Managing Security of Virtual Machine Images in a Cloud Environment

Jinpeng Wei[*]
Florida International University
weijp@cs.fiu.edu

Xiaolan Zhang
IBM T. J. Watson Research Center
cxzhang@us.ibm.com

Glenn Ammons
IBM T. J. Watson Research Center
ammons@us.ibm.com

Vasanth Bala
IBM T. J. Watson Research Center
vbala@us.ibm.com

Peng Ning
North Carolina State University
pning@ncsu.edu

ABSTRACT

Cloud computing is revolutionizing how information technology resources and services are used and managed but the revolution comes with new security problems. Among these is the problem of securely managing the virtual-machine images that encapsulate each application of the cloud. These images must have high integrity because the initial state of every virtual machine in the cloud is determined by some image. However, as some of the benefits of the cloud depend on users employing images built by third parties, users must also be able to share images safely.

This paper explains the new risks that face administrators and users (both image publishers and image retrievers) of a cloud's image repository. To address those risks, we propose an image management system that controls access to images, tracks the provenance of images, and provides users and administrators with efficient image filters and scanners that detect and repair security violations. Filters and scanners achieve efficiency by exploiting redundancy among images; an early implementation of the system shows that this approach scales better than a naive approach that treats each image independently.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection; H.3.5 [**INFORMATION STORAGE AND RETRIEVAL**]: Online Information Services

General Terms

Management, Security

[*]Part of this research was performed while being a summer intern at IBM Watson.

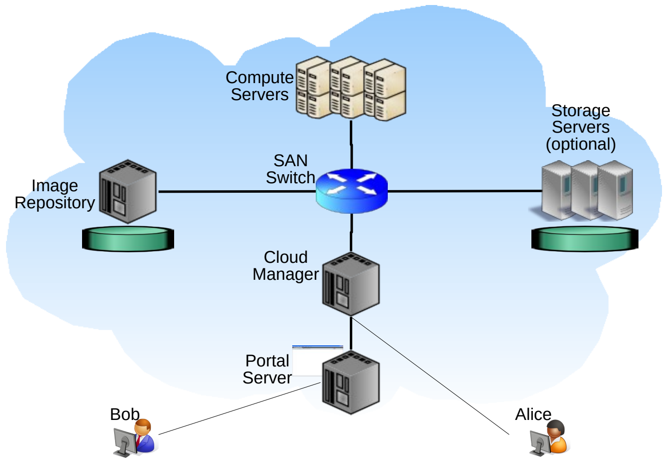

Figure 1: High Level Architecture of a Typical Cloud.

Keywords

Cloud Computing, Virtual Machine Image, Image Repository

1. INTRODUCTION

Cloud computing has emerged as one of the most influential technologies in the IT industry and is rapidly revolutionizing the way IT resources are managed and utilized [9]. Through clever use of virtualization technologies, the cloud offers customers the ability to start businesses without having to pay huge upfront capital expenses and the flexibility to scale the IT infrastructure up and down as a business evolves without worrying about over or under provisioning. Major services in this field include Amazon's EC2 [7], IBM's Smart Business cloud offerings [16], Microsoft's Azure [18], and Google's AppEngine [12].

Figure 1 shows the architecture of a typical cloud at a high level. An end user Bob connects to the cloud via a portal from his browser. Alternatively, a user Alice can choose to directly connect to the cloud manager via a command line interface similar to that used in EC2. A cloud provides three types of resources: a collection of virtual machine (VM) images, a set of computer servers on which the VM images can be run, and optionally a storage pool to store persistent user data.

Despite the cloud's huge potential in reduced costs and improved productivity, and overwhelming enthusiasm from customers, security experts repeatedly warn that security problems could inhibit wide adoption of the cloud model [9, 4, 14, 5]. For example, John Chambers, CEO of Cisco, stated in his keynote speech at the 2009 RSA conference, "[cloud computing] is a security nightmare, and it can't be handled in traditional ways". And many open questions have been raised in security for cloud computing [13, 9].

In this paper, we discuss the question of securely managing VM images in a cloud environment. VM images are unique entities in the cloud with special traits. First, VM images need high integrity, because they determine the initial states of running virtual machines, including their security states. In other words, the security and integrity of such images are the foundation for the overall security of the cloud. Second, many of the VM images are designed to be shared by different and often unrelated users. For example, one of the major efficiency improvements promised by the cloud is realized through employing VM images built by a third party [15, 19]. As we will discuss in detail later, sharing of VM images poses privacy and safety issues.

Unfortunately, existing approaches to cloud security built by cloud practitioners fall short when dealing with VM images. Such techniques mainly focus on two aspects of security: 1) security of running instances, and 2) integrity and privacy of customer data. For example, EC2 supports the concept of a security group, which is roughly a set of IP addresses that are either allowed or denied access to a target VM as defined by a corresponding firewall rule [6]. Amazon also provides the S3 storage service, which stores their customers' non-volatile data in an encrypted manner [8]. However, techniques that protect running instances are not applicable to VM images because by definition such images are dormant (e.g., not running). Additionally, techniques that protect customer data are also inadequate for VM images because a VM image is not simply a piece of static customer data but also includes the entire software stack that boots a VM into its initial state (or, in the case of a snapshot, the previous checkpoint state). For these reasons, VM images call for non-traditional measures to secure them, which motivates this paper.

In the remainder of this paper, we elaborate security risks and challenges in a cloud environment from the image perspective (Section 2), and propose an image management system that attempts to mitigate, if not eliminate, these security risks (Section 3). To our best knowledge, this is the first image management system that is designed with built-in image security functionality. We discuss advantages of our approach over alternative designs as well as its limitations (Section 4). We then describe related work (Section 5) and conclude with an outlook of the future (Section 6).

2. SECURITY RISKS IN AN IMAGE REPOSITORY

One of the value propositions of cloud computing is reduction of management cost, in both hardware and software. This cost reduction is achieved by sharing the knowledge of how to manage a piece of IT asset, be it a simple Red Hat Linux installation, or an enterprise-wide Websphere deployment, via VM images. Thus VM image sharing is one of the fundamental underpinnings of cloud computing.

VM image sharing unavoidably introduces security risks. We elaborate these security risks from the perspectives of various roles that are involved in the sharing activity. The *publisher*, or *owner*, of an image is the one who contributes the original image to the repository. She is mostly concerned about confidentiality (e.g., inadvertent leaking of sensitive information and unauthorized accesses to the image). The *retriever*, or *consumer*, of an image is the one who retrieves the image from the repository and runs it on the compute servers. She is mostly concerned about safety (e.g., a malicious image that is capable of corrupting or stealing the retriever's own private data). Common to both the retriever and the administrator is the risk of non-compliance (e.g., running unlicensed software or software with expired licenses). The *administrator* is concerned with the security and compliance of the cloud system as a whole and the integrity of individual images. The administrator assumes the liability of potential damages caused by malware contained in any image stored in the repository.

We next look at each security risk in more detail. Note that, although the context of our discussion is a public cloud, similar risks appear in private clouds that operate within an enterprise, which must secure all of its assets, including its images.

2.1 Publisher's Risk

By publishing an image, the publisher risks releasing sensitive information inadvertently. Although traditional software publishers run similar risks, the problem is larger for image publishers because images contain installed and fully configured applications: the configuration might require dangerous operations like creating password-protected user accounts and, if the publisher sets up the application by running an instance of the image, she may unwittingly create files that should not be made public. For example, suppose that Alice, while configuring an application in a running instance, starts a web browser. If Alice is careless, she will publish the history of her browsing session along with her image!

Alice's carelessness is egregious but note that similar carelessness goes unpunished when applications are installed and configured on unshared physical machines. Alice is surprised because installation and configuration, which used to be private, now result in a public image that can be searched and executed by others.

In addition to wanting protection from releasing information inadvertently, the publisher may want to share her image with only a limited set of users. Therefore, the store should support some form of access control for images.

2.2 Retriever's Risk

The retriever risks running vulnerable or malicious images introduced into the repository by a publisher. While running a vulnerable virtual machine lowers the overall security level of a virtual network of machines in the cloud, running a malicious virtual machine is similar to moving the attacker's machine directly into the network, bypassing any firewall or intrusion detection system around the network.

Virtual machine image sharing provides an easier way of developing and propagating Trojan horses. Traditionally, a

Trojan horse program can only be developed and tested on the hacker's machine, and will only run on a victim's machine if the victim's software stack satisfies its dependencies. Therefore, to target a wide range of victims, the hacker must develop and test variances of his Trojan horse on different software stacks and make sure that the right version is delivered to the right victim. Using a virtual machine image as a carrier for the Trojan horse makes the hacker's job easier, because the virtual machine image encapsulates all software dependencies of the the Trojan horse. In other words, the dependency on the victim's software stack is eliminated.

The retriever also risks running illegal software (unlicensed or with expired licenses) contained in an image.

2.3 Repository administrator's risk

The repository administrator risks hosting and distributing images that contain malicious or illegal content [1]. It is as much in his interest as in the consumer's to keep the images free of such content.

Security attributes of dormant images are not constant. Typically, the security level of a dormant VM image degrades over time, because a vulnerability may be unknown when the VM image is initially published but become known and exploitable later. Anecdotal evidence shows that, if dormant VM images are not managed (e.g., scanned for worms), a virtual environment may never converge to a steady state, because worm-carrying VM images can sporadically run, infect other machines, and disappear before they can be detected [11]. The same property holds for software licenses. Administrators thus carry a *latent security risk* that stems from long-lived but inactive images. This risk is often overlooked by administrators due to the high maintenance cost of keeping those images up to date with regard to security patches and software licenses. As the number of VM images grows, so does the risk and along with it the cost of maintenance.

3. APPROACH

3.1 Approach overview

We propose an image management system called Mirage that addresses the security concerns outlined in Section 2 in an efficient manner. It provides the following security management features:

- An access control framework that regulates the sharing of VM images. This reduces the publisher's risk of unauthorized accesses to her images.
- Image filters that are applied to an image at publish and retrieve time to remove unwanted information in the image. Unwanted information could be information that is private to the user, such as passwords, or malicious, such as malware, or illegal, such as pirated software. Filters address the security risks of all three parties. Filters reduce the publisher's risk of inadvertently releasing private information, the retriever's risk of consuming illegal or harmful content, and the administrator's risk of assuming liability for hosting such content.

[1]For simplicity, we assume that the repository administrator is also the cloud administrator. This is the case for the majority of existing cloud offerings.

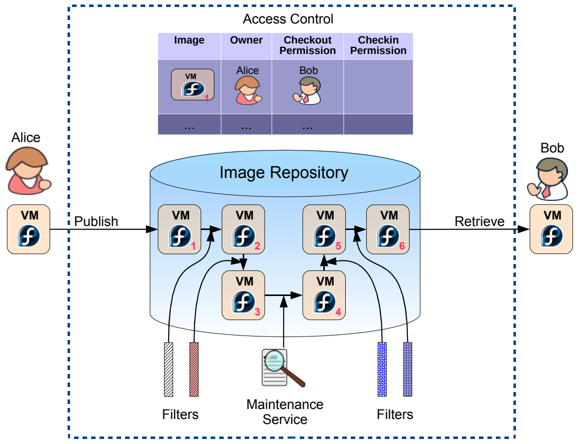

Figure 2: Security Features of the Mirage Image Management System

- A provenance tracking mechanism that tracks the derivation history of an image and the associated operations that have been performed on the image (through the image repository API). Security functionality like auditing can be built on top of this provenance tracking layer. Provenance tracking provides accountability and discourages the intentional introduction of malicious or illegal content, which in turn reduces the administrator's risk of hosting images that contain such content. We also use the provenance to track modifications to the image that result from applying filters.
- A set of repository maintenance services, such as periodic virus scanning of the entire repository, that detect and fix vulnerabilities discovered after images are published. These reduce the retriever's risk of running malicious or illegal software and the administrator's risk of hosting them.

Figure 2 shows the overall architecture of our image management system, with an emphasis on its security capabilities. It consists of four major components that implement the four features outlined above: (1) an access control framework that regulates the sharing of VM images, (2) filters that remove unwanted information from images at publish and retrieval time, (3) a provenance tracking mechanism that tracks the derivation history of an image, and (4) repository maintenance services, such as periodic virus scanning, that detect and fix vulnerabilities discovered after images are published. We next discuss each of them in more detail.

3.2 Access Control

Each image in the repository has a unique owner, who can share images with trusted parties by granting access permissions. We currently support two types of access permissions, *checkout* and *checkin*. A checkin permission implies a checkout permission. Retrieving and running an image requires checkout permission. Revising an image and storing the revised image in the repository requires checkin permission. Note that, even without checkin permission for an image, a user can retrieve the image, modify it, and publish it as a new image; however, the provenance-tracking system would not consider the new image to be a revision of the original.

All other operations on an image, such as granting and revoking access to the image, require the operator to be the owner (or the repository administrator). By default an image is private, meaning that no one but the owner and the administrator can access the image.

3.3 Image Transformation by Running Filters

Filters at publish time can remove or hide sensitive information from the publisher's original image. For example, a "remove" filter excludes a file from the original image, and a "hide" filter keeps the file but replaces its content with some safer version (e.g., replacing credit card numbers with invalid numbers). Examples include a filter that removes a web browser's temporary files and a filter that removes the shell's input history.

Two types of filters can be applied at publish time: repository-specific filters and user-specific filters. Repository-specific filters are system-wide filters that reflect security best practices. Some of them are mandatory, and others are optional. The publisher can specify composition of optional filters. User-specific filters are intended to remove or hide user-specific sensitive content from the images. Since different users may have different notions about what content is sensitive, these filters can only be supplied by the user.

For the safety of the repository, user-supplied filters are never executable code. Instead, they are high-level specifications of transformation rules that are interpreted by the repository. An example rule is "replace *patternA* with *patternB*", where *patternA* and *patternB* are both regular expressions. For binary files, simple regular expression based pattern matching might not be sufficient. How to design effective and efficient filters for binary files remains to be our future work.

The order in which filters are applied matters. E.g., if user-specific filters were allowed to run after critical repository-specific filters, they could invalidate the latter's guarantees.

Filters can also be applied at retrieve time. Such filters may be specified by the publisher, who may want to restrict the information that is exposed to a particular retrieve, or by the retriever, who may want to protect herself from a malicious or illegal image.

3.4 Provenance Tracking

The image management system tracks the derivation history of an image by recording the parent image information when a new image is deposited into the repository, along with the information about the operation that resulted in the creation of the new image. For example, if Bob checks out an image A owned by Alice, modifies the image, and later checks it back in as a new image B, the system will record that image B derives from image A via method *checkin*. As another example, if the system discovers a vulnerability in image C and applies the latest security patch for it that results in a new image D, the system records the fact that image D derives from image C using method *maintenance*, as well as the specific patch that was applied.

The provenance information is used in two ways. It can be consumed by an audit system to trace the introduction of illegal or malicious content. It can also be used to alert the owners of derived images when the parent image is patched (e.g., a vulnerability is discovered and fixed), so that the derived images can be patched as well.

Although not a security feature, the provenance information can also be displayed to the end user for her to visually inspect the derivation history of her images.

3.5 Image Maintenance

As we mentioned in Section 2, dormant images are more than just static data. They are IT assets and should be managed like physical IT assets: they should be regularly checked for compliance, scanned for malware, and patched with the latest security fixes.

Unfortunately, administrators often fail to manage dormant images properly, mainly for two reasons. First, because a dormant image's lack of security or integrity is not obvious until it is run, it is tempting to defer its maintenance. Second, maintenance operations are time consuming. The time it takes to start a running instance of an image, scan and patch the instance, and then capture it back to a new image, is on the order of hours. Imagine a repository with thousands or millions of images. It could easily take months to perform just one round of maintenance for a repository of a typical size cloud.

Our repository provides a set of maintenance services that can be efficiently run over the entire repository (See Section 3.6. Example maintenance services include malware detectors (e.g., virus scanners), license compliance managers (e.g., IBM Tivoli LCM [17]), and security patchers.

3.6 Feasibility of Our Approach

An acute reader might have noticed that our approach to managing image security could introduce huge performance overheads, both in space and in time.

As a side effect of running the filters and maintenance services, new revisions of a published image are generated constantly. One can choose to discard the original image, but there are valid reasons to keep the original image. One reason is to ensure that the transformation never renders an image unusable, because the user can always go back to the original revision if she desires. Therefore, our system retains all revisions. A naive approach to storing these revisions is clearly inadequate. Similarly, a naive approach to running the maintenance services over the entire repository is insufficient.

Our approach leverages the content addressable nature of the Mirage store [22] to exploit similarities among images. If two files have the same content, Mirage stores that content only once, even if the files belong to two different images. Because images along a derivation chain tend to be similar, storing the many revisions produced by filters and maintenance operations does not require an unreasonable amount of storage. Also, when the same content occurs in many images, maintenance operations and filters operate on that content only once, which reduces their running time.

Our preliminary experiments show encouraging results. In the experiments, we compare the running time of ClamAV [1], a free virus scanning tool, on a group of nine images using conventional methods versus our approach. Figure 3 shows that the scanning time in the traditional approach grows linearly with the number of images; whereas for Mirage, the total scanning time grows much more slowly. This is expected because the images derive from the daily build of the same application, so the differences between images are minimal. Once the first image is processed, the additional work to process the next image is proportional to the

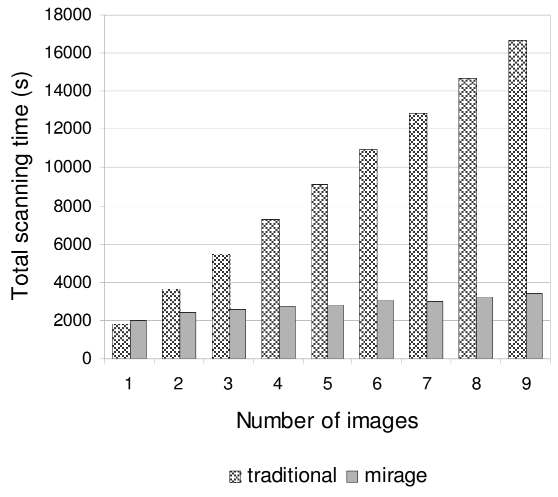

Figure 3: ClamAV Scanning Time.
The images are derived from daily builds of a large, commercial, Eclipse-based development environment. The size of each image is around 6GB, and each image contains around 60,000 files. This experiment is run on a laptop with Intel Pentium M processor at 2G Hz, 2GB main memory and 80GB hard disk. The operating system is Ubuntu Linux 7.10, and the kernel version is 2.6.22-14.

difference between the two images. In contrast, with the traditional approach, all files in the new image need to be scanned regardless of whether they have changed or not. As a result, our approach achieves a speedup of 4.9 even for a relative small repository with 9 images. We expect the savings to increase as the size of repository increases.

Once a vulnerability is identified, the next step is to eliminate the vulnerability by applying a security patch. As with virus scanning, we expect similar performance savings in this step. However, batch VM patching does introduce additional ”safety” concerns because it involves changes to a VM image. How to make sure that state changes on one VM can be safely applied to another is our future work.

4. DISCUSSION

4.1 Advantages over other design alternatives

4.1.1 Client-side security management

Arguably the security and management functionalities can be performed at the client's side instead of at the repository. For instance, users can employ software tools such as SecureClean [3] or Privacy Protector [2] to remove traces of personal information from the user's hard drive. Users can also schedule a periodic task to scan the dormant images for viruses and expired licenses. However, doing it only at the client side is potentially less effective and less efficient. It is less effective because relying on the users solely to protect their sensitive information or to cleanse a downloaded VM image before running is not practical. For instance, not all users are aware of privacy protection tools or have access to them. Implementing security at the client side may also miss out many performance optimization opportunities that are only present in a centralized image repository system, as we demonstrated in Section 2. Additionally, having a large set of images provides the opportunity to explore data mining techniques to automatically discover sensitive or malicious data that might have been missed by off-the-shelf tools.

Note however, that our approach does not preclude filters applied at other places such as the publisher's machine, which are complementary to those in the repository.

4.1.2 Applying filters lazily

An alternative design to our continuous maintenance service is to defer the application of filters til the moment the image is retrieved. This way, images that are not retrieved never needs to be scanned or patched. While it is true that not every image is consumed eventually, our approach of continuous maintenance offers an attractive feature: instant deployability. In contrast, lazy filter application places these maintenance filters (e.g., virus scanning which are typically time-consuming) in the critical path of the retrieval of an image.

4.2 Limitations

Admittedly filters cannot always be 100% accurate, since what constitutes illegal or private content is highly application dependent. We note that this is a problem intrinsic to information sharing – that when a user releases information, she would have to assume some degree of risk that she might inadvertently release some private data; conversely, when she receives information, she assumes the risk of receiving illegal content. Our system does not eliminate that risk entirely, but it does mitigate the risk, in a systematic and efficient way. In our future work, we will explore techniques to automatically identify sensitive data in an image and generate filters for them.

By the same token, our virus scanning does not guarantee to find all malware in an image. Nonetheless it is still a useful service because it reduces the likelihood of having malware embedded in images.

Our current approach of using simple pattern matching to support user-specific filters may render the resulting VM image unusable in some cases. One future direction is to provide some automated testing environment for the publisher to verify that the transformed image still functions correctly.

It may seem counter intuitive, but ”the ability to monitor or control customer content” might increase the liability of the repository provider [24]. Cloud providers thus need to weigh the benefits of security services (or the risks of *not* running those services) against the additional liability associated with those services. Nonetheless, we believe that in general privacy and security management for VM images is useful for the Cloud.

5. RELATED WORK

As mentioned in section 1, virtual machine image repositories such as VMware's Virtual Appliance Market Place [23] and Amazon's EC2 [7] have emerged. However, these repositories provide only basic services such as image store and retrieval. They do not provide security management of VM images within the repository. Maintenance of images are performed outside of the repository by starting the image in a VM and running the maintenance operation. This approach does not scale as well as ours, which can exploit similarities among images in the repository.

A number of approaches exploit redundancies of contents that belong to different data sets to achieve storage efficiency. The Jumbo Store [10] exploits similarity between successive directory snapshots by encoding the file system directory tree as graphs of small chunks of variable-length data and hashing the data chunks to detect redundancy. As a result, it provides very efficient incremental update and storage of directory snapshots. Similarly, Venti [21] uses hash of content blocks as the identifier for the blocks. These systems differ from Mirage in that they exploit redundancy at the block (chunk) level, whereas Mirage is a file level store. Additionally, these systems have very different design goals than Mirage: they serve as generic storage system, while Mirage is designed specifically for storing VM images.

Ventana [20] is a virtualization-aware file system that stores VM images and enables fine-grained sharing of VM images. However, it does not support filtering and maintenance of VM images as done by the Mirage repository. Both Ventana and Mirage offer version control.

6. CONCLUSION

Cloud computing offers great potential to improve productivity and reduce costs. It also poses many new security risks. In this paper we explored these risks in depth from an image repository perspective. In particular, we analyzed the risks faced by administrators and users (both image publishers and image retrievers) of a cloud's image repository. We presented a design that addresses those risks and argued that the design is implementable and efficient. Filters detect malicious images and remove sensitive information like passwords. Provenance tracking and access control enable publishers to control which images are available to which users and enables users to find images that meet their needs. And repository maintenance services reduce the risk to users of running vulnerable or illegal software.

Our preliminary results showed that filters run most efficiently at the repository, where they can exploit similarities among images. One can imagine other services that are best provided at the repository or at least with support from the repository. For example, an update notification service could inform a consumer when a bug fix is applied to an image used by the consumer. We expect that many more such services can be implemented efficiently in our image management system.

7. ACKNOWLEDGMENTS

We thank Arun Iyengar at IBM T. J. Watson Research Center for his useful feedback. We also thank our anonymous reviewers for their insightful comments that have greatly improved this paper.

8. REFERENCES

[1] Clam AntiVirus. `http://www.clamav.net/`.

[2] Privacy protector. `http://www.NetDuster.com/Privacy/`.

[3] Secureclean. `http://www.secureclean.com/`.

[4] Cloud security stokes concerns at RSA, April 2009. Available at `http://www.networkworld.com/news/2009/042309-rsa-cloud-security.html`.

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

[6] Amazon. Amazon EC2, Developer Guide. `http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/`.

[7] Amazon. Amazon Elastic Compute Cloud (Amazon EC2). `http://aws.amazon.com/ec2`.

[8] Amazon. Amazon Simple Storage Service (Amazon S3). `http://aws.amazon.com/s3`.

[9] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, and et al. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, 2009. Available at `http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-28.html`.

[10] K. Eshghi, M. Lillibridge, and et al. Jumbo store: providing efficient incremental upload and versioning for a utility rendering service. In *FAST'07*, 2007.

[11] T. Garfinkel and M. Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Tenth Workshop on Hot Topics in Operating Systems (HotOS'05)*.

[12] Google. Google App Engine. `http://code.google.com/appengine/`.

[13] B. Hayes. Cloud Computing. *Commun. ACM*, 51(7):9–11, 2008. Available at `http://doi.acm.org/10.1145/1364782.1364786`.

[14] J. Heiser and M. Nicolett. Assessing the Security Risks of Cloud Computing, June 2008.

[15] IBM. IBM AMIs on Amazon's EC2. `http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=229`.

[16] IBM. IBM Cloud Computing. `http://www.ibm.com/ibm/cloud`.

[17] IBM. IBM Tivoli License Compliance Manager. `http://www.ibm.com/software/tivoli/products/license-mgr/`.

[18] Microsoft. Azure Services Platform. `http://www.microsoft.com/azure/default.mspx`.

[19] Oracle. Oracle AMIs on Amazon's EC2. `http://developer.amazonwebservices.com/connect/kbcategory.jspa?categoryID=205`.

[20] B. Pfaff, T. Garfinkel, and M. Rosenblum. Virtualization aware file systems: getting beyond the limitations of virtual disks. In *Proceedings of the Third Symposium on Networked Systems Design and Implementation (NSDI '06)*, May 2006.

[21] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the 1th Usenix Conference on File and Storage Technologies*, 2002.

[22] 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 *The 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, March 5-7, 2008.

[23] VMware. Virtual Applicance Marketplace. `http://www.vmware.com/appliances/`.

[24] Eric Goldman. A Fresh Look at Web Development and Hosting Agreements. `http://www.ericgoldman.org/Articles/freshlookarticle.htm`.