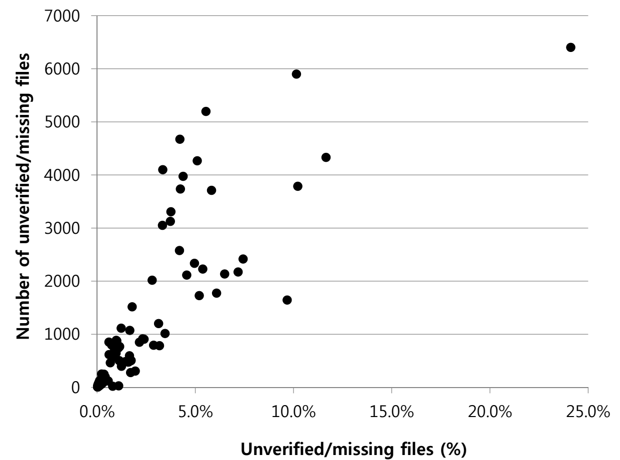

Figure 4: Number of unverified/missing files vs. percentage of unverified/missing files

that should not have been deleted. The VMCVT classifies these files into the file types shown in Section 3.2.1 using the `file` command. Then, by figuring out which software package each unverified/missing file belongs to, the VMCVT computes the integrity score of the installed software using the rules described in Section 3.2.2. Because the metadata regarding the criticality of files are not available, we treat all executables, source files, web pages, and image files as "critical", and compressed or other data files as "non critical." As the final step, the VMCVT generates a verification report (see Section 3.2.1).

4.1.4 *Constructing the whitelist*

A software whitelist was constructed manually based on the list of added, modified, and deleted files that `rpm verify` did not know about. This "rpm-unknown list" was generated for all the VAs, keeping track of the absolute paths and file names. By examining the absolute paths and the file contents, we figured out the exact software versions that were installed (without using `rpm`) on each VA; we then downloaded the source code and binary packages (e.g., tar.gz or zip files) for them from their respective vendor websites. After extracting the packages, we created MD5 checksums (the RIMs) for every file that was contained in the packages and added them to the whitelist. Here, we assumed that the downloaded packages represented trusted sources.

4.2 VA classification based on the percentage of unverified/missing files

4.2.1 *Classification method*

Figure 3 shows the cumulative frequency of unverified/missing file percentages. 90% of the VAs have less than 4.5% of unverified or missing files. Our intuition was that the number of unverified/missing files would indicate, to some degree, the integrity level of software in a VA. As the first step to study their characteristics, we first looked for a correlation between the number of unverified/missing files and the total number of files in a VA, but found none. We did find, however, a correlation between the number of unverified/missing files and the percentage (Figure 4, Pearson's: 0.84). Using that correlation, we classify the VAs into the following three "Integrity Level Groups" (ILG) to help demonstrate common VA characteristics:

- "ILG A"—44 VAs are in this group, and they have less than 0.1% of unverified/missing files;
- "ILG B"—59 VAs are part of this group, and they have 0.1-1% of unverified/missing files;
- "ILG C"—48 VAs in this group have $> 1\%$ of unverified/missing files.

4.2.2 *Characteristics of the ILGs*

The characteristics of the three groups are shown in Table 2. All average values shown in the table are "truncated average" values that exclude the top 20% and bottom 20% of the results; truncated average [22] was used to accommodate for the high variances we saw in the results. First, the average number of files in the VAs is similar across the three ILGs, averaging around 52,300. The average number of unverified/missing files, on the other hand, is significantly higher in ILG C ($1,915$) than in ILGs A (34) and B (172). That was expected, as the ILGs are classified based on the percentage of unverified/missing files and the average number of files across the three groups is the same.

Table 3 contains the number of software packages with each integrity score as well as the proportion of unverified/missing critical and non critical files, again showing the truncated average values for the three ILGs. The table indicates that the VAs in ILG C have a relatively larger number of software packages with low integrity scores (② and ①), while most of the software packages installed on VAs in ILG A have an integrity score of ③. Note, score ② represents a partially clean or medium-integrity package and ① a modified or low-integrity package (see Section 3.2.2).

The total number of packages averages around 420 in all three groups. About 99% of the software packages in the 44 VAs in ILG A are high-integrity, and the numbers are not too different for the 59 VAs in ILG B. Both groups have a small number of low-integrity and medium integrity packages. There is a big jump, however, as we reach ILG C,

Table 2: Integrity Level Groups for VAs and their file characteristics

	No. VAs	*Avg. % of unverified/missing files*	*Avg. # files*		*Avg. # unverified/missing file types*				
			All files	*Unverified/missing*	*Exe*	*Src/web*	*Img*	*Comp*	*Data*
ILG A	44	0.06%	53,737	29	3	2	1	0	23
ILG B	59	0.34%	50,292	165	16	24	3	1	121
ILG C	48	3.67%	52,198	1,871	291	635	28	6	911

Table 3: Average number of software packages installed and average number of software packages to which the three integrity scores have been assigned

	Avg. # software	*Avg. # unverified/missing*		*Avg. % of unverified/missing*		*Avg. # integrity scores*			*Avg. time*
		non critical files	*critical files*	*non critical files*	*critical files*	③	②	①	
ILG A	428	23	6	68%	18%	426	1	0	37s
ILG B	424	122	43	71%	25%	416	2	1	35s
ILG C	415	917	954	48%	50%	346	20	45	36s

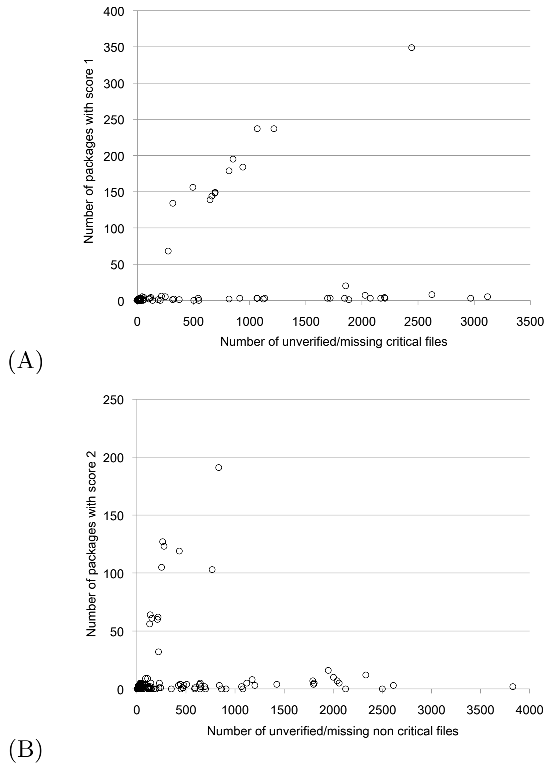

Figure 5: (A) # unverified/missing critical files vs. # score 1; (B) # unverified/missing non critical files vs. # score 2

whose VAs have, on average, 45 low-integrity packages and 20 medium-integrity packages. Those are about 11% and 4.8% of the total number of packages, respectively. We note this as our first key finding.

Finding 1: across the VAs, there is high variance in the number of unverified/missing files and the number of low-integrity and medium-integrity software packages.

While the majority of the unverified/missing files belong to what we deemed the non critical file category (data and compressed files) for ILGs A and B (68% and 71%, respectively), this proportion decreases significantly in ILG C (48%). In contrast, the proportion of unverified/missing files that belong to what we deemed the critical file category (executables, source/web files, image files) starts small in ILG A (18%), but increases significantly in ILG C (50%). This can be explained by the huge jump in the number of unverified/missing executables and src/web files from ILG A to ILG C.

Graphs (A) and (B) in Figure 5 plot the number of unverified/missing critical files against the number of packages with score ①, and the number of unverified/missing non critical files against the number of packages with score ②, respectively. Interestingly, in most VAs, both the unverified/missing critical files and non critical files are *concentrated* in a small number of packages. The graphs indicate that there is no strong correlation between the number of unverified/missing files and the number of packages they influence. Hence, the absolute number of unverified/missing files does not give a good estimation of the number of partially clean or modified packages. Our second key finding is as follows.

Finding 2: The number of unverified/missing files is not a good indicator of the number of medium-integrity or low-integrity packages installed.

There are some VAs, however, for which the unverified or missing files do spread out across a large number of packages, and these are the 12 or so outliers that we see in the two graphs. These outliers seem to show some correlation between the number of packages and the number of unverified/missing files, and so we investigate these further in Section 4.3.

4.2.3 *Verification time*

Table 3 shows that the entire verification process takes about 36 seconds on average across all three groups. This is how long publishers evaluating the configuration of their VA prior to publishing it will have to wait to see the results. The performance overhead could be significantly amortized by using the Mirage image library [13], which uses the Mirage Image Format [26] (see Appendix A).

4.3 VAs with large number of low-integrity packages

4.3.1 *Classification method*

In this second part of the analysis, we further investigate the outliers identified in Figure 5, which have noticeably larger number of partially clean and modified packages.

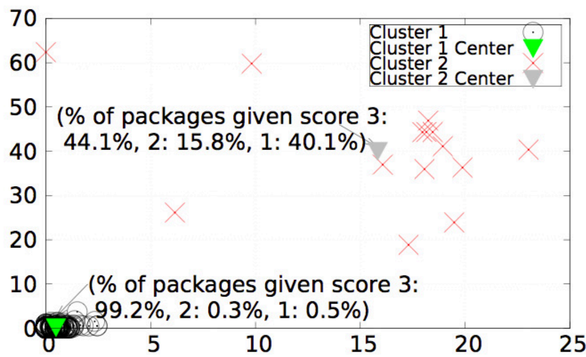

Figure 6: VA clusters based on the percentage of the software packages given each of the three integrity scores; k-means clustering with $k = 2$ is used.

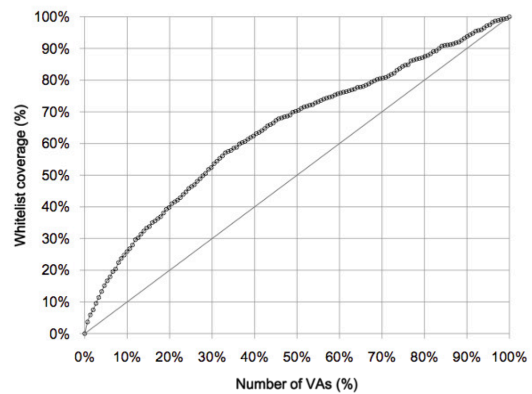

Figure 7: Percentage of the VAs used vs. percentage of the whitelist coverage

We do so by forming VA clusters based on the percentages of packages given integrity scores ① and ②. The k-means clustering method—an unsupervised learning algorithm—is used to identify the clusters ($k = 2$). Figure 6 shows the two distinct VA clusters: Cluster 1 represents VAs with low percentages of packages given scores ① and ②, comprising 137 VAs; Cluster 2 represents VAs with high percentages of packages given those two scores, consisting of 14 VAs.

4.3.2 Characteristics of the VA clusters

Table 4 summarizes the characteristics of the two VA clusters, showing that the 14 VAs in Cluster 2 have significantly high percentage of packages with scores ① and ②; the absolute numbers are also very high, averaging 171 and 77 for scores ① and ②, respectively. Only 43% of the packages are cleanly installed (high-integrity). The percentage of the unverified/missing critical files is high too, averaging 73%. In contrast, for the VAs in Cluster 1, 99% of the packages are cleanly installed, and the percentage of unverified/missing critical files averages only 22%. Considering that our samples are a representative set (see Section 4.1.2), here is our third key finding:

Finding 3: About 9% of the VAs have a significant portion of low-integrity and medium-integrity packages installed.

To cross-validate the results, we look at which ILGs the 14 VAs in Cluster 2 belong to: all belong to ILG C (the group with the highest percentage of unverified/missing files), except for one that belongs to ILG B. We took a closer look at the 14 potentially untrusted VAs and the types of unverified files they contain. Most of the files are ones that failed the `rpm verify` checks. A significant portion of the files are common system files like `/bin/cut` and `/bin/grep` which `rpm verify` flagged as not matching its database of trusted hashes for the package versions that are supposed to be installed. Although it is hard to say why those files ended up being unverified (and this is out of the scope), our virus scan results (see Section 5.2) show that 41 of those files are potentially malicious, infecting 7 of the 14 VAs. The VAs are from different publishers, were built to support different functions, and do not share common base images. What is most worrying about those 14 VAs is that none of them, in their name or VA description, mention anything about software customization or modification efforts. Just looking at the VA descriptions, they all appear to have only the cleanly installed, standard packages.

Interestingly, we found a correlation between the number of packages given score ① and the number of packages given score ② (see Figure 8, Appendix B, Pearson's: 0.65). This implies that VAs with a high number of medium-integrity packages, such as the VAs we see in Cluster 2, tend also to have a high number of low-integrity packages.

4.4 Scalability evaluation: whitelist size

This section analyzes the size of the whitelist that we created (including the `rpm` database used) to verify our sample of 151 VAs and the 47 base images, showing how the size changes as the number of VAs grows. Figure 7 plots the percentage of the VAs used against the percentage of the whitelist coverage (i.e., how much of the total whitelist was created). To get the values, we randomly shuffled all the VAs and incrementally generated the whitelist, adding the per-VA whitelist entries (the hashes used to verify a specific VA) to a global whitelist while removing any duplicate entries; the size of the global whitelist was recorded after each step. We repeated this process 10 times to get the average size for each incremental step, and used these average values to calculate the percentage values.

Intuitively, we see that the whitelist coverage percentage grows as the number of VAs used to construct the whitelist goes up. It does so more rapidly at first, but then slows down progressively. We can see that after 20% of the VAs are used, about 40% of the total global whitelist is already constructed. The size of the final global whitelist was 1,697,469 hash entries. The growth rate slows down because some software packages are installed on multiple VAs, and thus do not add any extra entries to the whitelist. Table 5 shows the top 10 most commonly installed software packages as well as their file counts (not counting the packages installed on the base images). `perl5.8.8`, which has 3,115 files, was installed commonly across 27 VAs. `python2.6` is another popular package that appeared on 22 VAs, containing 3,915 files. Such packages are responsible for the decreasing growth rate that we see on the graph.

4.5 Base image analysis

We found the base images for the VAs from the Amazon appliance store by looking at their Linux distribution

Table 4: Characteristics of the VA clusters

	# VAs	Avg. % of unverified/missing non critical files	Avg. % of unverified/missing critical files	Avg. % of integrity scores ③	②	①	Avg. no. of integrity scores ③	②	①
Cluster 1	137	68%	22%	99%	0.5%	0.3%	418	2	1
Cluster 2	14	26%	73%	43%	17%	40%	205	77	171

Table 5: Most common software packages installed

Software package	*# VAs*	*# Files*
perl5.8.8	27	3,115
selinux-policy-2.4.6	22	23
python2.6	22	3,915
kernel-2.6.21.7-2.fc8xen	15	251
kernel-2.6.18-xenU-ec2-v1.0	15	325
python2.4	12	3,038
kernel-2.6.18-238.9.1.el5xen	12	99
ec2-ami-tools-1.3-56066	11	165
rubygems-1.3.3	11	307

Table 6: Base image characteristics: truncated average values for all base images

All files	*Unverified/ missing files*	*# software*	③	②	①
38,928	5	386	385	0	0

and version information. In most cases, the base images were published by the distribution providers (e.g., Redhat, CentOS) or directly by Amazon (with the Amazon Linux AMI). It would be reasonable to expect that these base images would be clean and contain only verified files. Here we analyze their characteristics and find that it is indeed the case.

To verify 151 VAs, we downloaded 47 base images, each of which we used to verify about 3 derived images. Table 6 shows the average values for the base images. The average number of unverified/missing files is 5, which is much smaller than the number we saw for ILG A (see Table 4). Based on that characteristic and the fact that the number of low-integrity packages is sufficiently small, we claim that the base images are trusted and our assumption is valid.

4.6 On the usefulness of VA integrity assessment

Findings 1 and 3 clearly demonstrate the need for a priori software integrity assessment of VAs. When a consumer pays for a VA to use, she has the right to know that a VA is configured properly and meets her software integrity expectations.

To illustrate the usefulness of software integrity evaluation, in Table 7 we show example choices (among the 151 VAs studied) that a consumer would see when selecting a VA that provides software package `php-5.3`. The table is sorted by increasing number of unverified/missing files. The variances in the integrity scores are quite significant. VAs 5 and 7 both have more than 100 low-integrity and medium-integrity packages (scores ② and ①). Such VAs are part of Cluster 2 (see Table 4). The first two VAs have some number of unverified/missing files but the counts for the scores ② and ① are all 0. This is because all of their unverified/missing files are of type configuration files, not affecting the integrity scores.

High variances in the number of low-integrity and medium-

Table 7: Example VAs that provide php 5.3

	All files	*Unverified/ missing files*	*# software*	③	②	①
VA 1	34,410	21	351	351	0	0
VA 2	28,606	27	352	352	0	0
VA 3	63,719	48	472	470	2	0
VA 4	90,977	94	711	707	3	1
VA 5	101,582	617	711	454	123	134
VA 6	91,873	886	715	711	4	0
VA 7	74,039	2,018	641	301	103	237

integrity packages among VAs that provide similar functions reinforces the need for a priori software integrity assessment. Verification reports would help a consumer avoid VAs that are configured badly and choose ones with high-integrity packages. Providers could also make use of the verification reports to admit VAs for standard workflows only when they have high-integrity software packages. Compliance policies can be defined in terms of the integrity scores, allowing, for example, only the VAs with 1 or fewer low-integrity packages and 2 or fewer medium integrity packages to be published (using the ILG B average as a guideline). A more flexible policy might state that a VA for well-known workflow with more than 2 medium integrity packages can still be published but must always be monitored with runtime monitoring mechanisms when instantiated. Such policies would allow the repository providers to be more permissive (or open) and still maintain a reliable infrastructure.

4.7 Implementation challenges

We faced several challenges when implementing the framework and verifying Amazon VAs. This section highlights some of these challenges and suggests workarounds.

4.7.1 Origin of unverified/missing files

Identifying the software package to which an unverified or missing file belongs is not always trivial. Some packages get installed manually (i.e., without using `rpm`), and the files get located in different directories. For such files (unknown to `rpm`) that are unverified or missing, we manually identified keywords from their absolute path and matched them with software packages. The VMCVT prototype, while computing the integrity scores, used these keywords to figure out which package's score an unverified/missing file should influence. As a workaround for a production VMCVT, the publisher's log (see Section 3) should list all the directories under which a package's files are installed. The VMCVT would use that information to determine the origin of the unverified/missing files when rating the packages.

4.7.2 Creating a whitelist for proprietary software

A small number of VAs had proprietary/commercial software packages installed (e.g., Sugarcrm, Ideamax2 Gold, IBMSoftware/ITMAgent), and we could not find the source

or binary files or signed MD5 checksums for them. Hence, we selected one VA with a particular commercial software package installed and created a whitelist for the package by hashing the files from this VA instance, under the assumption that it was a trusted source. Other VA instances that also had the package installed were verified using this whitelist. We suggest that in a real system publishers submit a software review request to the Configuration Resolver provider, asking for their software to be reviewed and added to the whitelist.

4.7.3 Dealing with `tmp` and `var` files

We note that VAs typically have a lot of temporary files such as those in the `tmp` or `var` directory, and that there is no way to meaningfully verify their integrity. `tmp` files are less worrying, since they do not affect VA behavior, and we ignored them. We were less certain about the contents of `var` folders, though, and examined the types of files they contained before deciding to ignore them. Table 9 (see Appendix C) shows the top 10 most common `var` files across all the VAs we looked at. Most of the `var` files are log files and `rpm` database files, which, we argue, can also be deleted without affecting the VA behavior.

5. SECURITY IMPLICATIONS

So far, we have studied the integrity levels of real-world VAs in terms of their software identity, integrity, and completeness. Those software properties indicate the extent to which a VA will behave and operate as expected, and are also integral to figuring out how secure a VA is. This section discusses the security implications of measuring the VA integrity levels and suggests a few ways to add useful security metrics into the framework.

5.1 Expanding the integrity scores with blacklisting

There is room for improving the integrity scores to say more about the trustworthiness of the software packages. We suggest complementing our whitelist-based approach with "blacklisting" via scanning of all unverified files for viruses: score ⓪ is given to a package that fails an anti-virus check. Packages with score ⓪ fall under the "malicious" category.

A VA might be planted with a small number of malicious files (that are *known* to virus scanners), but overall has a high percentage of cleanly installed packages (score ③). Packages that contain those files will get score ⓪ though, and that VA will be flagged immediately as malicious. If those malicious files are new, clever and unknown to virus scanners, the maximum score it can get through our system is ②, which indicates a package that might not behave in an expected manner. Even though the integrity scores are not a silver bullet solution, it will provide some indication as to how trustworthy a package is whereas a virus scanner (or any other malware detection solution) will likely miss it.

5.2 Checking the unverified files against virus scanners

To demonstrate the usefulness of the combined approach and expanded integrity scores (see above), we checked all the unverified files against VirusTotal[3] [9], which is an online tool that runs multiple anti-virus engines and returns aggregated reports on the files. For a total of 111,981 unverified files (from all 151 VAs), VirustTotal flagged 45 of them as potentially malicious, which were spread out over 10 VAs (about 7% of the samples). Table 8 shows a summary of the viruses flagged by VirusTotal.

[3]We thank VirusTotal for allowing us to scan a huge volume of files.

Table 8: Viruses found from 111,981 unverified files

Report	*# Found*	*# VAs*	*Files*
Heuristic.BehavesLike. Exploit.CodeExec.F	34	8	setkeycodes; pt_chown; wordlist-compress; ...
TROJ_GEN.F47V0802	7	3	mksock; setcap
Win32.Trojan	2	1	0000; 0001 (postgres 9.1)
Win32.TRCrypt.Upkm	1	1	batik-svg-dom-1.7.jar (tomcat 6.0)
Heuristic.LooksLike. HTML.Suspicious-URL.H	1	1	BookmarkFile.pod (perl 5)

"Heuristic.BehavesLike.Exploit" is the most common report, being flagged 34 times across 8 VAs. To ensure that those are not false alarms, we looked at the number of times that a suspicious file (e.g., `setkeycodes`, `ctrlaltdel`) appeared in the unverified files list across all the VAs, and the number of times it was flagged. For most of those files, there was a difference between the two numbers, indicating that the flagged ones are, indeed, potentially malicious. For instance, `ctrlaltdel` appeared in 13 of the VAs as unverified, but only 6 of them were flagged by VirusTotal.

All of the 10 VAs infected with a potentially malicious file belong to ILG 3 (see Table 4). This is somewhat expected as the number of unverified files is much greater in ILG 3. Interestingly though just 7 of those VAs belong to Cluster 2 (see Table 4) which is characterized by the large number of low-integrity packages. We note that from the 14 VAs that are part of Cluster 2, virus scanners flagged only about 50% of them as potentially untrustworthy. This further reinforces the need for our framework as it can also flag the other 50% that are equally suspicious and would likely not meet a customers expectations when considering a VA for a standard (or well-known) workflow (e.g., an Apache web server).

Packages that contain these malicious files will get a score of ⓪, discouraging consumers from picking VAs that contain them. The providers could define a policy that disallows such VAs from being published in the first place. The results clearly indicate the need to check VAs against virus scanners. To that end, our approach will give a great performance boost by minimizing the number of files that need to be checked. Only the unverified files (i.e., files that fail whitelist checks) need to be checked, which represent only a small portion of the total number of files; 90% of the VAs contain less than 4.5% of unverified files (see Figure 3). For VAs that fall under ILG C (see Table 3), this would imply checking only about 2,012 unverified files through a virus scanner, as opposed to checking all 52,198 files.

5.3 Combining integrity scores with security vulnerability assessment results

The expanded integrity scores would still provide only a partial view of how secure a VA might be. A VA that is equipped with 100% cleanly installed packages is *likely to be*

more secure than a VA that has some modified packages, but not always. If some of the clean packages have known security vulnerabilities, then the first VA might be equally insecure or even more insecure than the second VA. Hence, to make strong claims about VA security, we also need to consider whether packages have known security vulnerabilities, and combine this information with their integrity scores.

Numerous security companies today assess software vulnerabilities and report them. This information could be used to build a more complete verification report, also indicating which packages are known to be secure or have known vulnerabilities. The corresponding integrity scores would then provide a stronger indication of how secure that VA is. We imagine that the providers could make use of this information and define a whitelist of software packages or combinations of packages that are known to be secure; it would also be possible to create a blacklist of packages that are known to have security vulnerabilities or bugs. Compliance policies could then be defined based on the whitelist and blacklist, specifying which software combinations are allowed to be published and which should not be admitted.

6. CONCLUSIONS AND FUTURE WORK

We studied the integrity of software packages in real-world virtual appliances (VAs) through a software whitelist-based framework, and found high variance in the software integrity across VAs. Our analysis of 151 Amazon VAs showed that about 9% of those real-world VAs were configured with a significant portion of modified (low-integrity) packages without any indication of the publishers' efforts to customize them—demonstrating the need for a priori assessment of software integrity to help consumers pick correctly installed VAs. Virus scanners were able to flag only about half of the VAs in that 9%, showing that the whitelisting-based integrity assessment has a role to play, and may complement blacklisting techniques like virus scanners in helping consumers pick more reliable VAs. While whitelist-based techniques are hard to realize in practice, we find that because a large number of software packages are installed on multiple VAs in the case of appliance stores, the rate at which the whitelist size grows slows down over time.

In the future, we plan to further expand the integrity assessment framework to compute an overall trust score for a VA based on the integrity scores assigned to individual software packages. In doing so, we will consider other factors such as the criticality of a software package and consumer's preference order for packages. Future work may also look at VAs that are built on Windows or other Linux distributions (e.g., `dpkg`-based distributions) to further validate the key findings presented in this paper.

Acknowledgments

This material is based on research sponsored in part by the Air Force Research Laboratory and the Air Force Office of Scientific Research, under agreement number FA8750-11-2-0084. The authors would like to thank Jenny Applequist for her editorial assistance, and the anonymous reviewers for their careful attention and insightful comments.

7. REFERENCES

[1] Amazon Elastic Compute Cloud. `http://aws.amazon.com/ec2/`.

[2] Amazon Machine Images (AMIs). `http://aws.amazon.com/amis`.

[3] Bit9 Global Software Registry. `http://www.bit9.com/products/bit9-global-software-registry.php`.

[4] BitNami Virtual Images. `http://bitnami.org/learn_more/virtual_machines`.

[5] CUBRID Virtual Images. `http://www.cubrid.org/virtual_machine_images`.

[6] IBM SmartCloud.

[7] NIST National Software Reference Library. `http://www.nsrl.nist.gov/`.

[8] Thunderflash Pre-Built Virtual Images. `http://thunderflash.com/`.

[9] Virustotal. `https://www.virustotal.com/`.

[10] VMWare Solution Exchange. `https://solutionexchange.vmware.com/`.

[11] Security Guidance for Critical Areas of Focus in Cloud Computing. `http://www.cloudsecurityalliance.org/guidance/csaguide.pdf`, April 2009.

[12] AntiVirus Performance Statistics. `http://winnow.oitc.com/malewarestats.php`, August 2012.

[13] G. Ammons, V. Bala, T. Mummert, D. Reimer, and X. Zhang. Virtual machine images as structured data: the mirage image library. In *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud'11, pages 22–22, Berkeley, CA, USA, 2011. USENIX Association.

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[15] K. D. Bowers, A. Juels, and A. Oprea. Hail: a high-availability and integrity layer for cloud storage. In E. Al-Shaer, S. Jha, and A. D. Keromytis, editors, *ACM Conference on Computer and Communications Security*, pages 187–198. ACM, 2009.

[16] K. D. Bowers, M. van Dijk, A. Juels, A. Oprea, and R. L. Rivest. How to tell if your cloud files are vulnerable to drive crashes. In Y. Chen, G. Danezis, and V. Shmatikov, editors, *ACM Conference on Computer and Communications Security*, pages 501–514. ACM, 2011.

[17] Bugiel, Sven and Nürnberger, Stefan and Pöppelmann, Thomas and Sadeghi, Ahmad-Reza and Schneider, Thomas. AmazonIA: when elasticity snaps back. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 389–400, New York, NY, USA, 2011. ACM.

[18] B. Danev, R. J. Masti, G. O. Karame, and S. Capkun. Enabling secure VM-vTPM migration in private clouds. In *Proceedings of the 27th Annual Computer Security Applications Conference*, pages 187–196, New York, NY, USA, 2011. ACM.

[19] T. Garfinkel and M. Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *Proceedings of the 10th conference on Hot Topics in Operating Systems - Volume 10*, HOTOS'05, pages 20–20, Berkeley, CA, USA, 2005. USENIX Association.

[20] J. H. Huh, H. Kim, J. Lyle, and A. Martin. Achieving attestation with less effort: an indirect and configurable approach to integrity reporting. In *Proceedings of the sixth ACM workshop on Scalable trusted computing*, pages 31–36, New York, NY, USA, 2011. ACM.

[21] K. R. Jayaram, C. Peng, Z. Zhang, M. Kim, H. Chen, and H. Lei. An empirical analysis of similarity in virtual machine images. In *Proceedings of the Middleware 2011 Industry Track Workshop*, Middleware '11, pages 6:1–6:6, New York, NY, USA, 2011. ACM.

[22] John A. Rice. *Mathematical Statistics and Data Analysis*, chapter 10, page 365. Duxbury Press, 2 edition, 1994.

[23] G. Kim and E. Spafford. The design and implementation of tripwire: A file system integrity checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 18–29. ACM, 1994.

[24] N. Leavitt. Is cloud computing really ready for prime time? *Computer*, 42(1):15 –20, jan. 2009.

[25] N. Quynh and Y. Takefuji. A novel approach for a file-system integrity monitor tool of xen virtual machine. In *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 194–202. ACM, 2007.

[26] D. Reimer, A. Thomas, G. Ammons, T. Mummert, B. Alpern, and V. Bala. Opening black boxes: using semantic information to combat virtual machine image sprawl. In *Proceedings of the fourth ACM SIGPLAN/SIGOPS international conference on Virtual execution environments*, VEE '08, pages 111–120, New York, NY, USA, 2008. ACM.

[27] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.

[28] T. Ristenpart and S. Yilek. When good randomness goes bad: Virtual machine reset vulnerabilities and hedging deployed cryptography. In *NDSS*. The Internet Society, 2010.

[29] H. Takabi, J. Joshi, and G. Ahn. Security and privacy challenges in cloud computing environments. *Security Privacy, IEEE*, 8(6):24 –31, nov.-dec. 2010.

[30] TCG. TCG Infrastructure Working Group Architecture Part II - Integrity Management. `http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_architecture_part_ii__integrity_management_version_10`, November 2006.

[31] D. Vincenzetti and M. Cotrozzi. Anti tampering program. In *Proceedings of the Fourth {USENIX} Security Symposium, Santa Clara, CA*. USENIX, 1993.

[32] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. Managing security of virtual machine images in a cloud environment. In *Proceedings of the 2009 ACM workshop on Cloud computing security*, CCSW '09, pages 91–96, New York, NY, USA, 2009. ACM.

[33] Y. Zhang, A. Juels, A. Oprea, and M. K. Reiter. Homealone: Co-residency detection in the cloud via side-channel analysis. In *IEEE Symposium on Security and Privacy*, pages 313–328. IEEE Computer Society, 2011.

[34] 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 *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 377–386, New York, NY, USA, 2010. ACM.

APPENDIX

A. AMORTIZING PERFORMANCE OVERHEADS

Mirage [13, 26] generates an index of a VA using the VA's filesystem structure in order to simplify maintenance and management of a large collection of images. During index generation, Mirage already computes hashes of all the files in the VA to detect duplicate files in the system. Integration of our framework with the Mirage library would involve only the addition of one step to look up the hash of the files in a whitelist when a file is first encountered. Thus, in addition to the other benefits of using Mirage, including reduced storage cost and query capabilities, the cost of producing the verification reports can be significantly amortized.

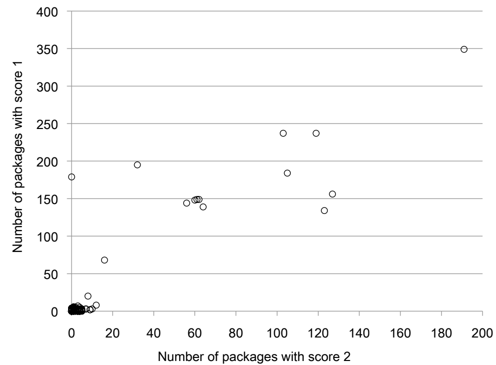

Figure 8: Number of software packages with score 1 vs. number of software packages with score 2

B. NUMBER OF MEDIUM-INTEGRITY PACKAGES VS NUMBER OF LOW-INTEGRITY PACKAGES

There is a correlation between the number of medium-integrity (partially clean) packages and the number of low-integrity (modified) packages (see Figure 8). As the number of packages with score ① increases, so does the number of packages with score ②, and vice versa.

C. TMP AND VAR FILES

Table 9: Most common files in "var" folders

# VAs	*Absolute path*
151	var/log/wtmp
151	var/log/spooler
151	var/log/secure
151	var/log/messages
151	var/log/maillog
151	var/lib/rpm/Triggername
151	var/lib/rpm/Sigmd5
151	var/lib/rpm/Sha1header
151	var/lib/rpm/Requireversion
151	var/lib/rpm/Requirename

As Figure 9 shows, most of the `var` files are log files and `rpm` database files, which, we argue, can be deleted without affecting the VA behavior. The `rpm` database, if needed, can be reconstructed after a VA is instantiated. We propose that temporary files be scrubbed from the VA before publication. Similarly, files in the `var` directory should also be scrubbed before publication, unless the software vendor metadata or the publisher's log explicitly mention the file as necessary.