AmazonIA: When Elasticity Snaps Back

Sven Bugiel*, Stefan Nürnberger*, Thomas Pöppelmann†,
Ahmad-Reza Sadeghi*†, Thomas Schneider*

*System Security Lab / CASED,
Technische Universität Darmstadt, Germany

†Fraunhofer SIT,
Darmstadt, Germany

ABSTRACT

Cloud Computing is an emerging technology promising new business opportunities and easy deployment of web services.

Much has been written about the risks and benefits of cloud computing in the last years. The literature on clouds often points out security and privacy challenges as the main obstacles, and proposes solutions and guidelines to avoid them. However, most of these works deal with either malicious cloud providers or customers, but ignore the severe threats caused by unaware users.

In this paper we consider security and privacy aspects of *real-life* cloud deployments, independently from malicious cloud providers or customers. We focus on the popular Amazon Elastic Compute Cloud (EC2) and give a detailed and systematic analysis of various crucial vulnerabilities in publicly available and widely used Amazon Machine Images (AMIs) and show how to eliminate them.

Our Amazon Image Attacks (AmazonIA) deploy an automated tool that uses only publicly available interfaces and makes *no* assumptions on the underlying cloud infrastructure. We were able to extract highly sensitive information (including passwords, keys, and credentials) from a variety of publicly available AMIs. The extracted information allows to (i) start (botnet) instances worth thousands of dollars per day, (ii) provide backdoors into the running machines, (iii) launch impersonation attacks, or (iv) access the source code of the entire web service. Our attacks can be used to completely compromise several real web services offered by companies (including IT-security companies), e.g., for website statistics/user tracking, two-factor authentication, or price comparison. Further, we show mechanisms to identify the AMI of certain running instances.

Following the maxim "security and privacy by design" we show how our automated tools together with changes to the user interface can be used to mitigate our attacks.

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

Categories and Subject Descriptors

K.6.5 [**MANAGEMENT OF COMPUTING AND INFORMATION SYSTEMS**]: Security and Protection

General Terms

Security

Keywords

Cloud Computing, Privacy, Awareness, Attacks, App Store, Virtual Machine Images, Secure Shell

1. INTRODUCTION

Cloud computing offers fine-grained IT resources, including storage, networking, and computing platforms, on an on-demand and pay-per-use basis. The high usability of today's cloud computing platforms makes this rapidly emerging paradigm very attractive for customers who want to instantly and easily provide web-services that are highly available and scalable to the current demands [36].

In the most flexible service model of cloud computing, *Infrastructure-as-a-Service* (IaaS), customers can build entire *virtual infrastructures* by renting resources like storage, network, and computing platforms in form of virtual machines with administrative access to the whole operating system. Images of these virtual machines can easily be shared to be run by other users, similar to an app store for the cloud.

Albeit the various advantages of cloud computing, serious concerns about security and privacy hinder many users from "going into the cloud". Most solutions to preserve security and privacy in the cloud proposed so far consider potentially faulty/malicious cloud providers or technical savvy/rogue customers. However, the much more serious and ubiquitous threat of unaware users who unintentionally harm their own or others' security or privacy is often overseen.

The main goal of this paper is the investigation and evaluation of security and privacy threats caused by the unawareness of users in the cloud. Although the methods and techniques described in this paper are applicable to arbitrary IaaS providers, we focus on one of the major cloud providers, Amazon's Elastic Compute Cloud (EC2) [5] and adapt our terminology accordingly. In the following, we describe the players involved in the (Amazon) Cloud App Store and the resulting security challenges.

The Cloud App Store. As shown in Fig. 1, the Cloud App Store involves a Provider and typically two kinds of users, Publisher and Consumer. The *Provider*, Amazon in

our case, operates the IaaS cloud infrastructure, authenticates users and bills them for the resources they consumed.

The *Publisher* creates and publicly offers cloud apps, called *Amazon Machine Images* (AMIs). For this, he selects an existing AMI (AMI-1 in Fig. 1), instantiates it (Instance-$1_{\text{AMI-1}}$), logs into the running instance to configure it, and finally publishes a snapshot as a new AMI (AMI-2).

The *Consumer* selects this AMI from a list of available AMIs, instantiates it (Instance-$2_{\text{AMI-2}}$), and uses it for her purposes. Optionally, a Publisher can declare an AMI as *paid AMI* to earn money from Consumers invoking it.

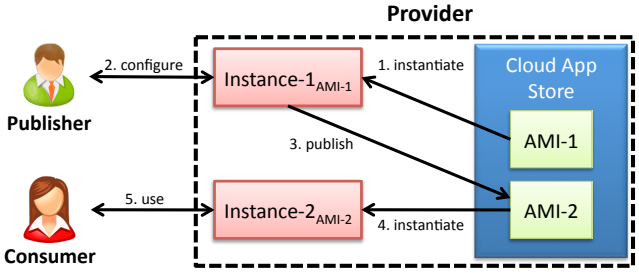

Figure 1: Basic System Model of Cloud App Store

The Cloud App Store poses security challenges for both, Consumers and Publishers (see also [48, 17]).

Security of Consumer. The Consumer must trust the Publisher not to include any malware into the AMI. Such a malicious AMI could contain a Trojan horse that spies on or modifies the Consumer's data, or a backdoor for malicious remote login. Even though full protection against such malicious AMIs is almost impossible, filters, virus scanners, and rootkit detectors could provide at least some level of protection [48].[1]

Security of Publisher. The Publisher on the other hand might accidentally publish AMIs that contain highly sensitive information. Examples include keys, credentials, passwords, command history/log files, or source code.

Although Amazon's user guide recommends to ensure that all confidential information is removed before publishing an AMI [12, Sharing AMIs Safely], many users seem to be unaware of the crucial consequences of ignoring these recommendations, do not have the appropriate tools at hand, or simply forgot private data in their AMIs.

The Gap between Theory and Practice. The Provider could filter AMIs for Trojans, backdoors, or confidential information to reduce the chance of malicious or sensitive data within AMIs. This was proposed in [48], but although the automated filtering system presented in that paper seems to be used already within the IBM SmartCloud [32], the explicit filtering rules are not available to the public.

In contrast, Amazon currently does not provide automated scanning of public AMIs as they are not responsible/liable for what users do with their own data. Though Amazon quickly reacts on incidents reported to their security hotline and informs affected customers, e.g., those running an AMI in which a backdoor was found [15].[2]

In this paper we show that these previously reported incidents are only the tip of the iceberg and many of the publicly available AMIs have severe security vulnerabilities leaking highly sensitive data.

Our Contribution and Outline.

After summarizing related work in §2 and giving background information on the Amazon Web Services (AWS) in §3 we present the following contributions.

Extraction of Sensitive Information from Public AMIs (cf. §4). Through an extensive analysis we were able to extract highly sensitive information from several publicly available EC2 AMIs. To make the analysis cost and time effective we developed an automated tool that uses different search strategies and exploits technology specific aspects of the Amazon cloud. The costs for running our attack were less than $20 while the information we extracted from the AMIs would allow an attacker to cause financial damage of several $10,000 per day and could severely harm the reputation of several companies that operate services in the cloud. After testing overall 1225 AMIs we got hold of the source code repositories, administrator passwords and other types of credentials of various web service providers.

SSH Vulnerabilities in AMIs (cf. §5). We discovered several vulnerabilities in AMIs that are introduced by incorrect usage and configuration of SSH. About one third of the tested 1100 public AMIs in Europe and the US-East region contain an SSH backdoor, i.e., a (forgotten) public key that allows remote login for the Publisher. We identified multiple instances that use the same SSH host key which allows an external attacker to correlate these instances running the same or a similar AMI, identify candidates for corresponding public AMIs, and mount several attacks, e.g., host impersonation.

Countermeasures (cf. §6). We provide several mechanisms to protect against our attacks on public AMIs. Besides organizational measures we propose to use our tools to enhance the security of the interfaces for publishing AMIs and also extensions to the interface of the Cloud App Store.

2. RELATED WORK

In this section we briefly revisit previous work on the security challenges of publicly sharing Virtual Machine (VM) images (AMIs in our terminology) on which we build our practical attacks. Afterwards we review the main related work on general cloud security, security aspects specific to the Amazon cloud, and methods for searching private data.

VM Image Analysis.

As summarized in §1, security and privacy risks for the Consumer and Publisher when sharing VM images have been identified in [48]. Shared VM images may contain either malware that was intentionally or unintentionally included by the Publisher. To protect against these threats, the authors propose filtering of VM images by the Provider which has been implemented in the Mirage image manage-

[1] In principle, this is similar to mobile app stores where downloaded apps must be trusted as well. Recently, Google's mobile app store withdrew 25 Android apps that were infected with malware [13]. As such attacks also harm the reputation of the mobile app store provider, some providers already review new apps submitted to the store to ensure that they perform as expected [9, 31].

[2] "For security reasons, we (Amazon) recommend that any instance based on a publicly available AMI that is distributed with an included SSH public key should be considered compromised and immediately terminated." [14]

ment system [38]. The IBM SmartCloud [32] deploys a system that is presumably based on (a mechanism like) Mirage for automated patching [52] and periodical malware scans [41] of public VM images.

We show that the risks pointed out in [48] ubiquitously occur in Amazon's Cloud App Store and have more severe consequences than previously thought. As our results show, Amazon does not apply any centralized filtering as proposed in [48], and considers this as the responsibility of cloud users. As countermeasure against these threats we propose tools and extensions to the user interface of the Cloud App Store that allow even technically less skilled users to protect their published AMIs from containing sensitive data.

General Cloud Security Challenges.

The risks and threats for cloud computing as analyzed in [23, 26, 21, 45] concern mainly the vulnerabilities inherent to cloud infrastructures, e.g., protection of the outsourced data and computations against eavesdropping and illegitimate modifications; new threats induced by sharing physical resources with other users' VMs (*multi-tenancy*); non-availability of the users' outsourced data and cloud-based services; or vendor lock-in to a single provider.

The security threats induced specifically by the usage model and capabilities of VMs, e.g., massive scalability and VM snapshots, i.e., copies of the current state of a VM, were discussed in [29, 40, 24]. These threats include, e.g., the rollback of a VM to an already compromised state, the conflict between traditional security management systems and the VMs' flexibility, or the loss of entropy upon state rollback affecting the freshness needed for cryptographic mechanisms and protocols.

Recent guidances and best practices aim at mitigating these threats and securing cloud computing [35, 22]. However, we show that many Publishers and Consumers do not adhere to these recommendations.

Amazon Specific Cloud Security Challenges.

The documentation of the Amazon Web Services (AWS) [8] contains several relevant security guidelines. Moreover, the Elastic Compute Cloud (EC2) [5], an integral part of AWS, has recently been subject to scientific and industrial security research as summarized next.

AWS Security Recommendations. An overview of the security processes in the AWS cloud infrastructure is given in [47]. Further, Amazon provides security guidelines and best practices on how to use AWS [19], including advises and references [43] on how customers can secure their AWS credentials in EC2 instances with respect to the risk of unintentionally embedding them in public AMIs. However, the proposed solutions require technical knowledge of the AWS mechanisms which conflicts with the high usability of the Cloud App Store. Thus, Cloud App Store users are either unaware of the security risks/guidelines or are only capable of using the store but not of implementing the security guidelines. This assumption is underlined by the high number and severity of our findings.

VM correlation. The possibility to gain insight into the EC2 network topology, and how to exploit this knowledge for placing a malicious VM A purposefully on the same physical host as a specific foreign VM B, and finally to eavesdrop on VM B via side-channels, was demonstrated in [39]. Our SSH correlation attack (cf. §5) provides additional information about running VMs that can be used to enhance their approach.

Malicious VMs. The attacks presented in [17] exploit design flaws of the EC2 Cloud App Store such as phishing by publishing a malicious AMI that appears high in the list of available AMIs. Our countermeasures proposed in §6 can be used to mitigate these attacks.

Private Data Search.

Several methods and sources exist to search for unintentionally leaked private data in public resources. Examples include recovery of sensitive data from second-hand harddrives [28], or "Google hacking" [16, 46] which allows to query the Google search engine to find private information in its indices, e.g., private keys, hashed passwords, or private information about a person.

3. BACKGROUND ON AWS

In this section we recall the main aspects of the Amazon Web Services (AWS) [8]: the Amazon Elastic Compute Cloud (EC2) [5] in §3.1 and authentication to AWS in §3.2. For detailed information we refer to Amazon's documentation on AWS [18]. Readers already familiar with AWS can skip this section.

3.1 Amazon's Elastic Compute Cloud (EC2)

In 2006, Amazon introduced the Elastic Compute Cloud (EC2) [5] as part of the Amazon Web Services (AWS). It allows users to run Virtual Machines (VMs) on-demand on the infrastructure provided by AWS. The VMs behave similar to a physical server and contain a full-blown operating system (currently supported are various Linux distributions, FreeBSD, OpenSolaris, and Windows). Running VMs are called *instances* and isolation between instances is enforced by the XEN hypervisor [19]. Fig. 2 illustrates the instantiation of VMs from Amazon Machine Images (AMIs) via the AWS Cloud App Store. When a Consumer starts a VM, she has to specify 1) an AMI from which the instance is derived (AMI-ID), 2) a type defining the resources available to the instance (Type), and 3) the geographical region in which the VM is deployed (Region) as described next.

Region and Type.

The Region defines one out of five EC2 data centers (two in the US, one in Europe, and two in Asia). To guarantee failsafe operation, each data center of a region is currently split into two or more independent *availability zones*.

The Type specifies how many resources are allocated for the instance (CPU, RAM, temporary disk space, speed of the network connection) and determines the costs ranging from \$0.02 to \$2.10 per instance operation hour. Besides costs for running instances, the user is also monthly charged for I/O usage and consumed network bandwidth.

Amazon Machine Images (AMIs).

Amazon introduced the concept of pre-built VM image templates, called Amazon Machine Images (AMIs) for rapid deployment of instances. An AMI contains a whole operating system together with applications and data. When an instance is started, a copy of the selected AMI is booted and control over the instance is handed over to the user. AMIs are managed in the *Cloud App Store* and are either provided

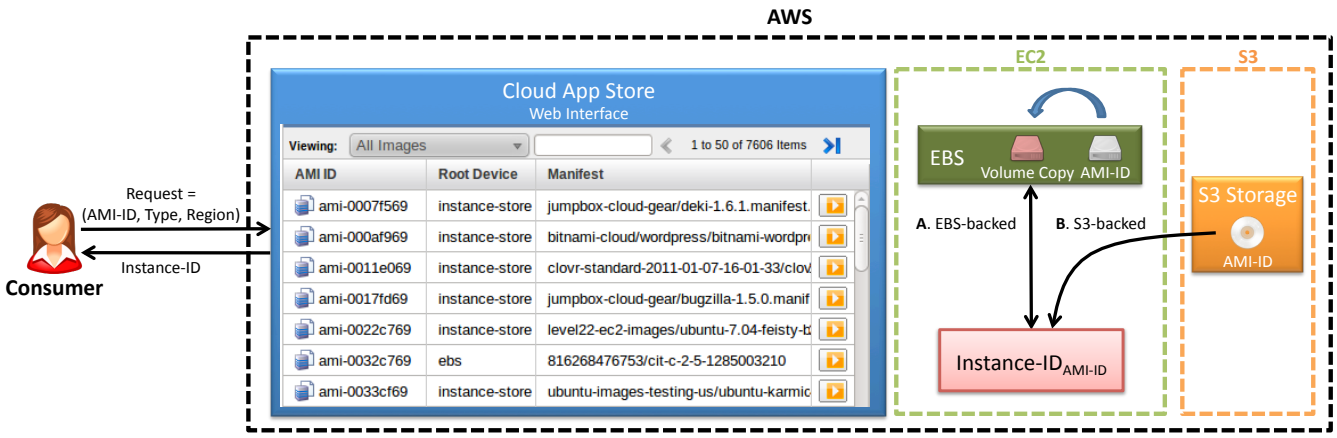

Figure 2: VM instantiation in Amazon AWS. The Consumer chooses the image (AMI-ID), resources (Type), and availability zone (Region) for her VM on the Web Interface of the AWS Cloud App Store. Depending on the type of the AMI, the VM is instantiated (Instance-ID$_{\text{AMI-ID}}$) either as (A) EBS-backed or (B) S3-backed.

directly by Amazon or by third party publishers. Users can take these public AMIs to create their own AMIs which are either kept for themselves (private AMIs), made accessible to a group of users (shared AMIs), or made publicly available for every user of EC2 (public AMIs) as shown in Fig. 1. AMIs are further distinguished by the storage type they are based on – either S3 or EBS – as described next.

S3-backed AMIs. S3-backed AMIs are stored on the highly available Simple Storage Service (S3) [7]. As shown in Fig. 2, S3-backed AMIs are instantiated by first copying the image onto the hard drive of a physical EC2 node which then boots the image. A new S3-backed AMI can be created from within a running S3-backed instance by a process called *bundling*, which stores the current state of the instance's file system on S3 and registers this state with the Cloud App Store as new AMI. The data in an S3-backed instance is only persistent for the life of the instance and lost upon instance termination or failure. This resembles the usage model of a live CD.

EBS-backed AMIs. EBS-backed AMIs reside on the Elastic Block Storage (EBS) [4]. EBS offers persistent, attachable block devices, called *volumes*, which can hold an arbitrary file system. When a user instantiates an EBS-backed AMI, a *bitwise* copy of the image is created on EBS and the VM is started from this new image as shown in Fig. 2. Storing the instance's data persistently on an EBS volume enables the user to stop the execution of an EBS-backed VM. He then has only ongoing costs for the storage occupied by the block device. New EBS-backed AMIs are created by storing a bitwise copy of the current state of a volume, called *snapshot*, on EBS and registering this snapshot with the Cloud App Store as new AMI. EBS volumes cannot only be used to hold bootable AMIs, but are also a general way to store persistent data. They can be attached on-the-fly to running instances (comparable to a USB flash drive in the cloud).

Networking.

All instances are executed in an environment which provides logging, monitoring, and security capabilities such as a simple firewall for inbound traffic (cf. [20]). On startup, the instance is assigned an external IPv4 address for Internet connectivity and an internal address for communication with other EC2 instances. The user is only charged for data traffic with the Internet over the external address.

3.2 Authentication in AWS

AWS uses different authentication mechanisms to provide authenticated access to the AWS account and to running instances as described next.

Authentication to the AWS Account. During the initial account creation and verification, an AWS Customer associates her email address with an AWS account, selects a password, and provides credit card and personal information for the monthly billing. After successful verification by Amazon she obtains a password to log into a web management system where she can inspect log files and the current account activity, change personal information, and manage instances. In this web management system, the Customer can register credentials for an Application Programming Interface (API) to control the life cycle of an instance by means of tools provided by 3^{rd} parties or Amazon [34]. We refer to those credentials as *API keys*.[3] The API keys can be used for example to start or terminate instances, create EBS volumes, or to register public images. Moreover, API keys are required within an S3-backed instance during the bundling of the same instances in order to access the S3 storage. However, API keys do not provide direct access to personal or credit card information.

Authentication to Instances. After an instance has been started, the control needs to be securely transferred to the Customer by providing her with a secure and authenticated login channel. For this, Linux/Unix-based instances commonly use Secure Shell (SSH) [51]. Here, a customer generates an SSH key pair $(\mathsf{pk}, \mathsf{sk})$ and registers the public key pk with AWS. Upon instantiation of an AMI, this key is automatically made available to and imported by the SSH server running in the newly created instance and can be used for secure logins of the user who holds the corresponding secret key sk.

[3]Technically, API keys provide authorized access to a SOAP, Query, or REST-based web service.

4. AMI PRIVACY ANALYSIS

By systematically examining only little more than 10% of all public Linux-based AMIs we were able to extract a lot of private information. The results of our analysis are summarized in §4.1. To automate the analysis we implemented a tool as described in §4.2. We calculate the costs for analyzing all AMIs in §4.3 and discuss the reasons for and implications of our analysis in §4.4.

4.1 Our Findings

We analyzed a total of 1225 AMIs in the European and US-East region of the Amazon EC2 cloud. In our privacy analysis of these AMIs we discovered AWS API keys, private keys and credentials, and private data, including source code. For each of these categories we describe the possible threats and our findings next. A summary of our findings in EBS-backed AMIs is given in Tab. 1.

Finding	US-East-1		EU-West-1	
Analyzed AMIs	550	(100%)	550	(100%)
AWS API Keys	12	(2%)	2	(0.35%)
SVN Credentials	4	(0.7%)	3	(0.55%)
SSH/SSL User Keys	14	(2.5%)	5	(0.9%)
SSH Host Keys	205	(37%)	122	(22%)
SSH Backdoor	253	(46%)	93	(16%)

Table 1: Summary of the Findings of our Privacy Analysis of Public EBS-backed AMIs (April 2011)

4.1.1 AWS API keys

Our most significant findings are API keys (cf. §3.2) which could be used to abuse the AWS account of the AMI Publisher. Those keys had been used by the Publisher of the examined AMIs to query the AWS API, in most cases for authorization before publishing and registering the running instance as new AMI.

Threats. The presence of AWS API keys in public AMIs is a very serious threat to the owners of those keys. It allows the adversary to destroy the victim's virtual infrastructure or to create his own infrastructure at the expense of the victim [23]. For example, an attacker could discover multiple API keys in the public images of other customers. He then creates a public image of his own, which is configured to mount a DDoS attack against a chosen target. Using the discovered API keys he is able to instantiate several instances of his DDoS-image and run the attack at the expenses of the API keys' holders. This is in particular a problem as it is currently not possible to set a maximum limit of costs to prevent excessive usage in case of an accident (e.g., misconfiguration) or security breach [1]. With one AWS API key an attacker can cause costs of more than $3000 per day for running instances and additional charges for traffic and storage (cf. calculation in §A). Moreover, before the introduction of improved key management with the *Identity and Access Management (IAM)* [10] service in September 2010, customers had to use a single API key to control all their AWS services like EC2, the S3 storage or the SimpleDB database [42] engine. It seems that many EC2 customers have not migrated to this new service yet, e.g., because of unawareness or of being technically overwhelmed, and thus a discovered API key can be used to extract private information also from AWS services beyond EC2.

Findings. We retrieved 20 AWS API keys from S3 and EBS-backed AMIs in the US-East and European region.

4.1.2 Private Keys and Credentials

Private keys and login credentials are quite frequently stored or cached on a computer's hard drive. For instance, they are used to login to other hosts via SSH or to provide secure communication channels using SSL certificates.

Threats. Although most private keys are encrypted with a password when stored on disk, this data can be recovered [50]. Thus, these unintentionally published keys and credentials can compromise the security of the Publisher's IT infrastructure. For instance, an attacker can use private keys of SSH user authentication to log in to other hosts the key is authorized for. These hosts may be located within the virtual infrastructure of the Publisher in the cloud or even in the conventional IT infrastructure of the Publisher. Leaked private keys of valid SSL certificates that are used on commercial websites/services would allow an attacker to carry out man-in-the-middle attacks on those sites or implement ideal phishing attacks.

Findings. We discovered 16 unencrypted private keys for SSH user authentication, 3 unencrypted private keys of valid SSL certificates, and various other private keys. Most of the SSH keys we found were associated with an account with administrative privileges.

4.1.3 Private Data

Most AMIs contain information about their Publisher, resulting from the configuration and usage of the instance from which the AMI was created. This information can be obtained by analyzing, e.g., log, cache, or configuration files.

Threats. Private data can be used to profile Publishers, e.g., to identify their name, email address, other hosts they connected to, browser history, or their affiliation. The value of private information found by a random attacker is hard to estimate. However, leakage of private information might harm the reputation of a person or company, cause financial harm, or even entail legal consequences.

Findings. In one particular AMI we found a picture of the owner of the AMI, holding a badge stating his name and employing company. In other cases we were able to find out the name of the AMI publisher by using information in the bash history and the name of the SSH key used for login. However, we cannot give concrete numbers as privacy violations are hard to categorize.

4.1.4 Source Code

The Amazon cloud is well suited for software development and testing of new software products and especially start-up companies make use of it.

Threats. Obtaining and analyzing source code of new proprietary applications or websites allows an attacker to steal unpublished ideas and concepts. Moreover, it gives him enough insight to mount further attacks on these products (or their users) and even the ability to insert malicious code if he gains access to the source code repository.

Findings. We obtained credentials for 7 different private source code repositories. Most of these repositories were operated by businesses and the repository's copy stored in the AMI contained highly valuable intellectual property and hard coded passwords for administrative access.

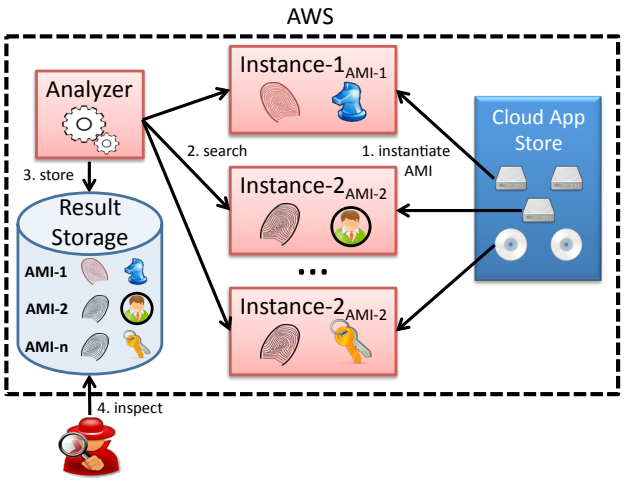

Figure 3: Approach for Analysis of AMIs

4.2 Tool for AMI Privacy Analysis

Next, we explain our tool for privacy analysis of AMIs and give details on its technical background and our strategy for cost- and time-efficient attacks. We have implemented our tool in the Python programming language and made use of the Boto library [34] in order to access the AWS API. We concentrated our analysis on Linux based AMIs, which form the majority of the provided AMIs (see [11] for detailed statistics), and thus designed our tools specifically for the Linux directory structure and key formats.

Fig. 3 depicts our approach of iteratively inspecting public AMIs for private information. It consists of (1) the instantiation of target AMIs, (2) the search for fingerprints, private information, or keys, and (3) the storing of the results in a data store. The last step (4) is a semi-automated inspection and analysis of all findings.

We describe how our tool accesses the data inside the AMIs (§4.2.1) and extracts private content from it (§4.2.2).

4.2.1 AMI Access

In order to analyze an AMI we mount its volume directly into the file system of our Analyzer instance (cf. Fig. 3). By running it in the cloud, we avoid expensive network traffic (e.g., downloading the image) and are able to profit from the scalability of cloud computing. Further by mounting target AMIs as volumes or via the SSH File System (SSHFS) [44] we minimize assumptions on the environment our analysis tools runs in (e.g., software dependencies). We describe the two methodologies in more detail next.

SSHFS. The SSHFS allows to mount remote volumes into the local file system by means of the SSH protocol. As SSH is the standard tool to perform system administration on EC2 Linux based instances and thus is supported by most AMIs, this approach is widely applicable. Although this method does not make any restrictions on where the Analyzer is run (e.g., it could be executed on a local host), deploying it on a dedicated EC2 instance reduces the network overhead significantly. Moreover, Amazon does not charge for cloud-internal data traffic.

EBS-Mount. Analyzing EBS-backed images is substantially faster than using SSHFS as EBS volumes provide raw disk access. As described in §3.1, EBS-backed images are stored on and booted from EBS volumes. Therefore, our analyzer tool first starts a public EBS-backed AMI to instantiate it on a new EBS volume. By dissolving the mapping between this instance and the new volume its file system is stored on, the tool obtains the EBS volume that contains the AMI to be tested. This EBS volume is then mounted by the Analyzer instance. This method allows also to analyze EBS-backed AMIs that are explicitly configured without SSH access (e.g., to protect the source code of an appliance).

Pattern	Explanation (Common Usage)
*.pem	Specifies a file containing a private key, public key or a certificate
*.priv	File extension for *priv*ate keys
*.pub	Used to store *pub*lic keys
*.crt	*Cert*ificates
id_rsa	Default file name of private SSH keys
*.gpg | *.pgp	Files related to use with the encryption and signing application GPG/PGP
*.jks	Acronym for Java key-store [37]
secret | *key* | *private*	Potentially interesting files
.bash_history	User's history of executed commands
.svn/ | .git/ | .hg/	Common source code repositories

Table 2: Example of Search Patterns (∗ denotes the wildcard character and | alternatives)

4.2.2 Extraction of Private AMI Content

Once an AMI is mounted into the file system of the Analyzer instance, the Analyzer searches for keys and private information in the AMI and stores its findings locally for later evaluation. Some of the patterns for file or directory names that our tool is searching for are listed in Tab. 2. These patterns include common names for key files/stores, shell history files, and source code repository directories.

The primary targets of the automated search are high-value findings, i.e., private keys and credentials, usually to be discovered in the home directories of users, the home directory of the superuser (root), and common locations for (custom) programs and configuration files.

If a search result indicates the presence of further, not automatically discovered private data, e.g., a source code repository directory was found or the shell history is non-empty, we manually investigate the corresponding AMI for further findings such as source code or private information.

Forensic Analysis of EBS Volumes. As described in §3.1, EBS-backed AMI instances are created as a *bitwise* copy of the original AMI volume they are derived from and hence provide access to raw blocks on the newly created volume. This enables our analyzer to apply forensic methods [28, 27] in order to recover and scrutinize deleted files which may contain private information.

4.3 Costs for AMI Privacy Analysis

Our analysis tool described in §4.2 currently takes approximately 10 minutes to start and completely analyze an AMI. However, as Amazon charges for each started instance hour [3], the costs for starting an AMI are one instance hour. For EBS-backed AMIs the smallest available type is the "Micro" instance for $0.03/h. The costs for S3-backed AMIs

depend on the AMI's architecture: a 32-bit S3-backed AMI can be started as "Small" instance for $0.10/h while 64-bit AMIs only support the "Large" instance type for $0.40/h. A medium scale analysis can even be done "for free" as Amazon offers free services to new customers in the first year (currently 750 "Micro" instance hours per month plus some storage and traffic volume).

Costs of Our Analysis. During our analysis, we examined 1225 AMIs (100 S3-backed 32-bit AMIs, 25 S3-backed 64-bit AMIs and 1100 EBS-backed AMIs). As we have split our analysis into two months, the costs for starting the EBS-backed AMIs were covered by the free offer and for the S3-backed AMIs we paid in total $20.

Costs for Analyzing All Public AMIs. Scanning *all* free and Linux based 9864 public AMIs that are currently available[4] (cf. Fig. 4) would cost approximately $1550: Starting all AMIs would cost around 3058 "Micro" instance hours, 4449 "Small" instance hours, and 2357 "Large" instance hours to start all AMIs. As the Analyzer component of our tool takes on average 10 minutes per AMI this adds $\frac{9864*10}{60} = 1644$ "Micro" instance hours. We note that such an exhaustive search can be highly parallelized and therefore carried out very fast, e.g., in less than one day with 70 analyzing instances, without increasing the costs.

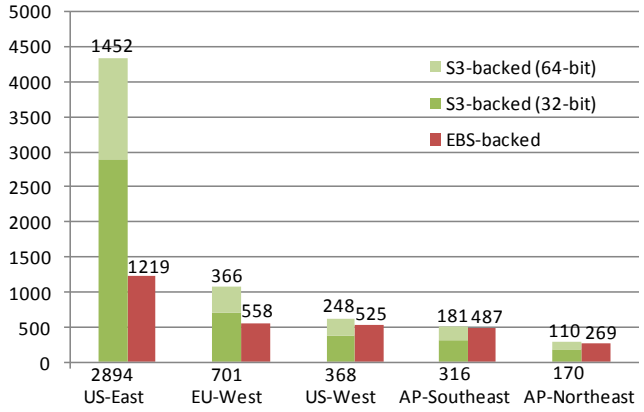

Figure 4: Linux Based Public AMIs per Region

Attack Optimization. One could even reduce the costs for the attack to almost zero by first scanning public AMIs that are likely to contain AWS API keys and then using these keys to start further analyzer instances at the expense of these initial victims. Candidates for the initial attacks are AMIs from inexperienced or unaware users, i.e., those who published only few AMIs or give suspicious names to their AMIs such as "backup" or "test".

4.4 Discussion

We discuss the reasons for and implications of our successful extraction of private information from public AMIs.

Reasons for Forgotten AWS API Keys.

We discovered AWS API keys in both S3- and EBS-backed AMIs. Our inspection of S3-backed AMIs revealed that those keys were unintentionally included during the bundling process when the AMI was created. Since S3-backed instances require an API key within the instance for bundling, the success rate to find API keys within this kind of AMIs is high. Our inspection of EBS-backed AMIs showed, that the Publishers used their API keys within their instances in order to access further AWS services such as S3 or EBS storage, and that the Publishers simply forgot to delete the key before publishing.

EBS Specific Problems.

In general our analysis shows that EBS-backed images contain more private information than S3-backed AMIs. We attribute this to the fact that data in EBS-backed instances is persistent (cf. §3.1). In fact, EBS-backed AMIs are much more dangerous than S3-backed AMIs for various reasons:

Loss of intellectual property. Publishers who want to protect their intellectual property by limiting access to the actual VM should not distribute it as an EBS-backed AMI. Indeed, in our analysis we found appliances which were configured explicitly without SSH access but could easily be inspected by mounting them as EBS volumes (cf. §4.2.1).

Forensic attacks. EBS backed AMIs or EBS volumes that once contained *any* valuable information, should not be made public. By applying forensic methods [27, 28] on a raw EBS volume we were able to reconstruct and inspect several deleted files, containing private information.

Snapshots. Snapshots are non-bootable EBS volumes with the purpose to ease sharing of (large volumes) of data. Snapshots can be created as new volume and filled with data, but they can also be created from instances. Thus, they also bare the risk of unintentionally publishing (forensically retrievable) private information. For instance, a first analysis of various snapshots revealed a private picture of the Publisher.

5. CLOUD SPECIFIC SSH THREATS

In this section we discuss vulnerabilities of the Secure Shell (SSH) protocol [30], which result from the cloud's dynamic nature and the usage model of AMIs. In particular, we evaluate SSH-based backdoors of instances (§5.1) and present new attacks due to the SSH-based identification of the AMI from which an instance is derived (§5.2).

SSH provides confidentiality and integrity, supports asymmetric key pairs for user authentication $(\mathsf{sk}_u, \mathsf{pk}_u)$ and for host authentication $(\mathsf{sk}_h, \mathsf{pk}_h)$. The SSH protocol is well-established and is the primary method for remote administration of Linux based instances in EC2 (cf. §3.2). For details on authentication in SSH we refer to §B.

5.1 AMIs with SSH Backdoor

Recently, Amazon informed several of its customers that a public AMI contained an SSH user authentication key pk_u and thus a backdoor allowing the publisher of the AMI who holds the corresponding sk_u to log into instances derived from that particular AMI [15]. Amazon strongly advises customers to terminate such instances and regards them as compromised [14]. However, this problem is not an isolated incident and, as we will show in this section, many of the publicly available AMIs contain SSH backdoors.

Threats. An SSH user authentication key deployed in a public AMI poses a severe threat to the AMI Consumer's privacy, as the key owner is able to log in to the affected Consumer's instances. A potentially malicious Publisher

[4]Note that the number of public AMIs changes frequently.

is thus able to deliberately eavesdrop or modify the Consumer's data and services in the instance.

Findings. Our analysis revealed that 30% of the 1100 analyzed EBS-backed AMIs in the European and US-East region at the time of the analysis contained pk_u and thus a backdoor for the AMI Publisher (detailed numbers are given in Tab. 1). By examining the affected AMIs we discovered that the backdoor problem is not limited to AMIs created by individuals, but also affects appliances of well-known open-source projects and even of companies that offer IT-security products. Moreover, our investigation yields that an overwhelming number of AMIs allows SSH login and that most users have administrative privileges[5].

5.1.1 Approach

Our approach to identify AMIs with a SSH backdoor is based on our analysis tool presented in §4.2. For this, we analyzed the *.ssh/authorized_keys* file. It is located in the users' home directories and contains the public keys pk_{u_i} of users that are allowed to log in.

5.1.2 Discussion

Reasons for Forgotten Public Keys. The reasons why such a huge number of AMIs contain backdoors for the Publisher are similar to those for unintended inclusion of private information as discussed in §4. However, the Publisher has no strong incentives to remove his public key before publishing an AMI, as the only negative consequence of not doing this is that he can be traced among AMIs.

Countermeasures against SSH Backdoors. In case a Consumer decides that she wants to run a particular AMI (e.g., because of the unique features offered by this AMI), she should check the *authorized_keys* file in every home directory and delete all public keys pk_u (except for her own). In addition to that she should examine the configuration of the SSH server in case other authentication methods (like insecure password authentication) are enabled or other locations for authorized keys are specified. Another countermeasure is to use the Amazon provided inbound firewall to generally restrict the IP range from which users are allowed to login via SSH. It also supports features for the implementation of a multi-tier infrastructure with a central gateway that validates access attempts to other instances [20].

5.2 AMI Identification via SSH

Some public AMIs contain an SSH host key pair $(\mathsf{sk}_h, \mathsf{pk}_h)$ which is generally used to prove the identity of the remote host to the client during login (cf. §B for details). Usually, a host key pair is generated from fresh entropy when installing SSH or the operating system. However, in cloud apps, where instances of an AMI are only a copy of the hard drive's state of an already configured machine, the SSH host key pair is not regenerated. Hence, all instances of an AMI are using the same SSH host key pair.

The severity of this vulnerability can be amplified by combining it with the technique for extracting information from public AMIs described in §4.2. The attacker can examine several (or all) public AMIs and extract the SSH host key pairs $(\mathsf{sk}_h, \mathsf{pk}_h)$ from it. Afterwards, he can use the host key fingerprint $\mathsf{f}(\mathsf{pk}_h)$ of an instance to look up the corresponding public AMI and the secret key sk_h.

[5]Many Linux distributions for EC2 allow the default user to execute commands as superuser via "sudo".

Attack	Prerequisites
Correlation of System Config.	-
Whitebox Attacks	public AMI identified
Impersonation Attacks	
Man-in-the-Middle Attacks	
Co-Location Attacks	
Phishing Attacks	owner of public AMI

Table 3: Attacks based on identical SSH host keys.

We describe the consequences of our attacks in §5.2.1, our findings in §5.2.2, our approach in §5.2.3, and the underlying causes in §5.2.4.

5.2.1 Threats

The possibility to identify instances that are using the same SSH host key or potentially even identifying the corresponding public AMI from which these instances are derived allows a variety of attacks as summarized in Tab. 3 and described in the following.

Correlation of System Configurations. Even if the attacker only detects instances with an identical fingerprint $\mathsf{f}(\mathsf{pk}_h)$, but does not manage to identify the corresponding AMI (e.g., because it is not public), he knows that these instances are likely to be instances of the same AMI (or a derived AMI). This enables him to discover test systems or forgotten servers which are not as secure as the productive system.

If the attacker identifies the corresponding public AMI he has more options:

Whitebox Attacks. The attacker can examine the contents of the AMI for potential vulnerabilities and misconfigurations like usage of default passwords or insecure services to launch attacks on the victim instance.

If the attacker extracts the host key pair $(\mathsf{sk}_h, \mathsf{pk}_h)$ from the AMI and has control over the network between the Consumer and the Provider (e.g., [25]) he can launch impersonation and man-in-the-middle attacks on SSH:

Impersonation Attacks. The attacker can redirect the SSH connection attempt of the Consumer to an instance under his control. In this case, verification of the host key fingerprint $\mathsf{f}(\mathsf{pk}_h)$ does not protect the Consumer as the attacker is able to equip his fake instance with the host key pair obtained from the public AMI. The Consumer may get suspicious that the impersonating instance is only similar to the expected environment, but there is a chance that the Consumer reveals private information or credentials before recognizing this, e.g., passwords upon login to other services. The attack especially works when automated scripts are used, e.g., a script that automatically uploads a backup copy onto an instance in the cloud. By impersonating this instance the attacker can obtain such a backup copy.

Man-in-the-Middle Attacks. If SSH is configured for password-based user authentication, an attacker can use sk_h to play as man-in-the-middle and eavesdrop or modify the communication between the Consumer and the VM.

Co-Location Attacks. The AMI type narrows down the possible instance type as 32-bit AMIs do not support the more powerful instance types because they cannot make use of the offered resources. This information can be ex-

ploited by an attacker to narrow down the search space in co-location attacks [39].

Phishing Attacks. If the attacker himself is the Publisher of the executed malicious AMI he can deniably identify victims of his own VM phishing attacks. The malicious AMI might contain intentionally embedded vulnerabilities [17, 48] or SSH backdoors (cf. §5.1).

5.2.2 Findings

Our analysis shows that approximately 29% of the 1100 analyzed AMIs (in US-East and EU-West) are misconfigured such that their derived instances do not recreate the SSH host key pair $(\mathsf{sk}_h, \mathsf{pk}_h)$ on first boot (cf. Tab. 1). We discovered that 62 distinct SSH host keys in the EC2 Tokyo region were used by more than one instance. We identified the public AMI of 11 of those host keys, which were contained in 278 instances running in this region.

Our experiment also revealed that $\frac{604}{2533} \approx 23\%$ of the instances in the Tokyo region with SSH access are using a non-unique host key pair $(\mathsf{sk}_h, \mathsf{pk}_h)$. However, we were not able to map all duplicate SSH host keys to public AMIs and believe that these misconfigured instances are derived from private AMIs owned by one user or a small group and are thus potential production or test systems with similar vulnerabilities (cf. *Correlation of System Configurations* described above).

5.2.3 Approach

Our approach for finding public AMIs using not freshly generated SSH host keys is illustrated in Fig. 5. Based on our analysis tool for public AMIs (cf. §4.2), we first extracted the SSH public key fingerprints $\mathsf{f}(\mathsf{pk}_h)$ of all AMIs in the Tokyo region and used this to obtain a mapping from 615 fingerprints to their corresponding AMI ID. Further, by making SSH connection attempts to all IP addresses in the Tokyo region's IP range (175.41.192.0/18), we gathered all mappings from IP addresses to host key fingerprints for the region. By matching the SSH fingerprints in both results, we found a mapping between the IP addresses of running instances to their corresponding AMI ID. Moreover, the diversity of SSH fingerprints in the region's IP range gave us the statistic about duplicated SSH host keys as explained in §5.2.2.

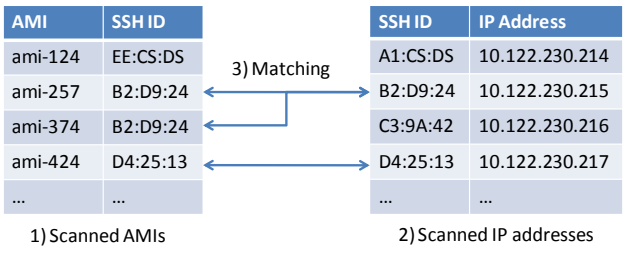

Figure 5: Matching Between AMIs and Instances

5.2.4 Discussion

In summary, our findings in §5.2.3 show that AMIs which do not recreate their SSH host key occur frequently.

The main reason for SSH host keys being included in public AMIs is presumably the same as for private information (cf. §4.4) and SSH user authentication keys (cf. §5.1.2) and, thus, countermeasures are similar. We therefore extend this discussion to the whole process of secure authentication and administration in the cloud, as it is negatively affected by the flexibility and usage model of computing clouds.

Alternatively, an attacker could use other publicly available information than the SSH host key to identify the AMI of an instance, e.g., the software version and configuration of a web-server running in the instance. However, the SSH host key is a much more distinguishing characteristic than the software configuration. Nevertheless, this is supplementary information to identify the AMI of an instance if more than one candidate image exists (e.g., *ami-257* and *ami-374* in Fig. 5).

Static vs. Dynamic Environments.

So far, the SSH protocol has been used mostly in and is designed for relatively static networks. Hence, manual verification of the host keys is only necessary in case of relatively infrequent changes to the underlying infrastructure (e.g., change of the server's IP address or hostname, or newly generated host key). In contrast, the cloud provides a highly dynamic environment with much shorter life time of instances and dynamically assigned IP addresses. Additionally, the use of preconfigured VMs instead of installation from scratch opens new vulnerabilities as shown in §5.2.

Trust on First Use.

This leads to the general challenge of secure authentication to started EC2 instances which is not sufficiently addressed in the current implementation of EC2 and even worse for AMIs with non-unique host keys. As described in §3 and shown in Fig. 2, the Consumer issues a request to start a public AMI and is returned the Instance-ID which can be used to look up the IP address of the instance (over the AWS API or Web Interface). When she connects to the IP address of this newly created instance, she has no prior knowledge on the contents of the AMI and its host key ("Trust-on-first-use", cf. [49]). The only option is to use an out-of-band method, provided by Amazon, to obtain the console output of the started instance to which the SSH server writes the host key fingerprint $\mathsf{f}(\mathsf{pk}_h)$ after the host key pair has been freshly created. However, as it takes a few minutes until the output of the console becomes available to the user, we assume that most users do not make use of this technique when establishing the first connection to a newly started instance. The situation is even worse when the instance's SSH server does not create a fresh host key pair where the user has no means to verify that he really connects to the machine he started.

This problem is aggravated when the instance is stopped and restarted with a newly assigned external IP address. In this case, the console output does not contain the host key fingerprint as no new host key was generated, and the user is forced to store the correct fingerprint upon the very first connect in order to be able to securely authenticate his instance after reboot or change of the external IP address.

6. COUNTERMEASURES

Finally, we present and discuss several countermeasures that create awareness, reduce the impact of lost credentials, or assist users in realizing that they accidentally published

sensitive information. Some general countermeasures have been proposed and discussed in [48], however, we believe that the high popularity and deep integration of Amazon specific services, e.g., S3 and EBS, into the EC2 Cloud App Store prevents adoption of these countermeasures as it would require fundamental changes to the architecture. We therefore present mechanism that specifically address the environment provided by Amazon and its limitations.

Organizational Measures.

Most problems that led to our attacks could have been prevented by the Publishers themselves. Especially as most public AMIs do not provide any additional value to other consumers, e.g., due to lacking functionality and documentation, they should not have been made public in the first place. Therefore, the first and most effective step should be to provide better information to users on the risks they are facing when publishing an AMI and to create problem awareness. In addition to that, despite the novelty of cloud computing and AMI publishing, companies and individuals should develop and follow guidelines, best practices, and processes for releasing their AMIs to the public. In this case AMIs do not differ from documents, software, or mediums containing confidential information (e.g., discharging of old hard drives §2). Another effective way to reduce the impact of an accidentally lost AWS API key is the recently introduced AWS Identity and Access Management (IAM) [10]. It allows managing of multiple security credentials for an AWS account with a different set of permissions which could be used to limit the damage an attacker can cause (cf. §A).

Tool Assistance for the Publisher.

Additionally, Publishers can be protected by enhancing Amazon's toolchain.

S3-backed AMIs. As discussed in §4.4 some Publishers of S3-backed AMIs have exposed their AWS API key used for authorizing the bundling operation [2]. We propose to extend the bundling command in a way that a warning is issued in case the AWS API key is included into the new bundle (cf. Fig. 6 for an example) or information about the key is stored in the command line history. Moreover, the extension can provide a *safe bundling* option to warn Publishers about data which is potentially private (e.g., by applying search patterns similar to those in Tab. 2).

```
$ ec2-bundle-vol -k /root/sk-HKZYCLO.pem
WARNING: The key sk-HKZYCLO.pem used to
    authorize the bundling operation will be
    included in the image file. Publishing the
    AMI may leak it to the public!
Do you want to proceed (y/n): n
```

Figure 6: Improved Bundling Tool

EBS-backed AMIs. Protection for EBS-backed AMIs has to be implemented differently than for S3-backed AMIs. EBS-backed AMIs can be almost instantly generated out of running instances and made public entirely over the web interface. As the underlying storage engine just instructs the storage layer to create a bitwise copy of a volume, we have to apply different countermeasures. Here, Amazon could extend their interface to inform the user on potential risks of private data being included and also offer tools to deal with them. This can be realized by a service (e.g., a public AMI) that is given an EBS volume as input, creates a report on potential privacy problems, and outputs a low-level sanitized volume to prevent forensic analysis.

Regular Scanning.

As described in [33], Providers have to find a trade-off between the benefits of providing security measures and the costs for implementing and operating them. However, we believe that it is possible for a provider like AWS to regularly scan the provided AMIs (cf. cost analysis in §4.3) while still ensuring that the Provider does not take any liability on undiscovered problems. As hackers are likely to re-implement the techniques described in our paper (especially given the high damage that can be caused), public cloud service providers should take immediate action.

For this, Amazon and other providers could adopt our tools, eventually increase their efficiency by integrating them into their infrastructure, and scan the Cloud App Store regularly, or when a new AMI is published. Although the scanning may result in more time and effort for manually verification of the results and informing affected customers of a discovered vulnerability, this approach would underline a proactive approach to security problems.

Improving the Cloud App Store.

As we have described in §5 a great deal of AMIs, deployed by a large number of consumers, does not satisfy even lowest quality standards (e.g., contains an SSH backdoor). This situation can be improved by combing an automated rating system that checks AMIs for predefined properties together with a reputation system that allows users to evaluate the usefulness and quality of an AMI. Such features are commonly available in today's mobile app stores as well as for online auction and shopping platforms to rate vendors and products, but still missing in Amazon's Cloud App Store and its web console. Moreover, the AWS web console should be extended to provide more information on particular AMIs like the date of publishing or more detailed descriptions about the AMI's purpose and its publisher. A process for submitting AMIs and extended documentation already exists, but this information [6] is not available in the web console. Additionally, well known and trusted AMI Publishers could be rewarded with a special status visible to the consumer (star/gold Publisher). Before earning such a status, a Publisher would have to earn good reviews or pass an analysis of their AMIs by Amazon.

Acknowledgements

The authors would like to thank Michael Waidner for helpful discussions.

We would also like to thank the AWS security team for their quick, professional response after we have notified them about our findings and their immediate reaction to protect their customers.

The research leading to this paper was in part funded by the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement n°257243 (TClouds project: `http://www.tclouds-project.eu`).

7. REFERENCES

[1] All Together Now: Amazon, we need those caps on billing. http://forums.aws.amazon.com/thread.jspa?threadID=50075#jive-message-217130.

[2] Amazon EC2: ec2-bundle-vol. http://docs.amazonwebservices.com/AmazonEC2/dg/2006-10-01/CLTRG-ami-bundle-vol.html.

[3] Amazon EC2 Pricing. http://aws.amazon.com/ec2/pricing.

[4] Amazon Elastic Block Store (EBS). http://aws.amazon.com/ebs/.

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

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

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

[8] Amazon Web Services. http://aws.amazon.com.

[9] Apple App Store Review Guidelines. http://developer.apple.com/appstore/guidelines.html.

[10] AWS Identity and Access Management (IAM) Documentation. http://aws.amazon.com/documentation/iam/.

[11] The Cloud Market. EC2 Statistics. http://thecloudmarket.com/stats.

[12] Amazon Elastic Compute Cloud User Guide, 2011. http://awsdocs.s3.amazonaws.com/EC2/latest/ec2-ug.pdf.

[13] An Update on Android Market Security, March 3 2011. http://googlemobile.blogspot.com/2011/03/update-on-android-market-security.html.

[14] AWS Developer Forums: Email from Amazon EC2 about my AMI being compromised, April 8 2011. https://forums.aws.amazon.com/thread.jspa?messageID=235613.

[15] Compromised EC2 image includes root access SSH key, 2011. http://pastebin.com/q1VH4rmF.

[16] A. Abdelhalim and I. Traore. The impact of google hacking on identity and application fraud. In *Pacific Rim Conference on Communications, Computers and Signal Processing*, pages 240 –244. IEEE, 2007.

[17] N. Arvantis, M. Slaviero, and H. Meer. Clobbering the Cloud!, 2009. http://www.sensepost.com/labs/conferences/clobbering_the_cloud.

[18] Amazon Web Services Documentation. http://aws.amazon.com/documentation.

[19] Amazon Web Services: Overview of Security Processes, August 2010. http://awsmedia.s3.amazonaws.com/pdf/AWS_Security_Whitepaper.pdf.

[20] S. Bleikertz, M. Schunter, C. W. Probst, D. Pendarakis, and K. Eriksson. Security audits of multi-tier virtual infrastructures in public infrastructure clouds. In *ACM Cloud Computing Security Workshop (CCSW'10)*, pages 93–102. ACM, 2010.

[21] Y. Chen, V. Paxson, and R. H. Katz. What's new about cloud computing security? Technical Report UCB/EECS-2010-5, Jan 2010.

[22] Cloud Security Alliance (CSA). Security guidance for critical areas of focus in cloud computing v2.1, December 2009. https://cloudsecurityalliance.org/csaguide.pdf.

[23] Cloud Security Alliance (CSA). Top threats to cloud computing, March 2010. cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf.

[24] Ionut Constandache, Aydan Yumerefendi, and Jeff Chase. Secure control of portable images in a virtual computing utility. In *ACM workshop on Virtual machine security*, VMSec '08, pages 1–8. ACM, 2008.

[25] Danny Dolev and Andrew C. Yao. On the security of public key protocols. Technical report, 1981.

[26] European Network and Information Security Agency (ENISA). Cloud computing security risk assessment, May 2009. http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-risk-assessment/.

[27] S.L. Garfinkel. Automating disk forensic processing with SleuthKit, XML and Python. In *Workshop on Systematic Approaches to Digital Forensic Engineering*, pages 73–84. IEEE, 2009.

[28] S.L. Garfinkel and A. Shelat. Remembrance of data passed: a study of disk sanitization practices. *IEEE Security and Privacy*, 1(1):17 – 27, January 2003.

[29] T. Garfinkel and M. Rosenblum. When virtual is harder than real: security challenges in virtual machine based computing environments. In *Workshop on Hot Topics in Operating Systems (HotOS'05)*, page 20. USENIX, 2005.

[30] B. Hatch. SSH Host Key Protection, Oct 2004. http://www.symantec.com/connect/articles/ssh-host-key-protection.

[31] C. Heath. *Symbian OS Platform Security*. John Wiley & Sons, 2006.

[32] IBM Cooperation. IBM SmartCloud. http://www-935.ibm.com/services/us/igs/cloud-development/#tab:details-security.

[33] D. Molnar and S. Schechter. Self hosting vs. cloud hosting: Accounting for the security impact of hosting in the cloud. In *Workshop on the Economics of Information Security (WEIS 2010)*, June 2010.

[34] J. Murty. *Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*. O'Reilly Media, 2008.

[35] NIST. Guidelines on security and privacy in public cloud computing. 2011. Special Publication 800-144.

[36] NIST. The NIST Definition of Cloud Computing (Draft). 2011. Special Publication 800-145 (Draft).

[37] O. Pirttikoski and Y. Kortesniemi. Local Key and Certificate Storage in JDK 1.3. *Proceedings of the NordSec2000, Reykjavk, Iceland*, 2000.

[38] 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 *SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE'08)*, pages 111–120. ACM, 2008.

[39] 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 *ACM conference on Computer and communications security (CCS'09)*, pages 199–212. ACM, 2009.

[40] T. Ristenpart and S. Yilek. Randomness goes bad:

Virtual machine reset vulnerabilities and hedging deployed cryptography. In *Network and Distributed Security Symposium (NDSS'10)*. ACM, 2010.

[41] M. Satyanarayanan, W. Richter, G. Ammons, J. Harkes, and A. Goode. The case for content search of vm clouds. In *Computer Software and Applications Conference Workshops (COMPSACW'10)*, pages 382 –387. IEEE, 2010.

[42] Amazon SimpleDB. `http://aws.amazon.com/simpledb`.

[43] S. Swidler. How to Keep Your AWS Credentials on an EC2 Instance Securely, August 2009. `http://shlomoswidler.com/2009/08/how-to-keep-your-aws-credentials-on-ec2.html`.

[44] M. Szeredi. SSH filesystem. `http://fuse.sourceforge.net/sshfs.html`.

[45] H. Takabi, J. B.D. Joshi, and G.-J. Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security and Privacy*, 8:24–31, 2010.

[46] E.I. Tatlı. Google Reveals Cryptographic Secrets. In *Kryptowochenende 2006 – Workshop über Kryptographie. Universität Mannheim*, page 33, 2006.

[47] J. Varia. Architecting for the cloud: Best practices, January 2011. `http://media.amazonwebservices.com/AWS_Cloud_Best_Practices.pdf`.

[48] J. Wei, X. Zhang, G. Ammons, V. Bala, and P. Ning. Managing security of virtual machine images in a cloud environment. In *ACM Cloud Computing Security Workshop (CCSW'09)*, pages 91–96. ACM, 2009.

[49] Dan Wendlandt, David G. Andersen, and Adrian Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX 2008 Annual Technical Conference*, pages 321–334, Berkeley, CA, USA, 2008. USENIX Association.

[50] J. Yan, A. Blackwell, R. Anderson, and A. Grant. Password memorability and security: empirical results. *IEEE Security and Privacy*, 2(5):25 –31, 2004.

[51] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.

[52] 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 *Annual Computer Security Applications Conference (ACSAC'10)*, pages 377–386. ACM, 2010.

APPENDIX

A. VALUE OF CAPTURED AWS API KEYS

We roughly estimate the potential financial damage that can be caused with a captured AWS API key. As the pricing is highly dynamic and depends on various factors we give a general overview only which may change over time.

It is not possible to set an upper limit on the money AWS will charge for service usage. There exist only limitations on the amount of resources a user can request at a time. These limits can be increased by contacting Amazon. A standard account is allowed to start 20 on-demand EC2 instances and to request 100 instances on the spot market. On the spot market, unused resources of EC2 are sold for a varying price, depending on the current demand. Given these limitations, an attacker can request 20 on-demand instances of the most expensive "Quadruple Extra Large" type ($2.10/h) and 100 "Quadruple Extra Large" type instances on the spot market (we assume $0.84/h). This leads to costs of $126/h or $3024/day. Note that this calculation is pessimistic as it does not include additional costs for network bandwidth, storage or I/O access. All in all, capturing an AWS API key allows an attacker to misuse simultaneously around 8000 GB memory and 3000 EC2 Compute Units (ECU), where one ECU is equivalent to the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.

B. SSH AUTHENTICATION

As described in §3.2, SSH [30] is the primary method for remote administration in EC2. The SSH protocol supports user authentication based on passwords or asymmetric key pairs and has built in protection against man-in-the-middle attacks as described next.

In the cloud environment mainly the asymmetric key based method is used. It is supported by the EC2 infrastructure as well as most public AMIs, and is considered to be more secure than password based authentication. As mentioned in §3.2, the AMI user's public key pk_u is imported by a newly created instance (called host in SSH context) from EC2 and the user then authenticates herself by proving possession of the associated secret key sk_u. In addition to that, SSH also supports a check of the host's identity to prevent an attacker from impersonating a remote host or launching a man-in-the-middle attack. This is realized with an unique and unforgeable asymmetric key pair $(\mathsf{sk}_h, \mathsf{pk}_h)$, called host key which is stored on the remote host. When a user connects to a host he has to verify that the fingerprint, i.e., a hash, $\mathsf{f}(\mathsf{pk}_h)$ of the host key belongs to the machine he wants to connect to (cf. Fig. 7). As no Certificate Authority is involved, it is the responsibility of the user to ensure the authenticity of the fingerprint when connecting to a new or unknown host, e.g., over an outbound channel. After the key has been accepted, the SSH client application caches the mapping between the key and the host's IP address or hostname.

```
john:~$ ssh ec2-user@47.127.0.233
The authenticity of host '47.127.0.233 (47.127.0.233)'
     can't be established.
RSA key fingerprint is
     f3:5a:f4:a2:f3:d1:e5:52:2c:e3:87:ff:07:13:01:11.
Are you sure you want to continue connecting (yes/no)?
```

Figure 7: SSH First Connection