# How Private is Your Private Cloud? Security Analysis of Cloud Control Interfaces

Dennis Felsch
Horst Görtz Institute for IT-Security
Bochum, Germany
dennis.felsch@rub.de

Mario Heiderich
Horst Görtz Institute for IT-Security
Bochum, Germany
mario.heiderich@rub.de

Frederic Schulz
Horst Görtz Institute for IT-Security
Bochum, Germany
frederic.schulz@rub.de

Jörg Schwenk
Horst Görtz Institute for IT-Security
Bochum, Germany
joerg.schwenk@rub.de

## ABSTRACT

The security gateway between an attacker and a user's private data is the Cloud Control Interface (CCI): If an attacker manages to get access to this interface, he controls the data. Several high-level data breaches originate here, the latest being the business failure of the British company *Code Spaces*.

In such situations, using a private cloud is often claimed to be more secure than using a public cloud. In this paper, we show that this security assumption may not be justified: We attack private clouds through their rich, HTML5-based control interfaces, using well-known attacks on web interfaces (XSS, CSRF, and Clickjacking) combined with novel exploitation techniques for Infrastructure as a Service clouds.

We analyzed four open-source projects for private IaaS cloud deployment (Eucalyptus, OpenNebula, OpenStack, and openQRM) in default configuration. We were able to compromise the security of three cloud installations (Eucalyptus, OpenNebula, and openQRM) One of our attacks (OpenNebula) allowed us to gain root access to VMs even if full perimeter security is enabled, i.e. if the cloud control interface is only reachable from a certain segment of the company's network, and if all network traffic is filtered through a firewall.

We informed all projects about the attack vectors and proposed mitigations. As a general recommendation, we propose to make web management interfaces for private clouds inaccessible from the Internet, and to include this technical requirement in the definition of a private cloud.

*CCSW'15,* October 16, 2015, Denver, Colorado, USA.
Copyright is held by the owner/author(s). Publication rights licensed to ACM.
ACM 978-1-4503-3825-7/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2808425.2808432.

## Keywords

Cloud Security; Cloud Interface; Infrastructure as a Service; XSS; CSRF

## 1. INTRODUCTION

According to NIST Special Publication 800-145 [24], cloud computing can be categorized into three service models: Software as a Service (SaaS), Platform as a Service (PaaS), and Infrastructure as a Service (IaaS). In this paper, we concentrate on IaaS cloud systems. An IaaS cloud provides the consumer with full control over the (virtualized) infrastructure to use. A consumer has the choice between different (virtual) hardware configurations and operating systems (bundled into a Virtual Machine, VM), and may select network configurations and storage systems.

### *The Cloud Control Interface (CCI).*

Even if all other components of a cloud system (VMs, virtual networks, persistent storage) are protected by perimeter security systems (network separation, firewalls, IDS), the Cloud Control Interface (CCI, interface 4 in Figure 1) necessarily must be exposed to the outside world, since it must allow for "on-demand self-service" [24]. Therefore, CCIs are implemented as public web APIs, mostly as a web application, but also in form of REST or SOAP-based APIs.

This gives an attacker the same type of access as the legitimate user, thus a user's data is only secure if the CCI is secure. Just to mention one serious attack from 2014, the British company *Code Spaces* went out-of-business because an attacker seized control over their Amazon AWS CCI and, after the company had been blackmailed and refused to pay, deleted all company data including backups in the Cloud.[1]

Because of this impact of a possible security breach of a CCI, we believe that CCIs should employ any know security measure to protect themselves. Unfortunately, this is not the case: Especially web application CCIs suffer from the complexity of the interface, which not only supports the growing range of HTML5 standards (Scriptless Attacks), but also implements legacy features going back to DOM Level 0 (DOM

[1]`http://www.infoworld.com/article/2608076/data-center/murder-in-the-amazon-cloud.html`

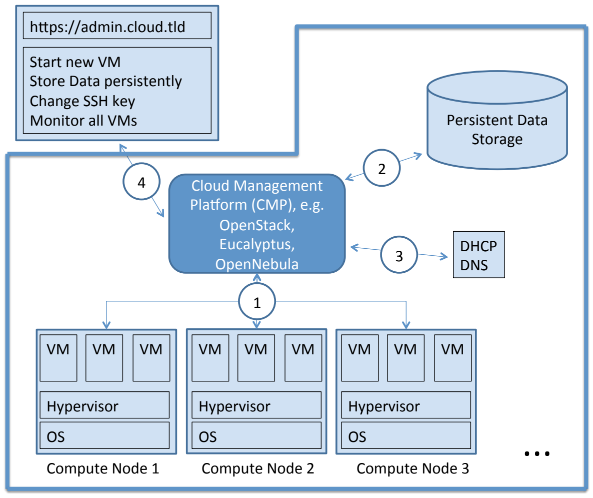


**Figure 1: IaaS cloud building blocks**

Clobbering), undocumented browser behaviour (mXSS) and large 3rd party libraries (e.g. jQuery).

## *The Promises of Private Clouds.*

Private Clouds are often advertised as being more secure than their Public Cloud counterparts, and thus as a solution for securely protecting company data. [24] describes these different deployment models, which are mostly applied to the IaaS service model: *Private Clouds*, provisioned for the use of a single entity, *Public Clouds* which are accessible for open use by the general public, and two intermediate models, *Community* and *Hybrid Clouds*. In this paper, we concentrate on private cloud deployments. According to [24] this is defined as follows: *"Private cloud. The cloud infrastructure is provisioned for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises."*

ISO/IEC 17788 [15] gives more or less the same description of this deployment model: *"A private cloud may be owned, managed, and operated by the organization itself or a third party [...]. Private clouds seek to set a narrowly controlled boundary around the private cloud based on limiting the customers to a single organization"* [15].

Both NIST and ISO definitions are organizational in their nature and do not define, how the deployment models differ from a technical point of view. For example, the boundary mentioned in the ISO definition could be realized using firewalls and network separation techniques. However, if a private cloud is operated by a third party, as mentioned in the standard, the security gain of a firewall protection is questionable.

In this paper, we investigate if private clouds really give better security, by investigating the *technical* features of four prominent cloud projects (Eucalyptus, OpenNebula, OpenStack) which are also deployed as private clouds, and a small project dedicated to the development of a private cloud (openQRM). In all cases, we investigate the security of the CCI.

For the CCI, we considered two scenarios: In Scenario 1, the CCI is fully exposed to the Internet, whereas in Scenario 2, a firewall is used to block direct access to the CCI from the Internet. The latter may be seen as a straightforward "provisioning" of a cloud implementation for private use.

## *Attacking Private Clouds.*

Since private clouds are provisioned for use by a single entity only, attacks on the virtualization layer may only be performed by insiders, i. e. employees of the entity using the cloud. We therefore concentrate on attacks on the CCI itself, which can be accessed through a variety of APIs. In this paper, we concentrate on the Web API, i. e. the API accessible through a standard web browser.

On a high level, the success of some of our attacks even in the presence of perimeter security mechanism like Firewalls can be explained as follows: We use a web browser as our "malicious insider" within the company network. The browser receives instructions through port 80 (HTTP), which is open in any firewall, and may transfer stolen data through the same port.

The transformation of a "trusted browser" into a "malicious insider" is done through standard web attacks: Either we directly control browser actions (CSRF) invisible to the user, or we execute malicious script code in the web interface (XSS).

## *Attacks on Web Interfaces.*

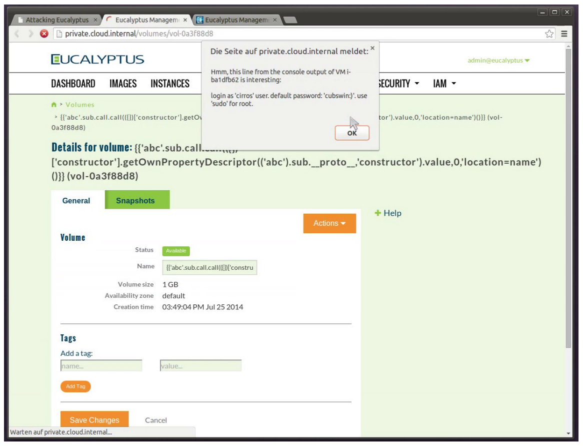
Web browsers are often used as the user's interface to a cloud: They are available for any operating system, the markup (HTML5, XHTML), data (XML, JSON), scripting (JavaScript, XSLT) and style (CSS) languages are platform independent, their communication protocols (HTTP, HTTPS, WebSockets) should pass through any firewall, and they are free-of-charge.

However, this wealth of features (which grows steadily) comes at the price of enhanced vulnerabilities: Scripting functionality may be misused through inserting malicious JavaScript into a web page (XSS, Scriptless Attacks), automatic rendering features may be misused to remotely control the browser in a malicious way (CSRF), style languages may be used to mask attacks (UI Redressing), and simple firewalls do not protect against these attacks. All these attacks are described in Section 4.2.

By using the same browser both for browsing the Web and for accessing security critical applications, attack vectors may be carried from the web through the firewall to the application, where they are executed.

## *Results.*

Of the four systems investigated, we were able to break security of the web interfaces of Eucalyptus, OpenNebula, and openQRM. In each of these systems, we found at least one attack that also worked if access to the web interface was restricted by a firewall blocking all direct access. All three major systems were well designed, but the inclusion of a direct data channel through the web browser to each VM proved to be problematic. For openQRM, nearly no security measures were in place. This seems to be an effect of most discussions on cloud security concentrating on the VM level, which does not raise awareness for the criticality of the CCI. We reported all findings to the corresponding security teams.



private.cloud.internal/volumes/vol-0a3f88d8

Die Seite auf private.cloud.internal meldet:

Hmm, this line from the console output of VM i-ba1dfb62 is interesting:

login as 'cirros' user. default password: 'cubswin:)'. use 'sudo' for root.

{{'abc'.sub.call.call(({})['constructor'].getO tor').value,0,'location=name')()}} (vol-0a3f88d8)

Details for volume: {{'abc'.sub.call.
['constructor'].getOwnPropertyDescriptor(('abc').sub.__proto__,'constructor').value,0,'location=name')
()}} (vol-0a3f88d8)

Volume

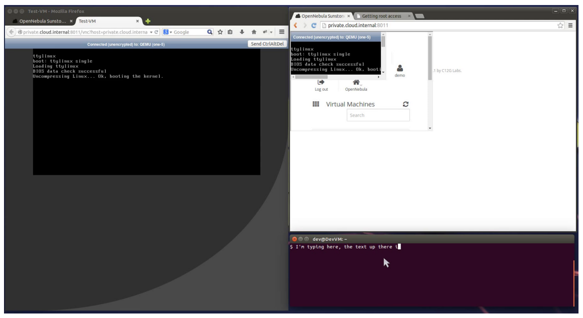
Status: Available
Name: {{'abc'.sub.call.call(({})['constru
Volume size: 1 GB
Availability zone: default
Creation time: 03:49:04 PM Jul 25 2014

Tags

Add a tag: name... value...

Warten auf private.cloud.internal...

**Figure 2: Successful XSS attack on Eucalyptus**



private.cloud.internal:8011/vnc?host=private.cloud.interna

Connected (unencrypted) to: QEMU (one-5)

```
ttylinux
boot: ttylinux single
Loading ttylinux
BIOS data check successful
Uncompressing Linux... Ok, booting the kernel.
```

private.cloud.internal:8011

```
$ I'm typing here, the text up there i
```

**Figure 3: Successful XSS attack on OpenNebula, deployed with CCI hidden behind a firewall**

**Eucalyptus.** Architecturally, Eucalyptus uses modern JavaScript Model View Controller libraries to render (user contributed) content, and is therefore well protected against direct XSS attacks. However, it also uses the library *AngularJS*, and since inputs to this library (in the form of *AngularJS expressions*) is not completely sanitized, it is possible to execute arbitrary JavaScript in the context of the Web CCI, both in Versions 4.0.0 and 4.0.1 (cf. Figure 2). By having the ability to execute arbitrary scripts, we were able to start and stop VMs. We were also able to read all content displayed in the Web CCI, e. g. passwords.

**OpenNebula.** For OpenNebula, we found several severe attacks. By injecting an arbitrary opening XML tag into the name of a VM, we can make this VM inaccessible from the web CCI. In addition to being able to execute arbitrary script code in the Web CCI (which allowed us to start and stop VMs, and to read critical data from the Web CCI, like in the Eucalyptus case) we were able to take complete control over *existing* VMs, and thus to read out the data stored and processed in this VM (cf. Figure 3).

**openQRM.** In openQRM, no working security measures for the Web CCI were in place. No CSRF protection was used (thus attacks through a firewall become possible), and a self-written XSS filter was easy to bypass. XSS vulnerabilities in the user Web CCI could easily be used against the admin Web CCI, since both interface use the same domain (and thus the Same Origin Policy does not prevent access). In addition, multiple SQL injection vulnerabilities existed.

*Contributions.*
In a field study, we examined the security of Web CCIs against classic and recent attacks discussed in the web security community. Our findings are mixed, with one product offering good protection (OpenStack), one product offering medium security (Eucalyptus), and two products which fail to protect VMs and data (OpenNebula, openQRM). This shows that the significance of CCI security for overall Cloud security is not completely understood yet. Our contributions are:

- We present novel attacks on how to access a VM directly through the victim's web browser. These attacks combine standard web attacks with novel methods on how to execute script code and to access data stored in VMs.
- We were able to break security of three out of four private cloud frameworks. For OpenNebula, we were able to access data on running VMs, resulting in the most severe of all attack scenarios. For openQRM, vulnerabilities could be found on all levels.
- We discuss a technical definition of private clouds, with respect to the Cloud Control Interface. In this definition, the CCI is not exposed to the Internet.

## 2. RELATED WORK

Most previous security analyzes of IaaS clouds concentrate on detection and exploitation of *VM co-location* [27, 32, 13, 31, 35]. Co-located VMs are assigned to be hosted on the same compute node within a cloud. The exploitation of co-location focuses on the deployment scheduler of a cloud management platform and the hypervisor layer used for virtualization.

A security survey that involves all layers of cloud computing including web attacks is given by Modi et al. [25]. Sempolinski and Thain [29] compare the structural concepts of three cloud management platforms, namely Eucalyptus, OpenNebula, and Nimbus. Two of these platforms coincide with those we are analyzing; however, they do not focus on security questions. Security analyzes with respect to operating system security of OpenStack and Eucalyptus are given in [28] and [17].

In 2009, Gruschka and Lo Iacono investigated the security of the SOAP based cloud control interface of Amazon's *Elastic Compute Cloud (EC2)* [16]. This work was extended by Somorovsky et al. [30], in which the interfaces of Amazon and Eucalyptus interfaces were compromised using both Cross-Site-Scripting and XML based attacks.

An overview on Cross-Site-Scripting (XSS) can be found in [21]. The basic idea and flavors of XSS are also presented in Section 4.2. Recently, novel XSS classes have been described: *Scriptless Attacks* [18], where scripting functionality of novel HTML5 elements is misused, and *mXSS* [19], where mutation features of the browser are used to change harmless markup into a malicious script. Other attacks on web applications like Cross-Site Request Forgery and UI-Redressing are depicted in [34] and [26].

## 3. FORMAL MODEL

In this section, we give a formal model of our research object, and the attacker model.

## 3.1 Modelling IaaS

For modelling IaaS, we follow [29]. IaaS systems typically consist of the following components (cf. Figure 1).

### *Virtualization.*

IaaS clouds[2] are realized using virtualization technology. This way, computing resources can be assigned to consumers dynamically. However, at some point in the architecture, real hardware has to supply the virtual resources. The task of creating such a relation is done by a hypervisor, a software that allocates computing resources from bare metal and assigns it to virtual resources. A Virtual Machine (VM) is an aggregation of virtual computing resources, virtual memory and storage, virtual networking, etc. For the consumer, a virtual machine is indistinguishable from a real machine in its behavior.

### *Cloud Management Platform.*

A Cloud Management Platform (CMP) is a software product that orchestrates a cloud infrastructure. An overview of a CMP's responsibilities is given in Figure 1.

1. The primary task of a CMP is to control its compute nodes and the hypervisors on these nodes. To determine on which node a new VM is to be instantiated, a CMP contains a scheduler that evaluates some policy for this decision. CMPs often include a component that can dynamically resize the compute resources of VMs or provide failover resilience for VMs.
2. Usually, virtual hard disks of VMs are not stored on compute nodes since this would limit the flexibility of a cloud installation. Instead, these images and volumes are stored on a large external storage. From there a CMP makes them available to the corresponding hypervisors.
3. Once a VM is instantiated from a template, it lacks necessary information like which IP address to use or who has access rights to it. This information is provided through a process called *contextualization*, which is carried out by the CMP.
4. All CMPs contain some interface where a Cloud Control Interface can attach.

### *Cloud Control Interface (CCI).*

A CCI, as outlined before, is a self-service interface that allows triggering and controlling actions like starting or stopping VMs. These interfaces may be based on any standardized technology (XML/SOAP, REST/HTTP, JSON, AJAX, etc.). In this paper, we concentrate on one technology that is offered in any cloud: We examine CCIs that are implemented using a web browser as client software.

## 3.2 Modelling private clouds

Much has been written about private clouds, but a complete formal definition is still missing. Available (partial) definitions only concentrate on VM co-location. In this section, we try to provide this definition for IaaS, and discuss implications on different types of VMs.

[2]Since we only deal with IaaS, we will omit the cloud type in the following

### *Private Clouds vs. Public Clouds.*

Private and public clouds share the same technology; there is no fundamental difference in the techniques employed. All Cloud software stacks analyzed in this paper can be used in both a public and a private cloud setup. Differences between private and public clouds must therefore be defined through properties of the setup itself. The first main difference, which is mentioned in all previous definitions, is VM co-location:

DEFINITION 1 (VM CO-LOCATION). *In a* public cloud*, Virtual Machines (VMs) of different tenants may be running on the same physical host, whereas in a* private cloud*, all VMs are be controlled by one entity.*

Closer investigation of Definition 1 reveals that here only the *goal* of a private cloud setup is defined: In a private cloud, it *should* be guaranteed that VMs are indeed controlled by a single entity only.

The second main difference is in the reachability of the Cloud Control Interface (CCI) (cf. Figure 4): In a private cloud, access to the CCI may be limited through network perimeter controls (e. g. firewalls; NAT, VLAN), whereas in a public cloud it may not. Please note that perimeter controls may be applied to all other components of the Cloud, regardless if it is public or private.

DEFINITION 2 (REACHABILITY OF THE CCI). *In a* public cloud*, the* Cloud Control Interface (CCI) *is reachable from the Internet. In a* private cloud*, the CCI may only be reachable from a well-defined, closed subnet, e.g. a company's Intranet.*

Remarks:

1. Please note that also the second definition does not describe a technical difference between private and public clouds, only the audience that may perform attacks on the CCI is different: For public clouds, all Internet users may start web attacks, whereas in a private cloud, this is limited to company employees.
2. Please note also that the reachability constraints for a private cloud CCI are only a possible security measure: They are not part of the Cloud Management Platforms we analyzed, but may differ for each deployment.

### *Classification of (Private) Cloud VMs.*

Virtual Machines (VMs) may have different network connectivity (cf. Table 1): (1) no network connectivity, (2) connectivity to a closed network, or (3) connectivity to the Internet. In cases (2) and (3), they may either (a) be reachable from the network (e.g. through HTTP of SSH), or (b) not reachable.

## 3.3 Attacker Model

All attacks described in this paper are in the *web attacker model*, i.e. they are practical attacks. The only prerequisite of an attack in the web attacker model is that the victim must visit a (malicious) webpage of the attacker. In case of a private cloud, the URL of the CCI must also be known to the attacker (for public clouds, it is known to the public). The URL can e. g. be retrieved by reading company documentation, emails, or public discussion forums. The attacker does not need to be able to resolve the CCI's domain

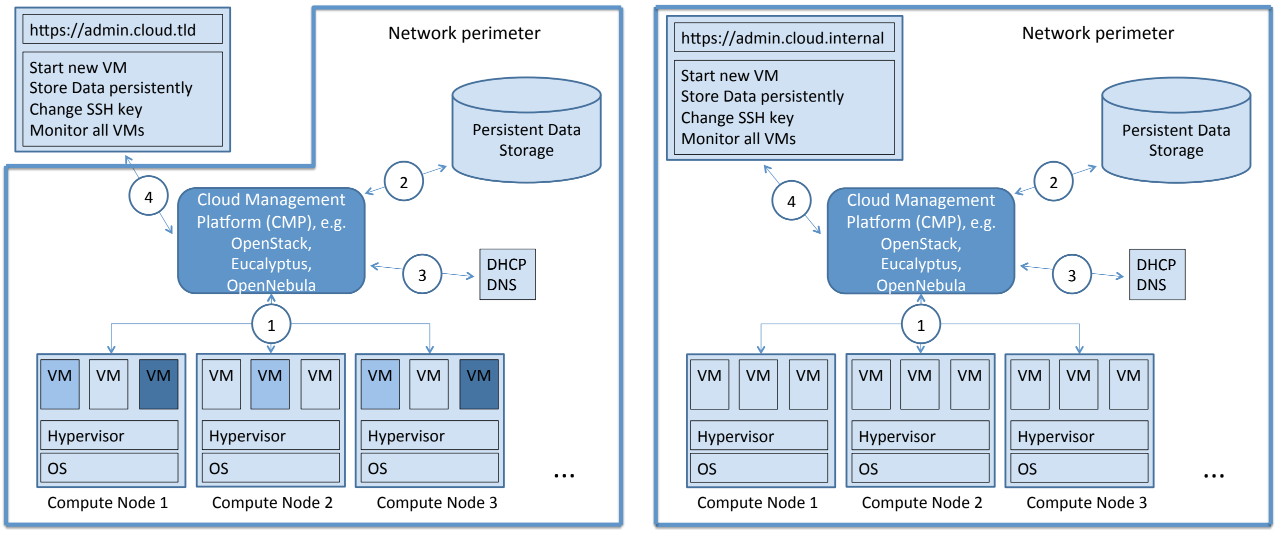


**Figure 4: In a public cloud (left), VMs controlled by different tenants may be co-located on one physical host, and the CCI is exposed to the Internet. In a private cloud (right), all VMs are controlled by employees of one entity, and the CCI may be protected by network perimeters**

| | public cloud | private cloud |
|---|---|---|
| CCI | Y | N |
| Persistent Storage | Y (partly) | N |
| Virtual Network | N | N |
| VM Type (1) | N | N |
| VM Type (2) | N | N |
| VM Type (3) | (a) Y, (b) N | (a) Y, (b) N |

**Table 1: Accessibility of different Cloud components from the Internet**

name; the only requirement is that the browser of the victim is able to resolve it. No further control over the network is assumed; especially firewalls, network encryption, and other standard protection mechanisms may be deployed.

If a CCI exposes vulnerabilities, a web attacker may be able to execute script code in the context of the CCI (XSS), may trigger actions at the CCI (CSRF), or may invisibly embed the CCI into the malicious webpage (UI-Redressing).

### *Definition of Successful Attacks.*

Since we are dealing with IaaS clouds, our main goal is to get access to some VM in the cloud. More specifically, this goal can be achieved with three levels of severity:

1. **DoS-Level**: Use or block resources of the cloud (e. g. block a user from accessing VMs).
2. **Control-Level**: Alter the state of the cloud (e. g. start or stop a used VM, modify accounts).
3. **Compromise-Level**: Take over a running VM, together with all data.

It is obvious that an attack in a higher level also allows attacks of the previous levels. Compromise-level attacks thus completely break cloud security, because an attacker can access both cloud resources and cloud data. Note that a Control-level attack does not necessarily allow an attacker to compromise a VM. Even if the attacker is able to inject e. g. a new SSH key into the victim's account, the VM to attack may have no network access (cf. Table 1) so that the attacker cannot make use of the key.

## 4. TECHNICAL BACKGROUND

In this section, we give some technical background on our research targets and on the different attack classes. Furthermore, we discuss valid an invalid attack goals in a private cloud scenario.

### 4.1 Private Cloud CMPs

The field of CMPs is dominated by three open source projects, which sometimes also have commercial spin-offs. We furthermore analyzed a small project that is also used to deploy private clouds. We decided to test the default configurations of the open source versions, because they were available for testing, and the results were comparable.

#### *Eucalyptus.*

Eucalyptus [5] emerged from a research project at the University of Santa Barbara and was commercialized in 2009. Hewlett-Packard acquired Eucalyptus in September 2014. The Eucalyptus Cloud is being used by a wide range of commercial and academic clients, including defense contractors and institutions such as Raytheon and the US Department of Defense. Only the recent versions of Eucalyptus are fully open source (GPLv3); older versions used proprietary components such as support for certain hypervisors or network storage systems. Eucalyptus mimics AWS, therefore Eucalyptus can manage either Amazon or Eucalyptus VMs. VM instances can also be moved between the Amazon Elastic Compute Cloud and a Eucalyptus private cloud.

#### *OpenNebula.*

OpenNebula [9] also started as a research project in 2005. It is used by information technology companies like IBM, Dell and Akamai as well as academic institutions and the European Space Administrations (ESA). By relying on standard Linux tools as far as possible, OpenNebula reaches a high level of customizability and flexibility in hypervisors, storage systems, and network infrastructures. OpenNebula offers an XML-RPC interface to control a cloud. Other interfaces (e.g. Ruby and Java interfaces as well as an AWS-compatible interface) are wrapped around this interface. Sunstone, OpenNebula's web interface, is also built around the XML-RPC interface. OpenNebula is distributed using the Apache-2 license.

### *OpenStack.*

The OpenStack project [12] seeks to offer an alternative to *Amazon Web Services (AWS)* and sees itself in direct competition. OpenStack tries to provide as much compatibility to Amazon's external interfaces as possible. It uses a modular architecture approach to support scalability as well as a broad variety of hypervisors, storage systems, network infrastructures, etc. The module that is primarily viewed at in this research is Horizon, OpenStack's web interface component. OpenStack is available under Apache-2 license terms.

### *openQRM.*

In 2003, openQRM [11] started as a commercial datacenter management solution, originally developed by Qlusters, Inc. settled in Palo Alto, California. Today, openQRM is an IaaS system distributed by openQRM Enterprise GmbH. One can use a free community edition or an enterprise edition with professional support and additional features. openQRM provides two separate web interfaces: An administration interface for privileged users and a cloud user interface for regular users. Furthermore, other interfaces like SOAP exist. The community edition of openQRM is available under GPL license terms.

## 4.2 Attacks on Web Interfaces

Three classes of attacks on web applications will be described in this section: code injection (XSS), remote control (CSRF) and layout (UI-Redressing).

### *Cross-Site Scripting (XSS).*

To perform a *Cross-Site Scripting (XSS)* attack, an adversary tries to inject his own malicious code into a webpage delivered by the target web application. The latter is necessary because the Same Origin Policy (SOP) restricts script access to web content having the same origin. Thus only if the code is either embedded in (inline scripts) or loaded into the webpage, it will have (read and write) access to the contents of this webpage (e. g. session cookies, form fields, etc.).

XSS comes in three flavors: (1) In a *reflected XSS* attack, the adversary first sends the malicious code to the target server (e. g. as a query string in a GET request, or in the body of a POST request) where it is embedded in a dynamically generated webpage. As soon as this webpage is rendered in the browser, the code is executed. (2) In a *stored XSS* attack, the adversary stores the malicious code in some subpage of the web application (e. g. discussion forum), and when this subpage is requested and rendered in the browser, the code is executed. (3) *DOMXSS* [23] relies on the fact that the content of a webpage may be changed dynamically by the browser itself, who may copy information from e. g. the document URL into the document itself. In contrast to the first two types of XSS, DOMXSS may not be detectable by the server, since the malicious script may never pass through the server.

Today, the most important countermeasure against XSS is server-side filtering, although some powerful client-side filters (e. g. NoScript for Mozilla Firefox [7]) exist.

### *Cross-Site Request Forgery (CSRF).*

Some actions of a web browser can be triggered without user interaction; e. g. if the browser parses an `<img src="http://somedomain.org/pic.jpg" />` tag, it automatically tries to load `pic.jpg` from `somedomain.org` by sending an HTTP GET request.

If `<img src="http://pizza.org/order.php?type=13&deliver=attacker"/>` is parsed, an HTTP GET request will be sent to `pizza.org` ordering pizza number 13, which will be delivered to the attacker's address. If we assume that the victim is logged in to `pizza.org`, then simply viewing the webpage of the attacker will send the pizza to the attacker, and the bill to the victim.

The possible impact of CSRF attacks has been shown in [34], where an attack was shown to empty online banking accounts with a complex CSRF attack. CSRF attacks can be mitigated by deploying *anti-CSRF-tokens*, which are in essence random values embedded into form fields. Only HTTP requests containing the correct nonce value will be executed by the server.

### *Clickjacking and UI-Redressing.*

Modern web browsers offer a large variety of design features, including transparency options for complete HTML frames. A basic example for a UI-Redressing attack is to load the target webpage as an invisible iFrame, and to place an innocent-looking visible page below it. The visible page may trick the user into performing mouse clicks, drag&drop operation, or into entering text, but all these actions will be caught by the invisible frame.

Through an `X-Frame-Options` HTTP header, a webpage can communicate to the browser whether it may be loaded as an iFrame into another page, and under which conditions.

## 4.3 Attacks on Clouds

We do not count each successful attack on the CCI web interface as a successful attack on the private cloud. Instead, we have to single out those attacks that can be performed without direct access to the CCI.

### *Valid Web Attacks on Public Clouds.*

In web application scenarios, the compromise of a session cookie often equals a compromise of the web application itself: It grants the attacker full access to the victim's account. This also holds for the CCI of public clouds. For private clouds, even if we are able to steal the cookie, we may not be able to use it, because we have no direct access to the CCI.

Likewise, for a stored XSS attack, the adversary needs access to some part of the CCI. In a public cloud, this may e. g. be configuration data for an account owned by the adversary, or a message posted in a discussion forum. This attack is applicable to private clouds if we do consider insider attacks, or if the victim creates the XSS as result of Clickjacking or if the cloud imports data from somewhere, e. g. emails, VM descriptions, or images.

### *Valid Web Attacks on Private Clouds.*

Once a malicious webpage is loaded into the browser of the victim, markup contained therein may trigger actions from the victim's browser to the CCI via CSRF. These actions will typically be sent as GET or POST requests inside the company's network, and do not have to cross perimeter boundaries anymore. As a precondition, the attacker needs to know the domain name of the CCI to be able to call it through an URL. Alternatively, scanning the private IP address space where the CCI is located may be feasible.

A malicious script can also send HTTP requests to arbitrary URLs to establish a communication channel through the network perimeter. The attacker may then send commands to the malicious script through HTTP responses. One example for such a script is BeEF [2]. As a precondition, the attacker needs a reflected or DOM XSS vulnerability to install such a script.

Furthermore, once a malicious webpage is loaded into the victim's browser, it may load the CCI as an invisible frame, and may redirect user actions to the CCI. We have the same preconditions as for CSRF plus a permissive (or no) X-Frame-Options-Header.

If the CCI allows setting specific data that is forwarded to a VM, it could be possible that an attacker manipulates a VM so that it leaks information to the attacker: The attacker could reconfigure the VM so that the attacker is able to connect to the VM if it is accessible from the Internet.

### *Points of Attack in Cloud Setups.*

The number of valid points of attack on the CCI depends on the features of the CCI: Whatever is done with the help of the browser may be subject to a web attack.

If we are able to steal or overwrite the victim user's passwords, we cannot directly access the account, but we can use this additional information in one of the previous attacks. Especially, we can perform these attacks even if the victim does not have an active session, i.e. is not currently logged in. If we overwrite a password, a user may however detect this.

If we can steal a private SSH key or overwrite/generate a key pair, we may directly access VMs that are accessible from the Internet. If VMs are started, stopped, and rebooted through the CCI, or if their configuration can be changed, these actions may also be triggered by an attacker.

If the screen of a VM is displayed in a browser, we can read all data from this screen. If the screen access is realized by a VNC remote control in an HTML5 canvas, we can also inject arbitrary data into the VM. Under certain circumstances, this includes installing software.

## 5. METHODOLOGY

We developed a systematic methodology for testing the CCI of private clouds. All tests were done manually; no automated tests were involved. We installed all three cloud systems under investigation on our own hardware, and set up different accounts.

The developers of **Eucalyptus** provide a FastStart script [4] that installs the most recent stable version of Eucalyptus using KVM as hypervisor. Since we wanted to stick as close to default settings as possible, we set up a fresh CentOS 6.5 machine – the operating system required by FastStart – and used the script to install a private Eucalyptus cloud. We tested version 4.0.0 and 4.0.1 of Eucalyptus for this paper.

The **OpenNebula** project offers quick start guides and package repositories for common Linux distributions. Following a quick start guide for OpenNebula 4.6 with KVM as hypervisor [10], we installed a private OpenNebula cloud on a fresh Ubuntu 14.04 machine. The version tested for this paper is OpenNebula 4.6.1.

The **OpenStack** Foundation maintains DevStack, a shell script to build complete OpenStack environments for development and testing [3]. Since it is a tool for developers, it is designed to install the latest (unstable) development version. With a configuration file that is documented in the FAQ of DevStack, one can also install a stable version. We decided to install the stable version 2014.1 (codename Icehouse) on a fresh Ubuntu 14.04 machine.

Our installation of **openQRM** followed a Howto-Guide provided by the developers [6]. We installed it on a fresh Debian Wheezy system. The version tested for this paper is 5.1 Community.

## 5.1 Existing Countermeasures

In a second step, we checked for existing countermeasures by inspecting delivered HTTP headers and HTML code.

### *Security-Related HTTP headers.*

Some attacks on web applications like UI-Redressing can be mitigated by using HTTP headers. Therefore, we started by analyzing the presence of security-related HTTP headers. The results are shown in Table 2. Only `X-Frame-Options` is used by OpenStack and Eucalyptus; both set its value to `SAMEORIGIN`. No other security headers are used.

openQRM uses HTTP Basic authentication. The other three web interfaces rely on HTTP cookies to identify a user's session. However, only OpenNebula and OpenStack set the *HttpOnly* parameter for their cookies. Eucalyptus does not protect its session cookies from malicious JavaScript.

### *CSRF-Protection.*

Eucalyptus and OpenStack protect their web interfaces with a token against CSRF. In the case of Eucalyptus, the token is transmitted in a hidden form field and checked by the server. OpenStack in contrast relies on the *Double Submit Cookie* mechanism of the underlying Django-server. This technique sends a CSRF-token both as cookie and as hidden form field. The server then only checks whether these two tokens match, but not if the matching value equals the value sent to the client. OpenNebula and openQRM did not make use of a CSRF-protection and therefore make themselves vulnerable to CSRF-attacks.

### *XSS-Protection.*

Of the four web interfaces, only openQRM relies on HTTP-GET parameters. If no HTTP-GET parameters are used, this reduces the attack surface for reflected XSS attacks. However, we were able to find reflected XSS vulnerabilities in the web interface of openQRM and Eucalyptus 4.0.1.

We were able to identify stored XSS vulnerabilities for all four web interfaces, despite some countermeasures. E. g., the interface of Eucalyptus is built upon AngularJS [1], a web application framework that is designed to provide basic protection against XSS. However, we were able to circumvent this protection (cf. Section 6).

| HTTP Header | Eucalyptus | OpenStack | OpenNebula | openQRM |
|---|---|---|---|---|
| `Content-Security-Policy` | N | N | N | N |
| `X-Content-Type-Options` | N | N | N | N |
| `X-Frame-Options` | Y | Y | N | N |
| `X-XSS-Protection` | N | N | N | N |

**Table 2: Security-related HTTP headers set by the web interfaces of cloud management platforms**

## 5.2 Accessibility of VMs through the CCI

Since clouds manage virtual machines, we looked for features that allow reaching the VMs in the cloud. OpenNebula, OpenStack, and openQRM combine their web interfaces with noVNC [8], an HTML5-based VNC client. This way, a user can see a VMs display output and send input commands (keyboard, mouse) to that VM. noVNC is based on HTML5 WebSockets for the connection and on the `canvas` element for rendering the output. Eucalyptus does not include a full VNC client, but it allows reading the console output of a VM.

At a first glance, noVNC and console output come handy for the initial setup of a VM or for debugging. However, the direct access to the VM through the web interface carries the risk that an attacker is able to steal data. For example in case of noVNC, once access to the web interface (e. g. by XSS) is established, the attacker can use the `canvas.toDataURL()` function to get a screenshot of a VM. Furthermore, the attacker could register event handlers for keystroke events to capture the victim's input.

## 5.3 Manipulating a VM

Once the CCI of a CMP allows reaching the VMs in the cloud, we looked for ways to manipulate data on the VMs. All CMPs use some mechanism to provide their VMs with contextualization information. This information is often security relevant, for example if the contextualization information carry SSH public keys that are set as authorized keys for the root account of a VM. If an attacker is able to substitute contextualization information, access to the VM or the data on it is possible.

However, in practice contextualization information is often read only after a VM has been instantiated. Therefore, an attacker could only influence newly created VMs, not VMs in use.

Another way to manipulate VMs is the noVNC interface. If controlling this interface is feasible, the attacker can dispatch keystroke events to it that are sent to the VM as keyboard input.

## 6. ATTACKS

In this section, we present vulnerabilities we found and demonstrate exploits that attack these vulnerabilities.

## 6.1 Eucalyptus Sandbox Bypass

Eucalyptus employs most modern JavaScript Model View Controller libraries to render the content that is being sent to the user's browser. This architectural aspect guarantees for a security model that is robust against XSS attacks.

Our security tests however showed, that this concept of rendering and securing user-controlled data was flawed because of a string reliance on the security promises the AngularJS library gives. AngularJS applications often mix user-controlled content into so-called AngularJS expressions. Those expressions execute a limited and scoped subset of JavaScript and are thought to be secure from injections.

The security of AngularJS expressions is built upon the AngularJS sandbox – a combination of a string-parser and several object checks that assure that the code to be executed does not violate any constraints and is considerably save to be evaluated. Based on a contribution by J. Horn [20], a bypass for this sandbox was tested against the AngularJS version used by Eucalyptus. It was found that the version in place is indeed vulnerable and by injecting the AngularJS expression in Listing 1, a user can gain full control over the client-side code that is executed.

```
{{'abc'.sub.call.call(({})['constructor'].
    getOwnPropertyDescriptor(('abc').sub.
    __proto__,'constructor').value,0,'
    location=name')()}}
```

**Listing 1: AngularJS expression used to execute remote code in Eucalyptus**

The expression essentially bypasses two different checks installed by the AngularJS sandbox and manages to get access to the JavaScript `Function` constructor – a function that in return accepts arbitrary strings to be executed in the global scope of the loaded document. This is being done by taking an arbitrary string and accessing one of its methods, accessing the `call()` method and, given the used parameters, returning the mentioned `Function` constructor. By not accessing the constructor, the sandbox is being bypassed, as it does not check return values and only direct access. This returned object is then again called indirectly to avoid the sandbox checks another time. The length restrictions the Eucalyptus platform imposes can be bypassed by using `window.name` to store the payload.

With Eucalyptus 4.0.1, the shipped version of AngularJS has been updated to fix this bug. As an additional protection, " in user input is replaced by ' ' which defuses AngularJS expressions. However, in a second examination we found out that this does not happen platform-wide. Some input field such as the name of a key pair remained unprotected. Using an AngularJS sandbox bypass discovered by M. Karlsson [22], we were again able to execute code in the context of the Management Console. Since the name of a key pair is also used in URLs (like `https://server/keypairs/[keyname]`) the XSS can be used in a reflected way so that a private cloud setup could be attacked with it.

### *Control-Level Attack on Eucalyptus.*

Since the above sandbox bypass for Eucalyptus 4.0.1 is a reflective XSS, Control-level attacks on a private Eucalyptus cloud are possible. Once the attacker can execute arbitrary JavaScript code, it is possible to create new VMs as well as start or stop existing ones.

Furthermore, the attacker can steal data from a VM. Eucalyptus offers a feature to read the console output of a VM for debugging. One cannot send commands to a VM this way, but if the VM outputs sensitive information (e. g. default login credentials) to the console, they can be stolen. We implemented a proof of concept exploit for Eucalyptus 4.0.0 that demonstrates the attack. A video of the attack is available[3]. Since the success of the attack depends on factors the attacker cannot influence, it is no Compromise-level attack. However, the risk that data could be stolen this way exists.

## 6.2 Stored XSS on OpenStack

Our security tests showed that OpenStack Horizon is robust and provides no attack-surface via content reflected from user-controlled GET parameters. The majority of data that a user can enter will be stored and reflected safely. This means, that a user can enter critical characters but upon displaying the result as part of the HTML of the OpenStack Management Interface, the data is encoded and escaped properly and no XSS attacks are possible. Therefore, we were not able to find an attack if OpenStack is used to set up a private cloud.

We did however identify an interface where no adequate escaping is used. With this interface injecting persistent XSS payload such as the string `<svg onload=alert(1)>` is possible. The vulnerability resides in the interface where *Host Aggregates* could be created. This interface is only available to administrators; thus, only an administrator could attack other administrators with it. For this reason, we think this XSS is not exploitable in practice.

## 6.3 OpenNebula: Denial-of-Service on VM

At its backend, OpenNebula manages VMs with XML documents. A sample for such an XML document is given in Listing 2.

```
<VM>
  <ID>0</ID>
  <NAME>My VM</NAME>
  <PERMISSIONS>...</PERMISSIONS>
  <MEMORY>512</MEMORY>
  <CPU>1</CPU>
  ...
</VM>
```

**Listing 2: Sample XML document describing a VM in OpenNebula**

OpenNebula 4.6.1 contains a bug in the sanitization of input for these XML documents: Whenever a VM's name contains an opening XML tag (but no corresponding closing one), an XML generator at the backend automatically inserts the corresponding closing tag to ensure well-formedness of the resulting document. However, the generator outputs an XML document that does not comply with the XML schema OpenNebula expects. Listing 3 shows the structure that is created after renaming the VM to `My <x> VM`. The generator closes the `<x>` tag, but not the `<NAME>` tag. At the end of the document, the generator closes all opened tags including `<NAME>`.

```
<VM>
  <ID>0</ID>
  <NAME>My <x> VM</x>
    <PERMISSIONS>...</PERMISSIONS>
    <MEMORY>512</MEMORY>
    <CPU>1</CPU>
    ...
  </NAME>
</VM>
```

**Listing 3: Sample XML document that results after the XML generator handled an unclosed element in the VM's name**

OpenNebula saves the incorrectly generated XML document in a database. The next time the OpenNebula core retrieves information about that particular VM from the database the XML parser is mixed up and runs into an error because it only expects a string as name, not an XML tree. As a result, the web-interface component of OpenNebula (called Sunstone) cannot be used to control the VM anymore. The Denial-of-Service can only be reverted from the command line interface of OpenNebula.

This bug can be triggered by a CSRF-attack, which means that it is a valid attack against a private cloud: By luring a victim onto a maliciously crafted website while logged in into Sunstone, an attacker can make all the victim's VMs uncontrollable via Sunstone. Since the attack blocks cloud resources from being used, it forms a DoS-level attack. A video of the attack is available[4].

## 6.4 OpenNebula: Compromise-Level Attack

Since CSRF protection is disabled in the version of OpenNebula we analyzed, a severe Compromise-level attack was possible. This proof-of-concept attack gives us root access to a victim's VM, which means that we can read and write all data, install software, etc. An attacker needs the following information for a successful attack.

### *ID of the VM to attack.*

OpenNebula's VM ID is a global integer that is increased whenever a VM is instantiated. The attacker may simply guess the ID. Once the attacker can execute JavaScript code in the scope of Sunstone, it is possible to use OpenNebula's API to retrieve this ID based on the name of the desired VM or its IP address.

### *Operating system & bootloader.*

There are various ways to get to know a VMs OS, apart from simply guessing. For example, if the VM runs a publicly accessible webserver, the OS of the VM could be leaked in the HTTP-Header `Server` [14]. Another option would be to check the images or the template the VM was created from. Usually, the name and description of an image contains information about the installed OS, especially if the image was imported from a marketplace.

Since most operating systems are shipped with a default bootloader, making a correct guess about a VMs bootloader is feasible. Even if this is not possible, other approaches can be used (see below).

[3]`https://html5sec.org/videos/euca_console.mp4`

[4]`https://html5sec.org/videos/one_dos.mp4`

*Keyboard layout of the VM's operating system.*
As with the VMs bootloader, making an educated guess about a VM's keyboard layout is not difficult. For example, it is highly likely that VMs in a company's cloud will use the keyboard layout of the country the company is located in.

### 6.4.1 Overview of the Attack

The key idea of this attack is that neither Sunstone nor noVNC check whether keyboard related events were caused by human input or if they were generated by a script. This can be exploited so that gaining root access to a VM in OpenNebula requires five steps:

1. Using CSRF a persistent XSS payload is deployed.
2. The XSS payload controls Sunstone's API.
3. The noVNC window of the VM to attack is loaded into an iFrame.
4. The VM is restarted using Sunstone's API.
5. Keystroke-events are simulated in the iFrame to let the bootloader open a root shell.

The following sections give detailed information about each step.

*Executing Remote Code in Sunstone.*
In Sunstone, every account can choose a display language. This choice is stored as an account parameter (e. g. for English `LANG=en_US`). In Sunstone, the value of the `LANG` parameter is used to construct a `script` tag that loads the corresponding localization script. For English, this creates the following tag:

```
<script src="locale/en_US/en_US.js?v=4.6.1
    " type="text/javascript">
</script>
```

Setting the `LANG` parameter to a different string directly manipulates the path in the `script` tag. This poses an XSS vulnerability. By setting the `LANG` parameter to `LANG="onerror=alert(1)//`, the resulting `script` tag looks as follows:

```
<script src="locale/"onerror=alert(1)///"
    onerror=alert(1)//.js?v=4.6.1" type="
    text/javascript"></script>
```

For the web browser, this is a command to fetch the script `locale/` from the server. However, this URL points to a folder, not a script. Therefore, what the server returns is no JavaScript. For the browser, this is an error, so the browser executes the JavaScript in the `onerror` statement: `alert(1)`[5]. The rest of the line (including the second `alert(1)`) is treated as comment due to the forward slashes. When a user updates the language setting, the browser sends an XMLHttpRequest of the form

```
{"action":{"perform":"update","params":{"
    template_raw":"LANG=\"en_US\""}}}
```

to the server[6]. Forging a request to Sunstone from some other web page via the victim's browser requires a trick since one cannot use an XMLHttpRequest due to restrictions enforced by the browser's Same-Origin-Policy. Nevertheless, using a self-submitting HTML form, the attacker can let the victim's browser issue a POST request that is similar enough to an XMLHttpRequest so that the server accepts it.

An HTML form field like

```
<input name='deliver' value='attacker' />
```

is translated to a request in the form of

```
deliver=attacker
```

To create a request changing the user's language setting to `en_US`, the HTML form has to look like

```
<input name='{"action":{"perform":"update"
    ,"params":{"template_raw":"LANG' value
    ='\"en_US\""}}}' />
```

Notice that the equals sign in `LANG=\"en_US\"` is inserted by the browser because of the `name=value` format.

Using this trick, the attacker sets the `LANG` parameter for the victim's account to `"onerror=[remote code]//`, where `[remote code]` is the attacker's exploit code. The attacker can either insert the complete exploit code into this parameter (there is no length limitation) or include code from a server under the attacker's control. Once the user reloads Sunstone, the server delivers HTML code to the client that executes the attacker's exploit.

Due to the overwritten language parameter, the victim's browser does not load the localization script that is required for Sunstone to work. Therefore, the attacker achieved code execution, but Sunstone breaks and does not work anymore. For this reason, the attacker needs to set the language back to a working value (e. g. `en_US`) and reload the page in an iFrame. This way Sunstone is working again in the iFrame, but the attacker can control the iFrame from the outside. In addition, the attack code needs to disable a watchdog timer outside the iFrame that checks whether Sunstone is correctly initialized.

From this point on, the attacker can use the Sunstone API with the privileges of the victim. This way, the attacker can gather all required information like OpenNebula's internal VM ID and the keyboard layout of the VM's operating system from Sunstone's data-structures based on the name or the IP address of the desired VM.

*Compromising a VM.*
Using the Sunstone API the attacker can issue a command to open a VNC connection. However, this command calls `window.open`, which opens a new browser window that the attacker cannot control. To circumvent this restriction, the attacker can overwrite `window.open` with a function that creates an iFrame under the attacker's control.

Once the noVNC-iFrame has loaded, the attacker can send keystrokes to the VM using the `dispatchEvent` function. Keystrokes on character keys can be simulated using `keypress` events. Keystrokes on special keys (Enter, Tab, etc.) have to be simulated using pairs of `keydown` and `keyup` events since noVNC filters `keypress` events on special keys.

To get root access to a VM the attacker can reboot a victim's VM using the Sunstone API and then control the VM's bootloader by interrupting it with keystrokes. Once the attacker can inject commands into the bootloader, it is possible to use recovery options or the single user mode of

[5]Executing `alert(1)` is commonly used to prove that XSS exists.
[6]The original request contains more parameters. Since these parameters are irrelevant for the technique, we omitted them for readability.

Linux based operating systems to get a shell with root privileges. The hardest part with this attack is to get the timing right. Usually, one only has a few seconds to interrupt a bootloader. However, if the attacker uses the *hard reboot* feature, which instantly resets the VM without shutting it down gracefully, the time between the reboot command and the interrupting keystroke can be roughly estimated.

Even if the bootloader is unknown, it is possible to (1) use a try-and-error approach. Since the variety of bootloaders is small, one can try for one particular bootloader and reset the machine if the attack was unsuccessful. Alternatively, one can (2) capture a screenshot of the noVNC canvas of the VM a few seconds after resetting the VM and determine the bootloader.

A video of the attack is available[7]. The browser on the right hand side shows the victim's actions. A second browser on the left hand side shows what is happening in OpenNebula. The console window on the bottom right shows that there is no user-made keyboard input while the attack is happening.

## 6.5 XSS on openQRM

The web interface of openQRM displays message boxes about successful actions of the user and the content of these boxes is controllable via HTTP GET-parameter. However, the XSS filter applied by openQRM is insufficient to block script execution. If the URL

```
https://server/openqrm/base/index.php?base
    =event&event_msg=<scr<script>ipt>alert
    (1)</sc</script>ript>
```

is opened by the victim, an attacker can inject arbitrary code. Since the attacker can now use the access privileges of the victim, performing Control-level attacks is possible.

## 7. RESPONSIBLE DISCLOSURE

All bugs found in this research were secretly reported to the corresponding developers. As reaction to our report, OpenNebula published a maintenance release 4.6.2, in which the developers removed the XML sanitization bug and introduced a CSRF protection. The bugs in Eucalyptus 4.0.0 and 4.0.1 were filed under CVE-2013-4770 and CVE-2014-5039. They have been fixed with Eucalyptus 4.0.2. The persistent XSS bug in OpenStack has been filed under CVE-2014-3594. It has been fixed in OpenStack Havana (2013.2.4), OpenStack Icehouse (2014.1.2), and the development version of OpenStack Juno (Juno-3).

The security bugs found in openQRM have been confirmed by the developers. Currently, no fixes have been published yet.

## 8. RECOMMENDATIONS

The following general recommendations can be made for improving the security of private cloud deployments:

- *Restrict access to the CCI.* Most of the bugs we found were stored XSS vulnerabilities. By restricting access to the CCI, such attacks are limited to insiders. If remote administration of the CCI is required, either a VPN can be used, or TLS can be deployed with mutual authentication (i. e. client certificates).
- *Apply countermeasures to all known types of web attacks.* All but one system did a good job here, with the one exception of the missing CSRF protection in OpenNebula. The use of Content Security Policy should be considered as an option.

[7]`https://html5sec.org/videos/one_root.mp4`

More specifically, the attacks presented in this paper can be mitigated on the part of the user, the administrator, or the developer. Some of the countermeasures however prevent attacks at the expense of usability.

### *Mitigation Actions for a User.*

Users can protect themselves from CSRF by using separate browsers for using a cloud's web interface and browsing the web. Since a separate browser does not help against UI-Redressing and Clickjacking, we furthermore recommend using a powerful client-side filter like the aforementioned NoScript for Mozilla Firefox [7]. Unfortunately, configuring strong filter software correctly is hard for unexperienced users.

### *Mitigation Actions for an Administrator.*

At first, administrators of a CMP have to be aware that running a private behind a company's network perimeter does not let all security risks vanish. To minimize the risk, a CMP should always be updated to the latest stable release. Secondly, administrators of a private cloud should have a look on the wire communication of their web interfaces and decide whether certain security related HTTP headers should be used. For example, an `X-Frame-Options` header can be easily added using a server side proxy. Furthermore, administrators should decrease the validity period of sessions to reduce the likeliness that a session of a user who forgot to log out is exploited.

### *Mitigation Actions for a Developer.*

Developers of cloud management platforms have to keep web attacks in mind when developing a web interface, even if that web interface is not meant to be publicly accessible. As our results show, using the victim's browser behind the network protection perimeter as relay for attacks poses a realistic threat. Furthermore, the rights management should always consider the possibility that an attacker might control an account in the private cloud. Since a cloud management web interface is a single centralized place where all users – consumers as well as administrators – use the same functionality, privilege escalation attacks are very powerful.

Developers should not leave the task to add security related HTTP headers to administrators since some headers need to be supported by the web interface's code. Especially `Content-Security-Policy` allows enforcing fine-grained security settings; however, the code structure of the web application has to follow some constraints so that security is enhanced while functionality is retained.

Furthermore, we recommend that tools like noVNC cannot be used unless a user confirms that the action was not started by some malicious JavaScript. This check could be realized by a CAPTCHA [33] or requiring the user to enter a password.

## 9. REFERENCES

[1] AngularJS Home Page. [online] `http://www.angularjs.org/`.

[2] BeEF Home Page. [online] `http://beefproject.com/`.

[3] DevStack Home Page. [online] `http://devstack.org/`.

[4] Eucalyptus FastStart. [online] `https://www.eucalyptus.com/install`.

[5] Eucalyptus Home Page. [online] `http://www.eucalyptus.com/`.

[6] Howto: Install openQRM 5.1 on Debian Wheezy. [online] `http://openqrm-enterprise.com/resources/documentation-howtos/howtos/install-openqrm-51-on-debian-wheezy.html`.

[7] NoScript Home Page. [online] `http://noscript.net/`.

[8] noVNC Home Page. [online] `http://kanaka.github.io/noVNC/`.

[9] OpenNebula Home Page. [online] `http://opennebula.org/`.

[10] OpenNebula on Ubuntu 14.04 and KVM. [online] `http://docs.opennebula.org/4.6/design_and_installation/quick_starts/qs_ubuntu_kvm.html`.

[11] OpenQRM Home Page. [online] `http://www.openqrm-enterprise.com/`.

[12] OpenStack Home Page. [online] `http://openstack.org/`.

[13] Gorka Irazoqui Apecechea, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. Fine grain cross-vm attacks on xen and vmware are possible! *IACR Cryptology ePrint Archive*, 2014:248, 2014.

[14] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585.

[15] International Organization for Standardization. Information technology – cloud computing – overview and vocabulary. ISO 17788:2014, ISO, Geneva, Switzerland, 2014.

[16] Nils Gruschka and Luigi Lo Iacono. Vulnerable cloud: Soap message security validation revisited. In *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pages 625–631. IEEE, 2009.

[17] Marjan Gusev, Sasko Ristov, and Aleksandar Donevski. Security vulnerabilities from inside and outside the eucalyptus cloud. In *Proceedings of the 6th Balkan Conference in Informatics*, pages 95–101. ACM, 2013.

[18] M. Heiderich, M. Niemietz, F. Schuster, T. Holz, and J. Schwenk. Scriptless attacks–stealing the pie without touching the sill. In *ACM Conference on Computer and Communications Security (CCS)*, 2012.

[19] Mario Heiderich, Jörg Schwenk, Tilman Frosch, Jonas Magazinius, and Edward Z. Yang. mxss attacks: Attacking well-secured web-applications by using innerhtml mutations. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[20] Jann Horn. AngularJS Sandbox Bypasses. [online] `https://code.google.com/p/mustache-security/wiki/AngularJS#Sandbox_Bypasses`.

[21] Martin Johns. *Code Injection Vulnerabilities in Web Applications-Exemplified at Cross-site Scripting*. PhD thesis, University of Passau, 2011.

[22] Mathias Karlsson. AngularJS 1.2.19-1.2.23 / > 1.3.0-beta.14. [online] `https://code.google.com/p/mustache-security/wiki/AngularJS#AngularJS_1.2.19-1.2.23_/_%3E_1.3.0-beta.14`.

[23] Amit Klein. DOM based cross site scripting or XSS of the third kind. `http://www.webappsec.org/projects/articles/071105.shtml`, 2005.

[24] Peter Mell and Tim Grance. The NIST definition of cloud computing. *NIST Special Publication 800-145*, 2011.

[25] Chirag Modi, Dhiren Patel, Bhavesh Borisaniya, Avi Patel, and Muttukrishnan Rajarajan. A survey on security issues and solutions at different layers of cloud computing. *The Journal of Supercomputing*, 63(2):561–592, 2013.

[26] Marcus Niemietz and Jörg Schwenk. Ui redressing attacks on android devices. *Black Hat Abu Dhabi*, 2012.

[27] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan 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)*, pages 199–212. ACM, 2009.

[28] Sasko Ristov, Marjan Gusev, and Aleksandar Donevski. Openstack cloud security vulnerabilities from inside and outside. In *CLOUD COMPUTING 2013, The Fourth International Conference on Cloud Computing, GRIDs, and Virtualization*, pages 101–107, 2013.

[29] Peter Sempolinski and Douglas Thain. A comparison and critique of eucalyptus, opennebula and nimbus. In *CloudCom*, pages 417–426, 2010.

[30] Juraj Somorovsky, Mario Heiderich, Meiko Jensen, Jörg Schwenk, Nils Gruschka, and Luigi Lo Iacono. All Your Clouds are Belong to us – Security Analysis of Cloud Management Interfaces. In *The ACM Cloud Computing Security Workshop (CCSW)*, October 2011.

[31] Kuniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory deduplication as a threat to the guest os. In *Proceedings of the Fourth European Workshop on System Security*, page 1. ACM, 2011.

[32] Venkatanathan Varadarajan, Thawan Kooburat, Benjamin Farley, Thomas Ristenpart, and Michael M Swift. Resource-freeing attacks: improve your cloud performance (at your neighbor's expense). In *ACM Conference on Computer and Communications Security (CCS)*, pages 281–292. ACM, 2012.

[33] Luis Von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):56–60, 2004.

[34] William Zeller and Edward W. Felten. Cross-site request forgeries: Exploitation and prevention. `https://www.eecs.berkeley.edu/~daw/teaching/cs261-f11/reading/csrf.pdf`, 2008.

[35] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. Cross-vm side channels and their use to extract private keys. In *ACM Conference on Computer and Communications Security (CCS)*, pages 305–316. ACM, 2012.