TaheriMonfared and Jaatun *Journal of Cloud Computing: Advances, Systems and Applications* 2012, **1**:16
http://www.journalofcloudcomputing.com/content/1/1/16

Journal of Cloud Computing
a SpringerOpen Journal

RESEARCH Open Access

Handling compromised components in an IaaS cloud installation

Aryan TaheriMonfared[1*] and Martin Gilje Jaatun[2]

Abstract

This article presents an approach to handle compromised components in the OpenStack Infrastructure-as-a-Service cloud environment. We present two specific use cases; a compromised service process and the introduction of a bogus component, and we describe several approaches for containment, eradication and recovery after an incident. Our experiments show that traditional incident handling procedures are applicable for cloud computing, but need some modification to function optimally.

Introduction

Although Cloud Computing has been heralded as a new computing model, it is fundamentally an old idea of providing computing resources as a utility [1]. This computing model will reduce the upfront cost for developing and deploying new services in the Internet. Moreover, it can provide efficient services for special use-cases which require on-demand access to scalable resources.

Cloud Computing has a variety of service models and deployment models which have been in use in various combinations for some time [2]. The chosen service and deployment model of a cloud environment will determine what kind of vulnerabilities might threaten it. One of the main obstacles in the movement toward Cloud Computing is the perceived insufficiency of Cloud security. Although it has been argued [3] that most of the security issues in Cloud Computing are not fundamentally novel, a new computing model invariably brings its own security doubts and issues to the market.

In a distributed environment with several stakeholders, there will always be numerous ways of attacking and compromising a component, and it is not possible to stop all attacks or ensure that the system is secure against all threats. Thus, instead of studying attack methods, a better approach is to assess the risk and try to understand the impact of a compromised component. To do this, the exact functionalities of each component must be determined, after which efficient approaches to tolerate such an attack can be identified. The first step of this process is to detect, and then analyze the incident, something which is subject to a set of best practice procedures which are dependent on knowledge about the normal behavior and operation of the system. The next step is about containing the incident. There are currently several public cloud providers; however, none of them disclose their security mechanisms. This highlights the need to study applicable mechanisms and introduce new ones to fulfill security requirements of a given cloud environment; in this article, we describe our work on an open-source deployment of a cloud environment based on the OpenStack cloud platform. When we talk about a compromised component in this document, we mean those components in a cloud environment that are disclosed (i.e., private contents revealed), modified, destroyed or even lost [4]. Finding compromised components and identifying their impacts on a cloud environment is crucial.

A brief primer on OpenStack

We have found the OpenStack cloud platform to be the best choice for a real case study in our research. In our laboratory configuration, we used the simple flat deployment structure. This will avoid further complexity which would be caused by a hierarchical or peer-to-peer architecture. We have four physical machines; one of them will be the cloud controller, and other three are compute worker nodes. The abstract diagram of our lab setup is depicted in Figure 1. It should be noted that although we focus on the OpenStack as a specific cloud software in our study, more or less the same components and processes can be found in other cloud platform implementations.

*Correspondence: aryan@uninett.no
[1]UNINETT, Trondheim, Norway
Full list of author information is available at the end of the article

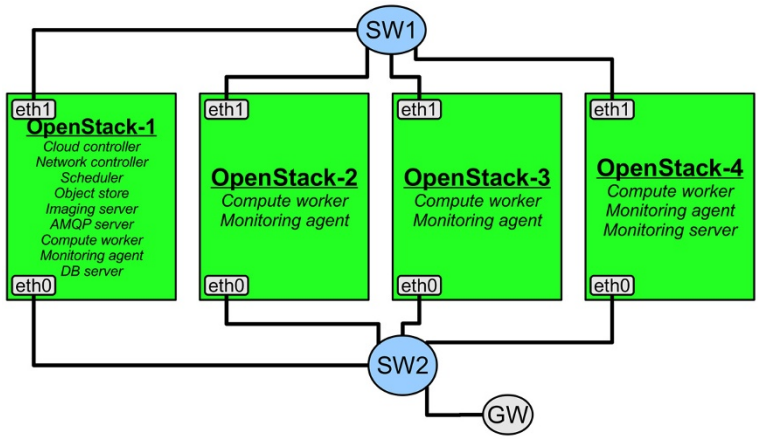


Figure 1 Lab setup.

OpenStack consists of a set of open-source projects which provide a variety of services for an Infrastructure as a Service (IaaS) model. Its five main projects deliver basic functionalities that are required for a cloud infrastructure, comprising: Nova (compute), Swift (storage), Glance (VM image), Keystone (identity), and Horizon (dashboard). The OpenStack community is fairly big, with a lot of leading companies involved. A big community for an open-source project has its own advantages and disadvantages, but further discussion on this topic is out of the scope of this article.

The Compute project (Nova) provides fundamental services for hosting virtual machines in a distributed yet connected environment. It handles provisioning and maintenance of virtual machines, as well as exposing appropriate APIs for cloud management. The object storage project (Swift) is responsible for delivering a scalable, redundant, and permanent object storage. It does not facilitate a regular file system in the cloud. Virtual machine disk images are handled by the Image Service project (Glance). Discovery, uploading, and delivery of images are exposed using a REST[a] interface. The image service does not store the actual images, but utilizes other storage services for that purpose, such as OpenStack Object Storage. The identity project (Keystone) unifies authentication for the deployed cloud infrastructure. Cloud services are accessible through a portal provided by the dashboard project (Horizon) [5].

The OpenStack architecture is based on a Shared Nothing (SN) and Message Oriented architecture. Thus, most of the components can run on multiple nodes and their internal communication functions in a synchronous fashion via a messaging system. In this deployment (and in the default installation of OpenStack), RabbitMQ is used as the messaging system. RabbitMQ is based on the Advanced Messaging Queue Protocol (AMQP) standard. These architectures are used to avoid common challenges in a distributed environment, such as deadlock, live lock, etc.

We have decided to focus on the Compute project of OpenStack, which has enabled us to dive deeply into the details, and exercise different modules in the Compute project. However, the same results are applicable to the rest of the OpenStack projects. All projects follow the same architectural concepts and design patterns, so despite their functionalities, their behavior in a distributed and highly scalable environment would be similar.

OpenStack Compute has 5 interacting modules, comprising: compute controller, network controller, volume controller, scheduler, and API server. These modules are depicted in Figure 2. They provide basic functionalities for hosting, provisioning and maintaining virtual machine instances. The compute controller interacts with the underlying hypervisor and allocates required resources for each virtual machine. The network controller provides networking resources (e.g. IP addresses, VLAN specification, etc.) for VM instances. The volume controller handles block storages devices for each VM instance, and the scheduler distributes tasks among worker nodes (i.e. compute controllers). The API server exposes all these functionalities to the outside world.

Article structure

We will continue to discuss general aspects of incident handling in a specific cloud environment, and our case studies for possible attack scenario to such a model.

The rest of the paper is structured as follows: First, we will explain the adapted form of the NIST incident handling guideline for the cloud model (Section "Incident handling"). Then two incidents will be processed by the adapted guideline (Section "Case studies"). Applying the guideline leads us to a set of new challenges that have

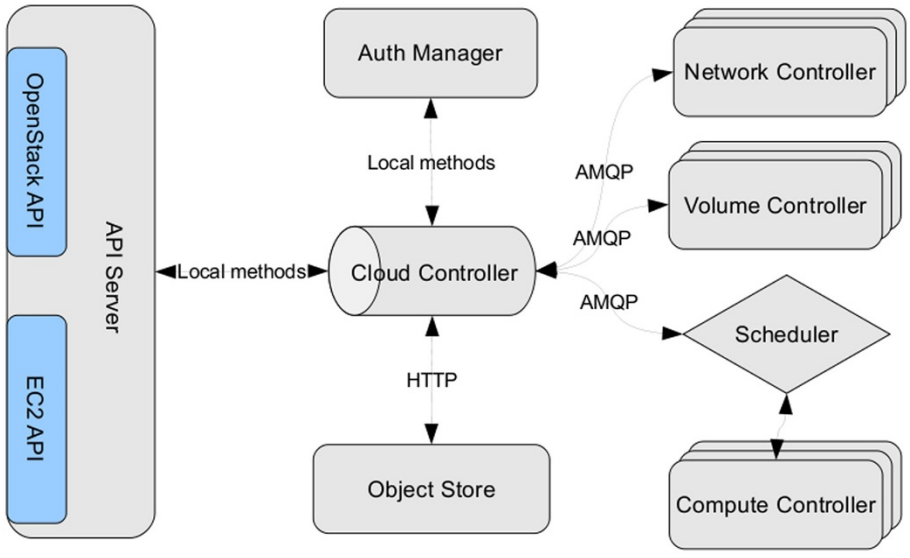


Figure 2 OpenStack Compute basic architecture [6].

not been addressed previously or require a careful re-analysis. Finally, by analyzing these challenges, a group of security mechanisms are proposed which address existing deficiencies (Section "Approaches for containment and recovery"). A brief comparison of mechanisms are provided as well (Section "Comparison").

Incident handling

We will focus on cloud platform components, specifically on their functionalities, access methods, interacting components and the impacts in case of being compromised. The symptoms of a compromised component are useful in detecting security breaches and must be considered when performing further analysis. Studying the detection and analysis phase of the incident handling procedure, and applying new characteristics of the Cloud Computing model, we identified several requirements for a cloud provider and a cloud consumer. Additionally, some influential challenges which will hinder implementation of these requirements or adaptation of existing mechanisms will be explained.

Detection and analysis of the compromised component

Studying the detection and analysis phase of the NIST incident handling guideline [7], and applying new characteristics of the Cloud Computing model, we identified several requirements for a cloud provider and a cloud consumer.

Cloud provider requirements

The cloud provider should develop the following items to play its role in the incident handling process. Most of these items are orthogonal. In other words, a cloud consumer may request several items (i.e. security functionalities, services) together. Also, different consumers may not have similar demands. Thus, it may be beneficial for the provider to develop most (if not all) of the following items if it wishes to cover a larger set of consumers.

- **Security APIs:** The cloud provider should develop a set of APIs that deliver event monitoring functionalities and also provide forensic services for authorities. Event monitoring APIs ease systematic incident detection for cloud consumers and even third parties. Forensic services at virtualization level can be implemented by means of virtual machine introspection libraries. An example of an introspection library is XenAccess that allows a privileged domain to access live states of other virtual machines. A cross-layer security approach seems to be the best approach in a distributed environment [8].
- **Precursor or Indication Sources:** The cloud provider deploys, maintains and manages the cloud infrastructure. The provider also develops required security sensors, logging and monitoring mechanisms to gather enough data for incident detection and analysis at the infrastructure level. As an example, security agents, intrusion monitoring sensors, application log files, report repository, firewall statistics and logs are all part of security relevant indication sources. In case of a security incident, the cloud provider should provide raw data from these sources to affected customers and stakeholders. Thus they will be capable of analyzing raw data and characterizing incident properties.
- **External reports:** The cloud provider should provide a framework to capture external incident reports. These incidents can be reported by cloud consumers,

end users or even third parties. This is not a new approach in handling an incident, however finding the responsible stakeholders for that specific incident and ensuring correctness of the incident[b] requires extensive research. E.g., Amazon has developed a "Vulnerability Reporting Process" [9] which delivers these functionalities.

- **Stakeholder interaction:** A timely response to an incident requires heavy interaction with stakeholders. In order to ease this interaction at the time of crisis, the responsibilities of each stakeholder should be described in detail.
- **Security services:** Cloud consumers may not be interested in developing security mechanisms. The cloud provider can deliver a security service to overcome this issue. Security services which are delivered by the provider can be more reliable in case of an incident and less challenging in the deployment and the incident detection/analysis phases.
- **Infrastructure information:** When the cloud consumer or another third party wants to develop incident detection and analysis mechanisms, they will need to understand the underlying infrastructure and its architecture. However, without cloud provider cooperation that will not be feasible. So, the cloud provider should disclose enough information to responsible players to detect the incident in a timely fashion and study it to propose the containment strategy.

Cloud consumer requirements

A cloud consumer must fulfill requirements to ensure effectiveness of the incident detection and analysis process.

- **Consumer's security mechanisms:** The cloud consumer might prefer to develop its own security mechanisms (e.g. incident detection and analysis mechanisms). The customer's security mechanisms can be based on either the cloud provider's APIs or reports from a variety of sources, including: provider's incident reports, end-users' vulnerability reports, third parties' reports.
- **Provider's agents in customer's resources:** By implementing provider's agents, the cloud consumer will facilitate approaching a cross-layer security solution. In this method, the cloud consumer will know the exact amount and type of information that has been disclosed. Moreover, neither the cloud consumer nor the provider needs to know about each others' architecture or infrastructure design.
- **Standard communication protocol:** In order to have systematic incident detection and analysis mechanisms, it is required to agree on a standard communication protocol that will be used by all stakeholders. This protocol should be independent of a specific provider/customer.
- **Report to other stakeholders:** If the customer cannot implement the provider's agent in its own instances, another approach to informing stakeholders about an incident is by means of traditional reporting mechanisms. These reports should not be limited to an incident only, customers may also use this mechanism to announce a suspicious behavior for more analysis.
- **Cloud consumer's responsibilities:** Roles and responsibilities of a cloud consumer in case of an incident should be defined ahead of time, facilitating immediate reaction in a crisis.

Case studies

We now present two examples that illustrate handling a compromised node and an introduced bogus node, respectively.

Case One: a compromised compute worker

In the first case only one component, the nova-compute service in the compute worker, is compromised, as shown in Figure 3. Two incidents have happened simultaneously in this scenario: malicious code and unauthorized access. The malicious code is injected to the nova-compute service and introduces some misbehavior in it, such as malfunctions in the hosting service of virtual machine instances.

A malfunction can be provoked, e.g., through nefarious use of granted privileges to request more IP addresses, causing IP address exhaustion. The incident description for this scenario is given in Table 1.

The malicious code is injected after another incident, unauthorized access. The attacker gains access to resources on the OpenStack-4 host, that he/she was not intended to have. Using those escalated privileges, the attacker changed the python code of the nova-compute and restarted the service, causing it to behave maliciously.

Recommended actions by NIST and their corresponding realization in an OpenStack deployment are explained next. They will fulfill requirements, implied by the containment, eradication, and recovery phase. As explained before, the described scenario consists of two incidents, *unauthorized access* and *malicious code*. Thus, we will in the following briefly discuss recommended responses for both types of incident; an extended discussion can be found in [10].

The following discussion is given in two parts. In each part, actions proposed by the NIST guideline are adapted to the cloud model. First, containment actions from Table 2 will be adapted. Then, adapted forms of eradication and recovery actions are explained. A major effort has been put into adapting containment actions:

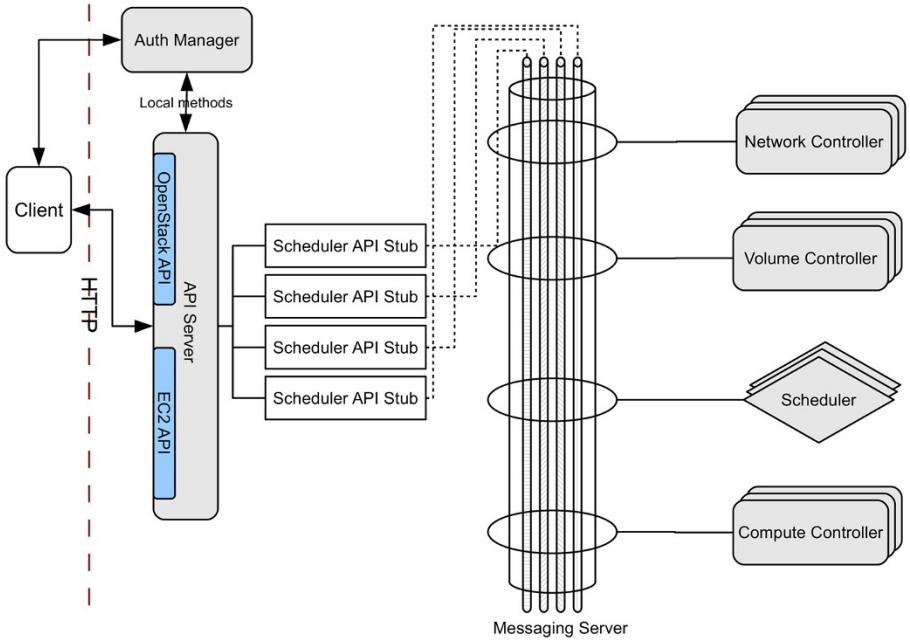


Figure 3 Case One - The nova-compute service in the OpenStack-4 host is compromised.

- **"Identifying and Isolating Other Infected Hosts"** Study the profile of the infected host and compare it to other worker nodes' profiles, in order to identify compromised hosts. Comparing profiles of components is simple, using provided monitoring facilities in our experimental environment.
- **"Blocking Particular Hosts"** The strategy should be analyzed in depth before its application. In a cloud environment when the consumer's instance is running in an infected worker node, it is not reasonable to disconnect the node without prior notice/negotiation to affected consumers (This constraint can be relaxed by providing the proper Service-level Agreement (SLA)). In addition, blocking the compromised host can be done with different levels of restrictions. Initially the communication with the outside of the organization should be blocked[c] assuming that the attacker is located outside of the organization infrastructure. Also, any further attack to the outside of the organization using compromised hosts will be mitigated.
  In the second step, communication of the compromised host with other components in the infrastructure is also restricted and the host is marked as compromised/infected/suspicious. Thus, other nodes will avoid non-critical communication with the compromised node. It will help the infrastructure to communicate with the compromised node for containment, eradication, and recovery procedures; and at the same time the risk of spreading the infection is reduced.
  The last step can be blocking the host completely. In this approach staff should access the host directly for analyzing the attack as well as assessing possible mitigation and handling strategies.
  However, blocking infected hosts will not contain the incident. Each host has several consumers' instances (VM instances) and volumes running on and attached to it. Blocking hosts will only avoid spreading the

Table 1 Case One - A compromised compute worker scenario specifications

	Incident description
Incident type	Malicious code and Unauthorized access
Current status	Ongoing attack, the malicious code is not patched nor contained yet
Compromised component(s)	One compute worker host
Physical Location	OpenStack-4
Affected Layers	Cloud platform layer, the OpenStack nova-compute service
General Information	Malicious code is injected into the nova-compute service of the OpenStack-4 host
Resources at risk	Running instances on OpenStack-4, Stakeholders and resources interacting with running instances on OpenStack-4 or the infected nova-compute service

Table 2 Containment strategies

NIST recommended action	Brief description
"Identifying and Isolating Other Infected Hosts"	Extract incident symptoms to detect other infected hosts.
"Blocking Particular Hosts"	After identifying the compromised component and its corresponding host (i.e. the compromised worker/compute host), that host should be blocked.
"Soliciting User Participation"	Interaction among cloud stakeholders (e.g. cloud providers, cloud consumers, third parties, end users, etc.) is a mandatory step toward fulfilling incident containment requirements.
"Disabling Services"	Disabling the infected service (nova-compute in our scenario) may reduce impacts of the compromised host. Disabling a service can disrupt other services and cause deviation from promised SLA by the provider.

incident to other hosts but instances are still in danger. An approach in a cloud environment is to disconnect instances and volumes from the underlying compromised layer. Signaling the cloud software running on the compromised host to release/terminate/shutdown/migrate instances and detach volumes are our proposed approaches. This approach is illustrated in Figure 4. We should use a quarantine compute worker node as the container for migrated instances. After ensuring the integrity and healthiness of instances, they can be moved to a regular worker node. This quarantine compute worker will be explained more in the following section. These approaches can be implemented at the cloud infrastructure layer for simplicity (Blocking by means of nodes firewall, routers, etc.)

- **"Soliciting User Participation"**
  The interaction can be implemented using different methods. Distributing security bulletins maintained by cloud or service providers is an example of notifying other stakeholders about an incident. Incident or vulnerability reporting mechanisms are also useful when an outsider detects an incident or identifies a vulnerability. These two methods can be developed and deployed independently of the cloud platform. Security bulletins are provided by the security team who handles security related tasks. Also, reporting mechanisms are delivered by means of ticketing and reporting tools. Direct and real-time communication among stakeholders is a complement to the above mentioned methods.
- **"Disabling Services"**
  In order to disable a particular service, we should first check the service dependencies diagram. An example of such a diagram is depicted in Figure 5. Disabling a service can take place in two ways.

It is possible to stop the service at the compromised host (Figure 6). In our scenario we can stop the nova-compute service to disable the compute service. It will instantly disconnect the cloud platform from running VM instances. *In the OpenStack platform stopping the nova-compute service will not terminate running instances on that host.* Thus, although the compute service is not working anymore, already running instances will continue to work even after nova-compute is terminated. Additionally, it is not possible to terminate an instance after stopping its corresponding compute service, because the

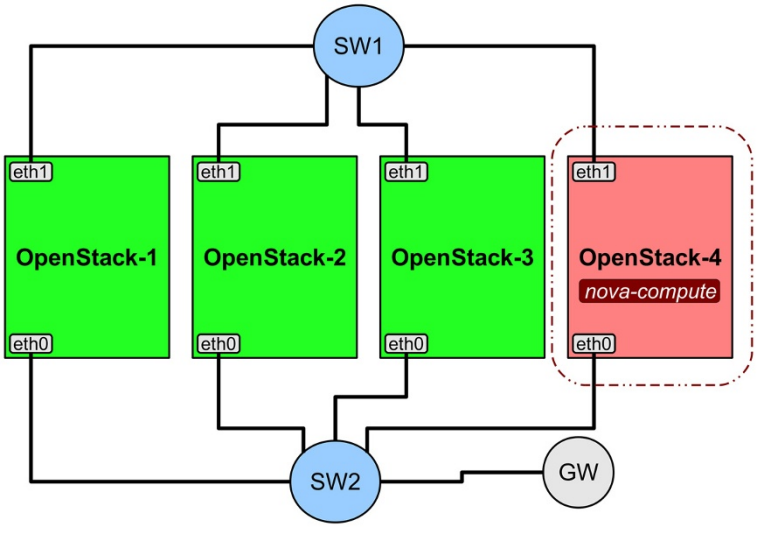


Figure 4 Blocking compromised compute communication. Red lightening represent disconnected communications.

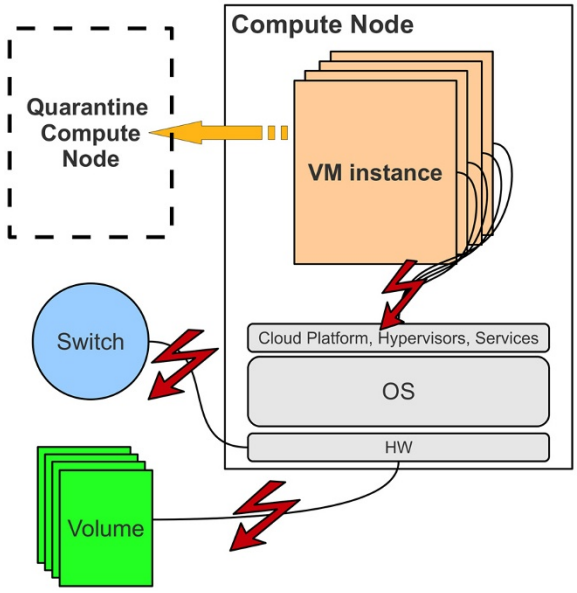


Figure 5 OpenStack Nova service dependencies.

administration gateway (i.e. nova-compute) is not listening to published messages. In order to maintain control over running instances we should migrate instances from the compromised node to a quarantine node before we terminate the compute service.

Another approach is discarding messages published by the compromised component or those destined to it (Figure 7). This is a centralized method and the cloud controller or the messaging server should filter out messages with the source/destination of the infected host[d]

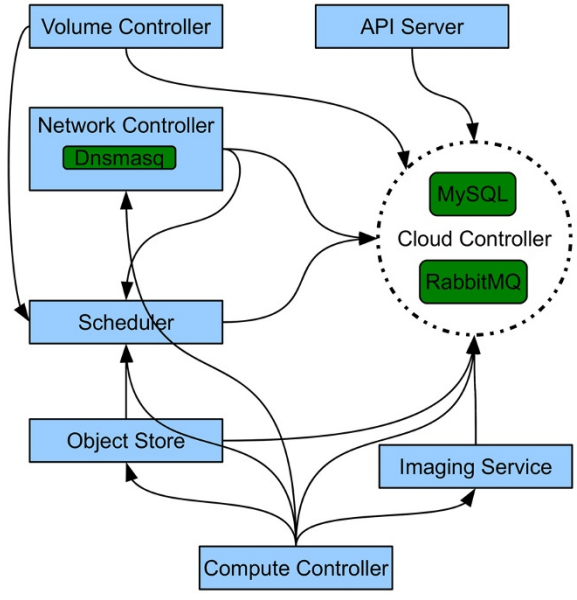


Figure 6 Stopping the compute service at the compromised host.

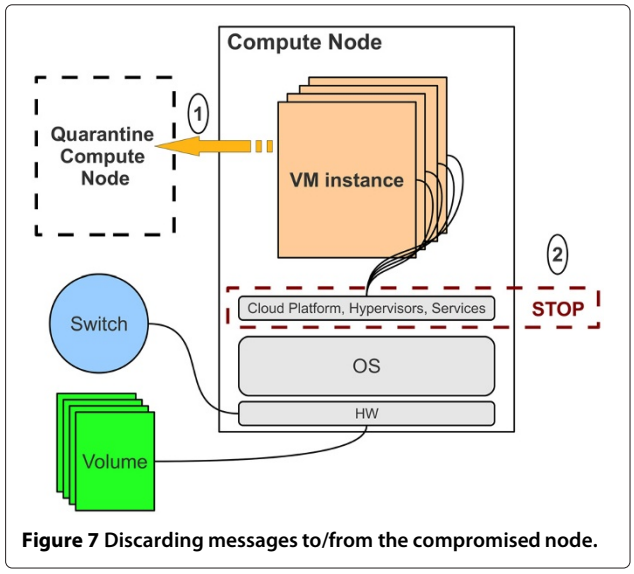


Figure 7 Discarding messages to/from the compromised node.

We continue by explaining four other actions which are recommended responses to an unauthorized access incident:

- **"Isolate the affected systems"**
  The same procedures as those which have been explained for "Identifying and Isolating Other Infected Hosts" (Section "Case One: a compromised compute worker") and "Blocking Particular Hosts" (Section "Case One: a compromised compute worker") can be applied here.
- **"Disable the affected service"**
  The same procedure as the one which has been explained for "Disabling Services" (Section "Case One: a compromised compute worker") can be applied here.
- **"Eliminate the attacker's route into the environment"**
  Access methods which have been used by the attacker to access cloud components should be blocked. Implementing filtering mechanisms in the messaging server is a crucial requirement which is highlighted in different strategies. The cloud provider should be capable of blocking messages which are related to the attack and blocks the attacker's route into the cloud environment. It should be noted that the mechanisms which we have used to meet requirements imposed by "Blocking Particular Hosts", "Identifying and Isolating Other Infected Hosts", "Disabling Services" (Section "Case One: a compromised compute worker") are appropriate actions for eliminating attackers' routes.
- **"Disable user accounts that may have been used in the attack"**
  A compromised user account may reside in multiple layers, such as the system, cloud platform, or VM

instance layer[e]. Based on the membership layer, the disabling and containment procedure will differ. Additionally, in each layer a variety of user types exist. As an example, in the cloud platform layer, the cloud provider's staff and cloud consumers have a different set of user types.

As it was explained, three phases are adapted. Containment phase was discussed, and eradication phase is the next one to be studied:

- **"Disinfect, quarantine, delete, and replace infected files"**
  These strategies are applicable in two layers depending on the container of the injected malicious code. The malicious code can be injected into either the cloud platform services (i.e. nova-compute) or the OS modules/services. If the injected malicious code is in OS modules/services, utilizing existing techniques is effective. By existing techniques, we refer to anti virus software and traditional malware handling mechanisms. In this case nothing new has happened, although side effects of the incident may vary a lot. However, if the malicious code is injected into a cloud platform service (in our case nova-compute), existing anti virus products are not useful, as they are not aware of the new context. Cleaning a cloud platform service can be very hard, so other approaches are more plausible. In general, we can propose several approaches for eradicating a malicious code incident in a cloud platform:
    - Updating the code to the latest stable version and apply appropriate patches to fix the vulnerability.
    - Purging the infected service on the compromised node
    - Replacing the infected service with another one that uses a different set of application layer resources (e.g. configuration files, repositories, etc.)

  It should be noted that in a highly distributed system such as a cloud environment, doing complicated tasks such as fixing a single infected node in real time fashion does not support the cost effectiveness policy. Thus, terminating the infected service or even the compromised node and postponing the eradication phase can be an appropriate strategy.
- **"Mitigate the exploited vulnerabilities for other hosts within the organization"**
  In order to complete the task, we should also update the cloud platform software on other nodes and patch identified vulnerabilities.

The last phase is about recovery of the system which was under attack:

- **"Confirm that the affected systems are functioning normally"**
  Profiling the system is useful in the recovery phase as well as in the detection and analysis phase. After containment and eradication of the compromised component, the component profile should be the same as a healthy component or be the same as its own profile before being infected. Using the provided tools in our deployment (i.e. Cacti) we can specify the exact period and components which we want to compare.
- **"If necessary, implement additional monitoring to look for future related activity"**
  After identifying attack patterns and the compromised node profile, we should add proper monitoring alarms to cover those patterns and profiles. As an example, if the compromised compute worker starts to request a large number of IP addresses after its infection, this pattern should be saved and monitored on other compute workers. So, if we experience a compute worker with the same profile and behavior, that worker node will flagged as possibly infected. In our monitoring tools, the administrator can define a threshold for different parameters; if the current profile of the system violates the threshold, graphs will be drawn with a different color to notify the user. We can also add other monitoring tools to generate the ticket in case of a matching profile.

Case Two: a bogus component

A bogus service is a threat to OpenStack is an open source software, an attacker can access the source code or its binaries and start a cloud component that delivers a specific service. When the attacker is managing a service, he/she can manipulate the service in a way that threatens the integrity and confidentiality of the environment. This section will discuss such an incident, where a bogus nova-compute service is added to the cloud environment. The incident description for this case is given in Table 3.

A bogus nova-compute service (or, in general, any cloud platform component) can run on a physical machine or a virtual instance. It is unlikely that an attacker will be capable of adding a physical node to the cloud infrastructure; however, for the sake of completeness we study both the case that the bogus service is running on a new physical machine and the one where it is running on a virtual instance. Both cases are depicted in Figures 8 and 9.

When the bogus service is running on top of an instance, the network connectivity may be more limited than compared to the other case (i.e., the bogus service is running

Table 3 Case Two - A bogus component scenario specifications

	Incident description
Incident type	Inappropriate Usage
Current status	Ongoing attack, the bogus compute worker is still up and serving a part of requests
Physical Location	OpenStack-5
Affected Layers	Cloud platform layer, the OpenStack nova-compute service, consumers' instances
General Information	A bogus compute worker node is added to the platform, it is a threat to the provider's and consumers' data confidentiality and integrity. Also a threat for the system availability.
Resources at risk	Running instances on OpenStack-5, Stakeholders and resources interacting with running instance on OpenStack-5

on a physical node). Initially, any given instance is only connected to the second interface (*eth1*). This connectivity is provided by means of the bridge connection (*br100*) that connects virtual interfaces (*vnetX*) to the rest of the environment. Thus, a running instance has no connectivity to the switch *SW2* by default. However, connectivity to the outside world can be requested by any consumer (e.g., an attacker) through a legitimate procedure. Thus, in Figure 9, we also connect the instance to *SW2*.

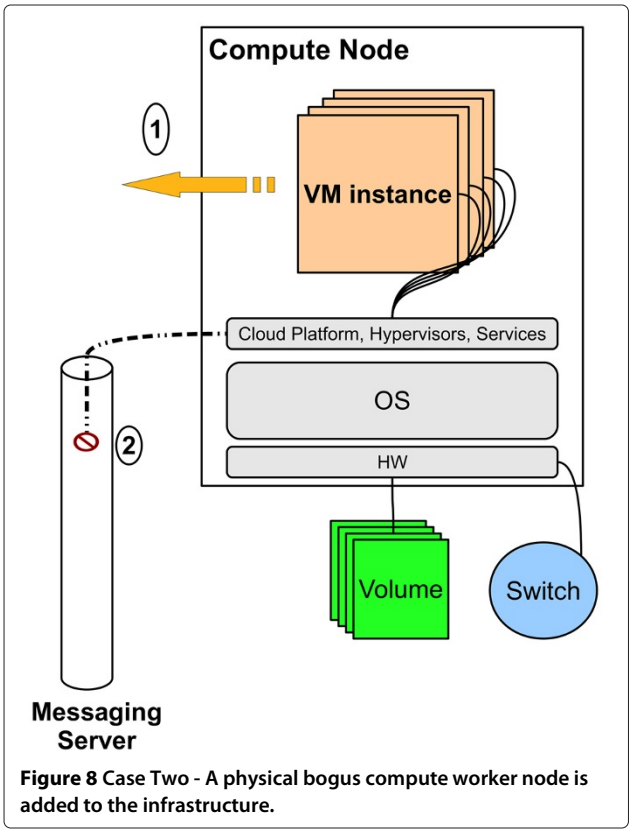


Figure 8 Case Two - A physical bogus compute worker node is added to the infrastructure.

We simulated the virtual bogus compute worker by deploying the nova-compute service on a running instance. There were multiple obstacles for simulating this scenario, including: the running instance, which turns out to be also a bogus worker, must have hosting capabilities; the bogus worker must respond to cloud controller requests to be recognized as a working node.

Detecting a bogus worker node or instance is a complex task if the infrastructure has not previously employed a proper set of mechanisms. However, a few parameters can be monitored as an indication of a bogus worker. Generally, a bogus worker is not working as well as a real one, because its main goal is not providing a regular service. A bogus worker aims to steal consumers' data, intrude on the cloud infrastructure, disrupt the cloud environment Quality of Service (QoS), and so forth. Without any prior preparation, a suspicious worker can be identified by monitoring the service availability and QoS parameters on each worker. Moreover, a suspicious virtual worker can also be recognized because of its high traffic towards the cloud infrastructure messaging servers.

Containing a bogus worker consists of both proactive and reactive techniques. When a bogus worker is detected, the containment procedure is fairly simple (i.e., applying reactive techniques). However, deploying a set of proactive techniques is more challenging. These techniques can be implemented as a group of security mechanisms and policies, such as *node authentication, manual confirmation, trust levels and timeouts*, and *no new worker policy*. They will be discussed further in Section "Policies".

Approaches for containment and recovery

This section introduces our proposed approaches for containment of intruders, eradication of malicious processes and recovery from attack. The proposed strategies can be grouped based on two criteria: The responsible stakeholder for developing and deploying the strategy, and the target layer for that strategy. Based on the first criterion we may have either the cloud provider or the cloud consumer as the responsible stakeholder; based on the second criterion, the target layer can be either the infrastructure/hardware layer or the service/application layer. We have devised a set of approaches which will be explained in detail in the following.

Restriction, disinfection, and replication of infected cloud platform components

A general technique for containing an incident is restricting the infected component. The restriction can be applied in different layers, with a variety of approaches,

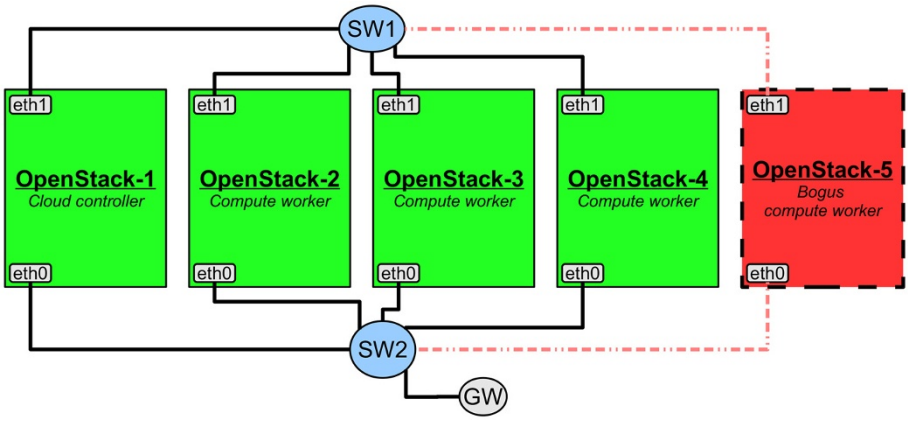


Figure 9 Case Two - A virtual bogus compute worker is added as a consumer's instance.

such as: filtering in the AMQP server, filtering in other components, disabling the infected service or disabling the communicator service. Additional measures can also be employed to support the restriction, like: removing infected instances from the project VLAN, disabling live migration, or quarantining infected instances. We explain each of these approaches in the following sections.

Filtering in the messaging server (cloud controller)

We will propose several filtering mechanisms in the messaging server in order to contain and eradicate an incident in a cloud environment. The OpenStack platform has been used to build our experimental cloud environment. This approach is a responsibility of the cloud provider and the target layer in the cloud platform application layer.

Advantages

- The filtering task at the messaging server level can be done without implementation of new functionality. We can use existing management interfaces of the RabbitMQ (either command line or web interface) to filter the compromised component.
- The filtering task can be done in a centralized fashion by means of the management plug-in, although we may have multiple instances of the messaging server.
- Implementing this approach is completely transparent for other stakeholders, such as cloud consumers.
- We can scale out[f] the messaging capability by running multiple instances of the RabbitMQ on different nodes. Scaling out the messaging server will also scale out the filtering mechanism[g].
- This approach is at the application layer, and it is independent of network architecture and employed hardware.
- The implementation at the messaging server level helps in having a fine-grained filtering, based on the message content.

Disadvantages

- A centralized approach implies the risk of a single point of failure or becoming the system bottleneck.
- Implementing the filtering mechanism at the messaging server and/or the cloud controller adds an extra complexity to these components.
- When messages are filtered at the application layer in the RabbitMQ server, the network bandwidth is already wasted for the message that has an infected source, destination, or even context. Thus, this approach is less efficient than one that may filter the message sooner (e.g. at its source host, or in the source cluster).
- Most of the time application layer approaches are not as fast as those in the hardware layer. In a large scale and distributed environment the operation speed plays a vital role in the system availability and QoS. It is possible to use the zFilter technique [11] as a more efficient implementation of the message delivery technique. It can be implemented on either software or hardware. The zFilter is based on the bloom-filter [12] data structure. Each message contains its state; thus this technique is stateless [11]. It also utilizes source routing. zFilter implementations are available for the BSD family operating systems and the NetFPGA boards at the following address, *http://www.psirp.org*.
- Filtering a message without notifying upper layers may lead to triggered timeouts and resend requests from waiting entities. It can also cause more wasted bandwidth.

Realization A variety of filtering mechanisms can be utilized in the messaging server; each of these mechanisms focuses on a specific component/concept in the RabbitMQ messaging server. We can enforce the filtering in the messaging server *connection*, *exchange*, and *queue* as will be discussed next.

- **Connection:** A connection is created to connect a client to an AMQP broker [13]. A connection is a long-lasting communication capability and may contain multiple channels [14]. By closing the connection, all of its channels will be closed as well. A snapshot of connections in our OpenStack deployment is available in Figure 10.
- **Exchange:** An exchange is a message routing agent which can be durable, temporary, or auto-deleted. Messages are routed to qualified queues by the exchange. A Binding is a link between an exchange and a queue. An exchange type can be one of *direct, topic, headers,* or *fanout* [15]. An exchange can be manipulated in different ways in order to provide a filter mechanisms for our cloud environment:
  - **Unbinding a queue from the exchange:** The compromised component queue won't receive messages from the unbound exchange. As an example, we assume that the compute service of the OpenStack-4 host is compromised. Now, we want to block nova traffic to and from the compromised compute service; so, we unbind the NOVA topic exchange from the queue COMPUTE.OPENSTACK-4. The RabbitMQ management interface is used to unbind the exchange, as shown in Figure 11.
  - **Publishing a warning message:** Publishing an alert message to that exchange, so all clients using that exchange will be informed about the compromised component. Thus, by specifying the compromised component, other clients can avoid communicating with it. The main obstacle in this technique is the requirement for implementing new functionalities in clients.
  - **Deleting the exchange:** Deleting an exchange will stop routing of messages related to it. It may have multiple side effects, such as memory overflow and queue exhaustion.
- **Queue:** The queue is called a "weak FIFO" buffer; each message in it can be delivered only to a single client unless re-queuing the message [15].
  - **Unbinding** a queue from an exchange avoids further routing of messages from that exchange to the unbound queue. We can unbind the queue which is connected to the compromised component and stop receiving messages by the infected client.
  - **Deleting** a queue not only removes the queue itself, but also remove all messages in the queue and cancel all consumers on that queue.
  - **Purging** a queue removes all messages in the queue that do not need acknowledgment. Although it may be useful in some cases, it may not be as effective as required during an incident.

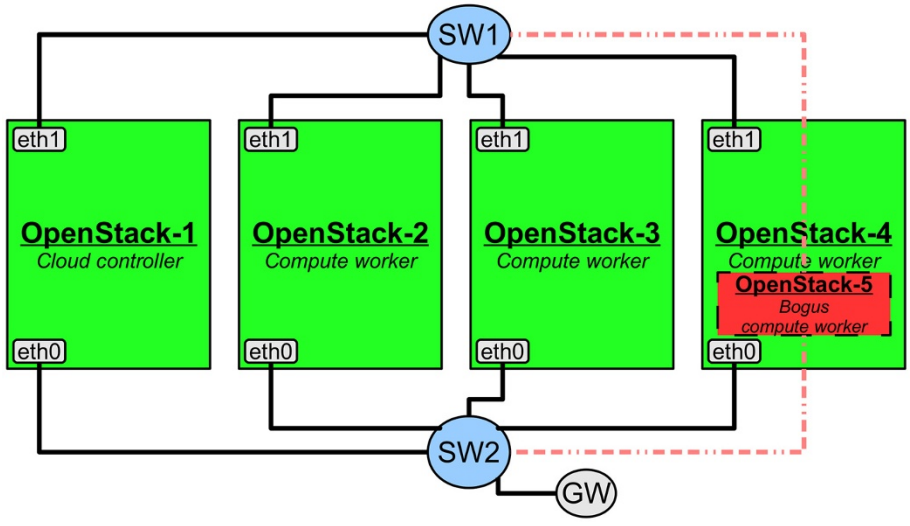
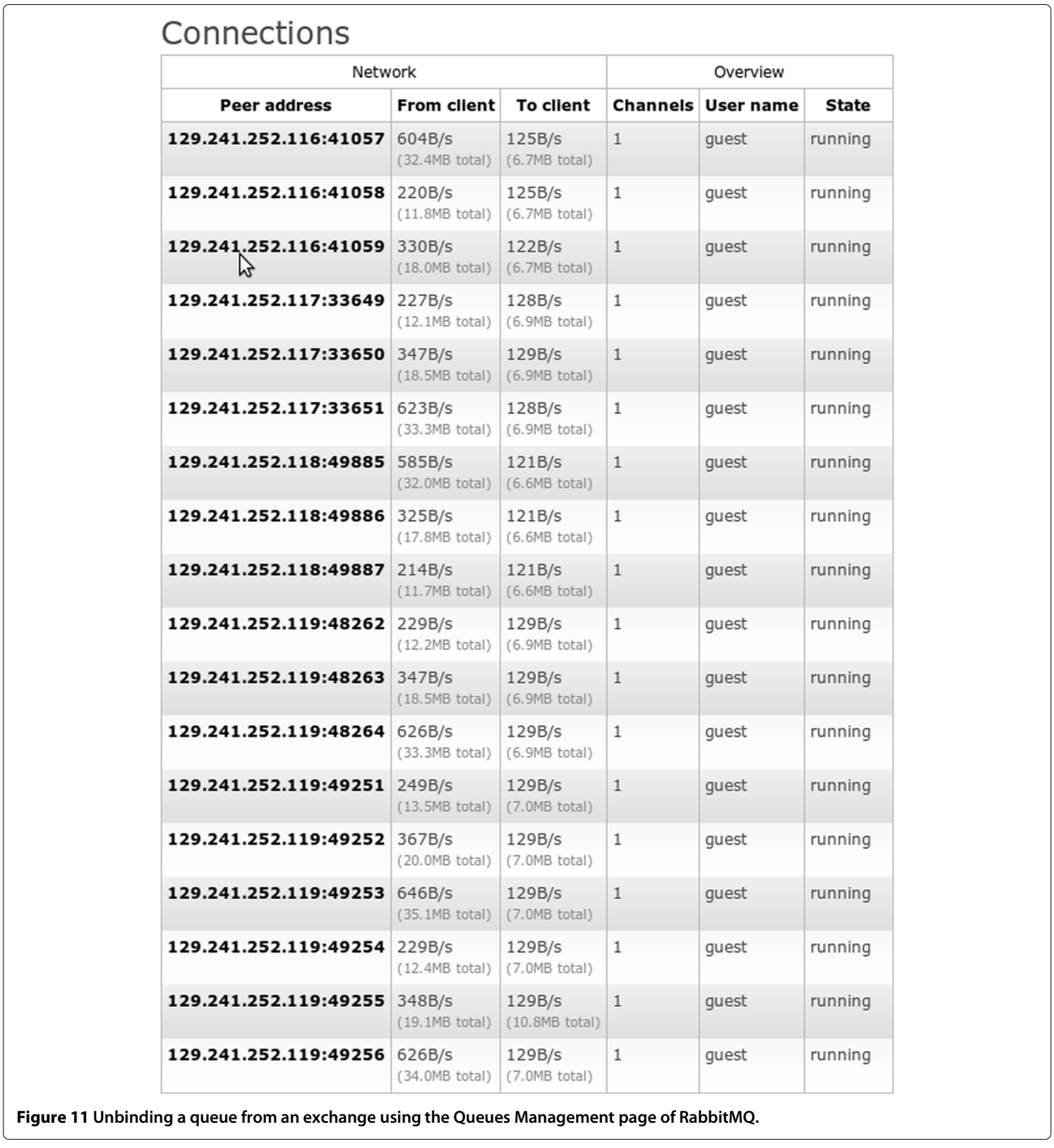


Figure 10 RabbitMQ Connections.



Connections

Network			Overview		
Peer address	**From client**	**To client**	**Channels**	**User name**	**State**
129.241.252.116:41057	604B/s (32.4MB total)	125B/s (6.7MB total)	1	guest	running
129.241.252.116:41058	220B/s (11.8MB total)	125B/s (6.7MB total)	1	guest	running
129.241.252.116:41059	330B/s (18.0MB total)	122B/s (6.7MB total)	1	guest	running
129.241.252.117:33649	227B/s (12.1MB total)	128B/s (6.9MB total)	1	guest	running
129.241.252.117:33650	347B/s (18.5MB total)	129B/s (6.9MB total)	1	guest	running
129.241.252.117:33651	623B/s (33.3MB total)	128B/s (6.9MB total)	1	guest	running
129.241.252.118:49885	585B/s (32.0MB total)	121B/s (6.6MB total)	1	guest	running
129.241.252.118:49886	325B/s (17.8MB total)	121B/s (6.6MB total)	1	guest	running
129.241.252.118:49887	214B/s (11.7MB total)	121B/s (6.6MB total)	1	guest	running
129.241.252.119:48262	229B/s (12.2MB total)	129B/s (6.9MB total)	1	guest	running
129.241.252.119:48263	347B/s (18.5MB total)	129B/s (6.9MB total)	1	guest	running
129.241.252.119:48264	626B/s (33.3MB total)	129B/s (6.9MB total)	1	guest	running
129.241.252.119:49251	249B/s (13.5MB total)	129B/s (7.0MB total)	1	guest	running
129.241.252.119:49252	367B/s (20.0MB total)	129B/s (7.0MB total)	1	guest	running
129.241.252.119:49253	646B/s (35.1MB total)	129B/s (7.0MB total)	1	guest	running
129.241.252.119:49254	229B/s (12.4MB total)	129B/s (7.0MB total)	1	guest	running
129.241.252.119:49255	348B/s (19.1MB total)	129B/s (10.8MB total)	1	guest	running
129.241.252.119:49256	626B/s (34.0MB total)	129B/s (7.0MB total)	1	guest	running

Figure 11 Unbinding a queue from an exchange using the Queues Management page of RabbitMQ.

Figure 12 depicts a simplified overview of messaging server internal entities and the application points of our approaches.

Filtering in each component

Applicable filtering mechanisms in the messaging server have been studied in the previous section. This section discusses mechanisms that are appropriate for other components. These components are not essentially aware of messaging technique details and specifications.

Advantages

- The implementation of the filtering mechanism in each component avoids added complexity to the messaging server and cloud controller.

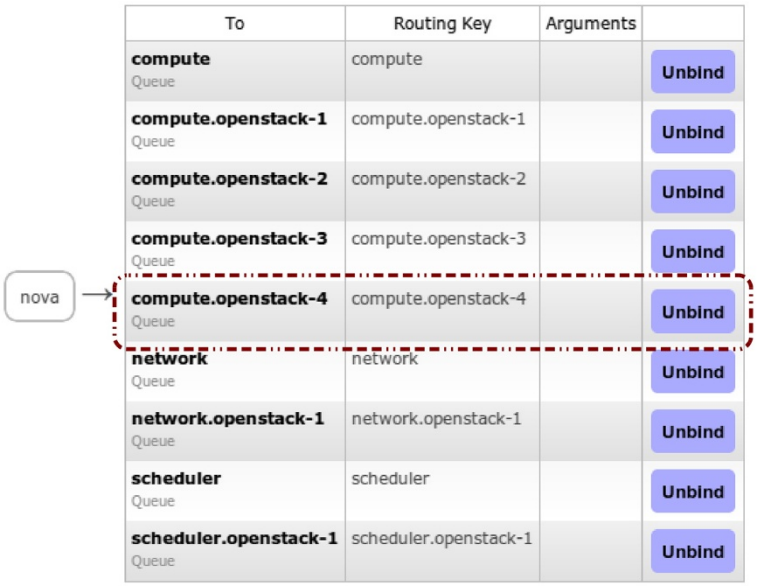


Figure 12 Overview of RabbitMQ messaging server and applicable containment approaches.

- This approach is a distributed solution without a single point of failure, in contrast to the previous one with a centralized filtering mechanism.
- Assuming the locality principle in the cloud, wasted bandwidth is limited to a cluster/rack which hosts the infected components. Network connections have much higher speed in a rack or cluster.
- This approach does not require a correlation/coordination entity for filtering messages. Each component behaves independently and autonomously upon receiving an alarm message which announces a compromised node. Traditionally, most security mechanisms have been employed at the organization/system boundaries. However, as there is no boundary in the cloud, performing security enforcement at each component is a more reliable approach.

Disadvantages

- When the filtering must be performed in each component, all interacting components must be modified to support the filtering mechanism. However, this issue can be relaxed by using a unified version of the messaging client (e.g., pika python client) and modifying the client in case of new requirements.
- The message which should be discarded traverses all the way down to the destination, and wastes the link bandwidth on its route.
- Dropping a message without notifying upper layers, may lead to triggered timeouts and resend requests from waiting entities. It can also cause more wasted bandwidth.

Realization This approach can be implemented at two different levels: blocking at either the messaging client level (e.g. AMQP messaging client) or the OpenStack component/service level.

First, the responsible client can be modified to drop messages with specific properties (e.g. infected source/destination). As an example, the responsible client for AMQP messaging in OpenStack is amqplib/pika; we must implement the mechanism in this AMQP client (or its wrapper in OpenStack) to filter malicious AMQP messages. Using this method, more interaction between OpenStack and clients may be required to avoid resend requests. Because of using the same AMQP client in all components, the implementation is easier and the modification process requires less effort. The second method is to develop filtering in each of the OpenStack components, such as nova-compute, nova-network, nova-scheduler, etc. This method adds more complexity to those components and it may not be part of their responsibilities.

We propose a combination of these methods. Implementing the filtering mechanism in the carrot/amqplib wrapper of OpenStack has advantages of both methods, and avoids unnecessary complexity. The OpenStack wrapper for managing AMQP messaging is implemented in *src/nova/rpc.py*. In order to identify the malicious message, we use the message address which is part of its context. Then, the actual dropping happens in the *AdapterConsumer* method. Assuming that the source

address is set in the context variable, filtering is straightforward. By checking the message address and avoiding the method call, most of the task is done. The only remaining part is to inform the sender about the problem, this can be implemented by means of the existing message reply functionality.

Disabling services

Disabling services is a strategy for containing the incident. The disabled service can be either the infected service itself or the communicator service. The latter handles task distribution and delegation. This method can be used only by the cloud provider, and is at the application layer.

Disabling an infected service An incident can be contained by disabling the infected service. It has several advantages, including:

- After the nova-compute service is stopped, running instances will continue to work. Thus, as a result consumers' instances will not be terminated nor disrupted.
- All communications to and from the compromised node will be stopped. So, the wasted bandwidth will be significantly reduced.
- Shutting down a service gracefully avoids an extra set of failures. When the service is stopped by the Nova interfaces, all other components will be notified and the compromised node will be removed from the list of available compute workers.

Like any other solution, it has multiple drawbacks as well, including:

- Keeping instances in a running state can threaten other cloud consumers. The attacker may gain access to running instances on the compromised node.
- The live migration feature will not work anymore. Thus, the threatened consumers cannot migrate running instances to a safe or quarantine compute worker node.
- Neither the cloud provider nor consumers can manage running instances through the OpenStack platform.

This approach requires no further implementation, although we may like to add a mechanism to turn services on and off remotely.

Disabling a communicator service An incident can be contained by disabling or modifying its corresponding communicator service. An example of a communicator service in an OpenStack deployment is the nova-scheduler service. The nova-scheduler decides which worker should handle a newly arrived request, such as running an instance. By adding new features to the scheduler service, the platform can avoid forwarding requests to the compromised node. Advantages of this approach are:

- No more requests will be forwarded to the compromised node.
- Consumers' instances remain in the running status on the compromised node. So, consumers will have enough time to migrate their instances to a quarantine worker node or dispose of their critical data.
- This approach can be used to identify the attackers, hidden system vulnerabilities, and the set of employed exploits. In other words, it can be used for forensic purposes.

Disadvantages of disabling communication include:

- New features must be implemented.
  These new features are more focused on the decision algorithm of the scheduler service.
- This approach will not secure the rest of our cloud environment, but it avoids forwarding new requests to the compromised node. However, this drawback can be seen as an opportunity. We can apply this approach and also move the compromised node to a **HoneyCloud**. In the HoneyCloud we don't restrict the compromised node, but instead analyze the attack and the attacker's behavior. But even by moving the compromised node to a HoneyCloud, hosted instances on that node are still in danger. It is possible that consumers' instances are all interconnected. Thus, those running instances on the compromised node in the HoneyCloud could threaten the rest of the consumers' instances. The rest of the instances may even be hosted on a secure worker node.
  The next proposed approach is a solution for this issue.

Replicating services

An approach to overcome the implications of an incident is replicating services. A service in this context is a service which is delivered and maintained by the cloud provider. It can be a cloud platform service (e.g nova-compute) or any other services that concern other stakeholders. The replication can be done passively or actively, and that is due to new characteristics of the cloud model. The replication of a cloud service can be done either at the physical or virtual machine layer.

Replicating a service on physical machines is already done in platforms such as OpenStack. The provider can replicate cloud services either passively or actively when facing an issue in the environment.

Replication of a service on virtual machines has multiple benefits, including:

- Virtual machines can be migrated while running (i.e. live migration), this is a practical mechanism for stateful services that use memory.
- Replication at the instance layer is helpful for forensic purposes. It is also possible to move the compromised service in conjunction with the underlying instance to a HoneyCloud. This is done instead of moving the physical node, ceasing all services on it, and changing the network configuration in order to restrict the compromised node communication.
- By using virtual machines in a cloud environment we can also benefit from the cloud model elasticity and on demand access to computing resources.

This approach is also the main idea behind the *Virtualization Intrusion Tolerance Based on Cloud Computing* (CC-VIT) [16]. By applying the CC-VIT to our environment, the preferred hybrid fault model will be Redundant Execution on Multiple Hosts (REMH), and the group communication is handle using the AMQP messaging. We can use physical-to-virtual converters to have the advantages of both approaches. These tools convert a physical machine to a virtual machine image/instance that can be run on top of a hypervisor. Moreover, each of these replicas can be either active or passive. This will have a great impact on the system availability.

Disinfecting infected components

Disinfecting an infected component is a crucial task in handling an incident and securing the system. It can be accomplished with multiple methods having a variety of specifications. None of the following approaches will be used for cleaning the infected binary files, instead less complex techniques are employed that can be applied in a highly distributed environment. Cleaning a binary file can be offered by a third party security service provider, but that will not be discussed further here.

1. **Updating the code**
   The service code can be updated to the latest, patched version. This process should be done in a smooth way so that all components will be either updated or remain compatible with each other after a partial component update. Several tools has been developed for this purpose; one of the best examples is the Puppet project [17].
2. **Purging the infected service**
   Assuming that the attacker has stopped at the cloud platform layer, we can ensure containment of the incident by removing the service completely.
3. **Replacing the service**
   Another method which is not as strong as the others is achieved by replacing the infected service with another one that uses a different set of application layer resources, such as configuration files, binaries, etc. Thus, we can be sure that the infected resources have no effect on the new service.

Isolation, disinfection, and migration of instances

In the following part, techniques which are handling virtual machine instances are discussed. Three major approaches can be chosen for handling an attacked instance: isolating, disinfecting, and migrating a given one. Each of them will be explained next.

Removing instances from the project VLAN

This approach does not contain the compromised node, instead it focuses on containing instances hosted by the compromised worker node. This is important because those instances may have been compromised as well. The first step toward securing the consumer's service is to disconnect potentially infected instances. The main usecase of this approach is when the attacker disrupts other solutions (e.g., disabling nova-compute management functionalities through escalated privileges at the OS layer), or when instances and the consumer's service security is very important (e.g., eGovernment services). It has several advantages specifically for cloud consumers, including:

- It can disconnect potentially infected instances from the rest of the consumer's instance.
- It does not require implementation of new features.
- The attacker cannot disrupt this method.

The disadvantages are as follows:

- This method only works in a specific OpenStack networking mode (i.e., the VLANManager networking mode).
- The consumer completely loses control over isolated instances, this may lead to data loss or disclosure, service unavailability, etc.

Disabling live migration

Live migration can cause widespread infection, or can be a mechanism for further intrusion to a cloud environment. It may take place intentionally or unintentionally (e.g., an affected consumer may migrate instances to resolve the attack side effects, or the attacker with consumer privileges migrates instances to use a hypervisor vulnerability and gain control over more nodes). Disabling this feature helps the cloud provider to contain the

incident more easily, and keep the rest of the environment safer.

Quarantining instances

When we migrate instances from a compromised node, we cannot accept the risk of spreading infection along instance migration. Thus, we should move them to a quarantine worker node first. The quarantine worker node has specific functionalities and tasks, including:

- This worker node limits instances' connectivity with the rest of cloud environment. As an example, only cloud management requests/responses are delivered by the quarantine host.
- It has a set of mechanisms to check instances' integrity and healthiness. These mechanisms can be provided by the underlying hypervisor, cloud platform, or third parties' services.

In order to deploy a quarantine node, a set of mechanisms should be studied and employed. Tools that implement such mechanisms will be presented below.

1. **Virtual Machine Introspection**
   This mechanism simplifies inspecting the memory space of a virtual machine from another virtual machine. The task is fairly complex because of the semantic gap between the memory space of those two virtual machines. XenAccess [18] is an example of an introspection library. Using XenAccess the privileged domain can monitor another Xen domain.
2. **Domain Monitoring**
   One of the basic methods to identify a compromised instance is by means of profiling and monitoring the instance behavior. Domain monitoring techniques provide an abstract set of data, compared to the detailed, low level output of a VM introspection tool. For a virtual machine running over a Linux box we can use the libvirt [19] library to access the suspicious instance and study its behavior.
3. **Intrusion Detection**
   Having an intrusion detection system in the hypervisor or cloud platform layer not only provide better visibility for security mechanisms but is also more resistant against a targeted attack from an unauthorized access to an instance. Livewire [20] is a prototype implementation of an intrusion detection system in a hypervisor. Another way to benefit from an intrusion detection system is Amazon's approach, which offers you a standalone Amazon Machine Image (AMI) that contains Snort and Sourcefire Vulnerability Research Team rules. The consumer can then forward its instances' traffic to the virtual machine with intrusion detection capabilities. The same approach can be utilized in our deployment. The main issue is the approach's performance and utilization.
4. **Utilizing trusted computing concepts**
   Trusted computing is a technology for ensuring the confidentiality and integrity of a computation. It is also useful for remote attestation. Thus, we can use the technology not only for securing our deployment but also to build a better quarantine and infection analysis mechanism. Approaches that have used this concept include vTPM: Virtualizing the Trusted Platform Module [21], TCCP: Trusted Cloud Computing Platform [22], and TVDc: IBM Trusted Virtual Datacenter [23].

It should be noted that although cloud providers or third-party service providers can offer an IDS agent service inside each instance, they cannot force the consumer into accepting it. It is a reasonable argument due to the consumer's organization internal security policies and resource overhead because of the security agent. Thus, applying security services to the underlying layer (i.e. hypervisor, cloud platform) is a preferred solution. Detailed specifications of such a compute worker node is a great opportunity for future work.

Recovering an instance

Recovering an infected or malfunctioning instance can be performed using different techniques. An instance can either be disinfected internally or rebooted from a clean image. However, a tight collaboration between provider and consumer is required for any of these techniques.

Obviously, disinfection of an instance cannot be performed solely by the provider, because it should not access the instance internally, but can only provide a disinfection service (e.g. instance anti-virus) to be used by the consumer at its own will. On the other hand, rebooting an instance from a clean image can be done by the provider or the consumer. Nevertheless, there are several issues in performing the reboot action. First, one must make sure that the instance termination will be done gracefully, so no data will be lost. Second, the VM image must be analyzed for any flaws or security vulnerabilities. Third, before attaching the storage to the rebooted instance, the volume must be disinfected.

Migrating instances

The affected consumer can migrate a specific instance or a set of instances to another compute worker or even another cloud environment. The migration among different providers is currently an open challenge, because of the weak interoperability of cloud systems and lack of standard interfaces for cloud services. In our

deployment, both Amazon EC2 APIs and Rackspace APIs are supported. Thus, in theory a consumer can move between any cloud environment provided by the Amazon EC2, RackSpace, and any open deployment of OpenStack without any problem.

Policies

In addition to all techniques that have been studied, a group of security policies should be developed and exercised. These policies can be implemented and enforced inside those techniques, as additional measures.

Component authentication

The component authentication policy enforces that each worker must have a certificate signed by a trusted authority. This authority can be either an external one or the cloud controller/authentication manager itself. Having a signed certificate, the worker can communicate with other components securely. The secure communication can bring us any of the following: confidentiality, integrity, authentication, and non-repudiation.

In this case, the worker's communication confidentiality and authenticity is important for us. For this purpose we can use two different schemes: message encryption or a signature scheme. Each of these schemes can be used for the whole communication or the handshake phase only. When any of those schemes are applied only to the handshake phase, any disconnection or timeout in the communication is a threat to the trust relation. As an authenticated worker is disconnected and reconnected, we cannot only rely on the worker's ID or host-name to presume it as the trusted one. Thus, the handshake phase should be repeated to ensure the authenticity of the worker. Although applying each scheme to all messages among cloud components is tolerant against disruption and disconnection, its overhead for the system and the demand for it should be studied case by case. By applying each of those schemes to all messages, we can tolerate disconnection and disruption. However, using cryptographic techniques for all messages introduce an overhead for the system which may not be efficient or acceptable.

Implementing this method in our environment is simple. The RabbitMQ has features that facilitate communication encryption and client authentication. The *RabbitMQ SSL support* offers encrypted communication [24]. Moreover, an authentication mechanism using the client SSL certificate is offered by the *rabbitmq-auth-mechanism-ssl* plugin [25].

No new worker policy

In addition to the previously discussed technical approaches, a set of management policies can also relax the issue. As an example, no new worker should be added unless there is a demand for it. The demand for a new worker can be determined when the resource utilization for each zone is above a given threshold.

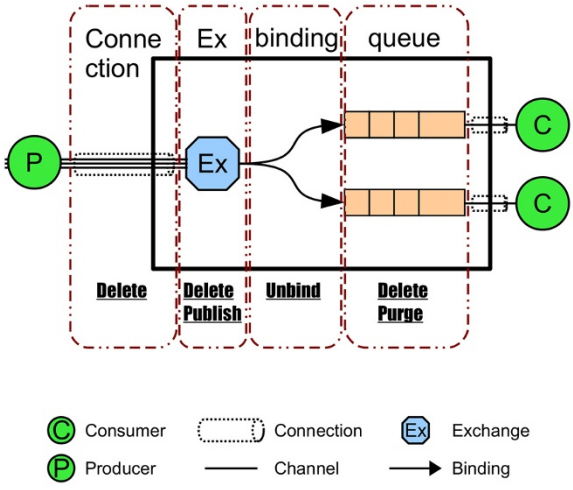


Figure 13 A simple Finite State Machine (FSM) model for trust states of a component.

Trust levels and timeouts

Introducing a set of trust levels, a new worker can be labeled as a not trusted worker. Workers which are not trusted yet, can be used for hosting non-critical instances, or can offer a cheaper service to consumers. In order to ensure the system trustworthiness in a long run, a not-trusted worker will be disabled after a timeout. A simple Finite State Machine (FSM) model of those transitions is depicted in Figure 13.

Assuming we have only two trust levels, Figure 14 depicts transitions between them. As an example, *T0* can be achieved by human intervention; and the second level of trust *T1* is gained by cryptographic techniques or trusted computing mechanisms.

This policy can be implemented in the cloud platform scheduler (e.g. nova-scheduler is the responsible component in the OpenStack platform). Implementing

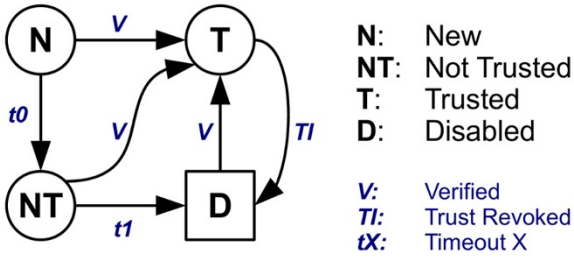


Figure 14 A simple FSM model for transitions between different trust levels of a component.

Table 4 Comparison (RS: Responsible stakeholder, CP: Cloud Provider, CC: Cloud Consumer, P: Proactive, R: Reactive)

Approach	RS	P/R	Service impact	Implementation/Enforcement difficulties	Dependencies
Filtering in the messaging server	CP	R	Platform components may never receive an expected message.	Unless deployed in distributed mode, can become a bottleneck.	Messaging server
Filtering in each component	CP	R	Platform components may never receive an expected message.	All components should be modified to support it.	Platform components
Disabling services	CP	R	Healthy components can become inaccessible. Losing control over instances managed by disabled components.	-	Platform interfaces
Replicating services	CP	P	Services should be replicated based on requirement and performance analysis of the environment.	-	Platform components
Disinfecting infected components	CP	R	Healthy components can become inaccessible. Losing control over instances managed by disabled components.	Configuration management tools and cloud platform interfaces should be deployed and configured.	Configuration management tools, Platform interfaces
Removing instances from the project VLAN	CP CC	R	The instance won't be accessible for the consumer and its services.	Highly dependent on the OpenStack VLANManager networking mode.	Platform components
Disabling live migration	CP	R	Consumer experiences lower QoS.	-	Platform interfaces
Quarantining instances	CP CC	R	Quarantined instances won't be accessible for the consumer.	Implementing this solution requires a lot of effort as discussed briefly in [10].	-
Disinfecting an instance	CP CC	R	-	A framework for analyzing VM images and disinfecting running instances must be developed	Platform interfaces
Migrating instances	CP CC	R	Consumer may experience lower QoS.	The cloud environment should consist of distributed and independent zones.	Platform interfaces
Component authentication	CP	P	Small overhead for all communications.	Developing a system for managing components certificates and identity.	Messaging server and identity services
No new worker policy	CP	P	-	Developing a policy manager component	Messaging server and policy manager
Trust levels and timeouts	CP CC	P	Lower QoS for non-critical use-cases Lower resource volume for critical use-cases	High complexity	Platform interfaces, and scheduler component

this policy will allow the cloud provider to offer more resources for non-critical use-cases. However, the offered QoS might not be as good as before. The effectiveness of this approach is highly dependent on a few parameters, such as the ratio of adding new workers, trust mechanisms and their performance, and consumers' use-cases and requested QoS.

A major challenge in this approach is about trust mechanisms. The simplest mechanisms will be manual determination and confirmation of authenticity and trust level. A recently added worker won't be used for serving consumers' requests until its authenticity is confirmed by the relevant authority (e.g. cloud provider). However, this does not scale: The human intervention can simply become a bottleneck in the system.

Comparison

A list of security mechanisms have been discussed. Although most of them are orthogonal to each other, they can be compared in terms of their common criteria (Table 4). A few criteria are extracted and explained in the following part.

- **Responsible stakeholder**: Each mechanism must be delivered by a single stakeholder or a group of stakeholders. Identifying those responsibilities and assigning them to the right bodies is a crucial step toward building a secure environment.
- **Proactive/Reactive**: Both proactive and reactive mechanisms have been discussed above. Knowing mechanisms' behavior is useful in their enforcement and comparison.
- **Service impact (Affected entities)**: Inevitably, enforcing each mechanism will introduce a set of side effects to delivered services and working entities. Identifying these side effects makes the enforcement process much more predictable.
- **Implementation/Enforcement difficulties**: Finding out implementation challenges of security mechanisms is important. These challenges are meaningful measures in comparing mechanisms with each other.
- **Dependencies**: Dependencies of approaches make them bounded to a specific platform and libraries. Having less or looser dependencies makes the solution more portable. Portable approaches can be developed as generic services that can be applied to a variety of platforms. Thus, we can see the importance of having common and standard interfaces among different platforms, such as Open Cloud Computing Interface (OCCI) and Cloud Data Management Interface (CDMI).

Conclusion

Cloud computing is a new computing model, whose definitions and realizations have new characteristics compared to other computing models. New characteristics hinder the application of existing mechanisms. In some cases, existing approaches are not applicable, and in other cases adaptation is required. Initially, we studied different aspects of a real cloud environment, working on a deployed environment instead of focusing on an imaginary computing model. Experimenting on a deployed environment is helpful in reducing the gap between academic research and industrial deployment/requirements. Many questions that are discussed in an academic environment are already solved in industry, or are not the right questions at all. A good blog post on this issue can be found in [26].

Although our lab setup was not big enough to be industry realistic, it was useful for understanding the ecosystem of the cloud model, and observing possible weaknesses in it. Obviously, deploying a larger infrastructure reveals more information about the exact behavior of the environment, and the result will be more accurate. However, that may not be feasible as a university project unless big players in the cloud are willing to contribute, as can be seen in efforts such as OpenCirrus [27] (supported by HP, Intel, and Yahoo!), the Google Exacycle [28] program, and Amazon grants for educators, researchers and students [29].

In our study we have decided to use the OpenStack cloud software. There were multiple reasons behind this decision, such as:

- Working on an open source project helps the community, and pushes the open source paradigm forward.
- Analysis of the platform and experimenting with different approaches is easier and more efficient when we can access the source code.
- Big companies are involved in the OpenStack project, and many of them are using the platform in their own infrastructure. Thus, OpenStack can become a leading open source cloud platform in the near future.

This study was started only 4 months after the first release of OpenStack, and much of the required documentation was either not available or not good enough. We studied the OpenStack components and identified their functionalities and other specifications. Moreover, working with a platform which is under heavy development, has its own challenges.

In order to secure the environment against a compromised component, we have to handle the corresponding incident. The NIST incident handling guideline has been studied and applied to our experimental cloud

environment. During the application process we did not limit ourselves to the lab setup, because it was not large/distributed enough. So, in the proposed approaches we considered a large scale, highly distributed target environment; and made those approaches compatible with such an environment. Moreover, the NIST guideline recommends a set of actions for each handling phase. These actions can be realized using a variety of mechanisms. We have studied several mechanisms and discussed their compatibilities with the cloud model. Additionally, we have proposed new approaches that are helpful in fulfilling incident handling requirements. Furthermore, in this process multiple questions and challenges were raised that can be interesting topics for future work in cloud incident handling and in general security of a cloud environment. We itemize a few of them in the following:

- Statistical measurement and analysis of each approach and study of the exact performance overhead.
- Large scale deployment of OpenStack with its latest release.
- Implementation of proposed approaches as a set of security services, and study their effectiveness for a cloud consumer and the cloud environment in general.
- Study the compatibility of approaches and guidelines to other cloud environments, specifically with those operated by industry or commercial cloud providers (e.g. Amazon, Rackspace, Google App Engine, Azure).

Endnotes

[a] REpresentational State Transfer

[b] Avoiding false positive alarms

[c] By the term *organization*, we mean all entities who are responsible for managing the cloud infrastructure, which can be referred to as the cloud provider

[d] In a publisher/subscriber paradigm the destination may be eliminated or masked by other parameters. So, we may filter messages that contain any evidence of being related to the infected host

[e] It should be noted, although we may use directory and federation services to unify users among services and layers, this may not be a feasible approach in a cloud environment. However, federation is applicable at each layer (e.g. system, cloud platform, VM instances)

[f] Scaling out or horizontal scaling is referred to the application deployment on multiple servers [30]

[g] But it may require a correlation entity to handle the filtering tasks among all messaging servers

Competing interests
The authors declare that they have no competing interests.

Author's contributions
ATM performed the configuration and testing of the OpenStack lab environment, and drafted the paper. MGJ supervised the practical work, and contributed to the writing to improve the quality of the text. All authors read and approved the final document.

Acknowledgements
This article is based on results from MSc Thesis work performed at Norwegian University of Science and Technology (NTNU).

Author details
[1] UNINETT, Trondheim, Norway. [2] SINTEF ICT, Trondheim, Norway.

Received: 3 February 2012 Accepted: 2 July 2012
Published: 16 August 2012

References

1. McCarthy J (1999) MIT Centennial Speech of 1961 cited in Architects of the Information Society: Thirty-five Years of the Laboratory for Computer Science at MIT. S.L. Garfinkel Ed. MIT Press, Cambridge MA
2. Mell P, Grance T (2011) The NIST Definition of Cloud Computing, Technical Report SP 800-145. National Institute of Standards and Technology, Information Technology Laboratory
3. Chen Y, Paxson V, Katz RH What's New About Cloud Computing Security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley 2010. http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-5.html
4. TaheriMonfared A, Jaatun MG (2011) As Strong as the Weakest Link: Handling compromised compoenents in OpenStack. In: Proceedings of the third IEEE International Conference on Cloud Computing Technology and Science (CloudCom)
5. OpenStack Community (2011) OpenStack Projects page. http://openstack.org/projects/
6. Walsh S (2011) Multiple Cluster Zones. http://wiki.openstack.org/MultiClusterZones
7. Scarfone K, Grance T, Masone K (2008) Computer Security Incident Handling Guide. Special Publications SP 800-61 Rev. 1, NIST. http://csrc.nist.gov/publications/nistpubs/800-61-rev1/SP800-61rev1.pdf
8. TaheriMonfared A, Jaatun MG (2011) Monitoring Intrusions and Security Breaches in Highly Distributed Cloud Environments. In: Proceedings of CloudCom 2011
9. AWS Security Team (2011) Vulnerability Reporting. http://aws.amazon.com/security/vulnerability-reporting/
10. TaheriMonfared A (2011) Securing the IaaS Service Model of Cloud Computing Against Compromised Components. MSc thesis, Norwegian University of Science and Technology (NTNU)
11. Jokela P, Zahemszky A, Esteve Rothenberg C, Arianfar S, Nikander P (2009) LIPSIN: line speed publish/subscribe inter-networking. In: Proceedings of the ACM SIGCOMM 2009 conference on Data communication, SIGCOMM '09. ACM, New York, pp 195–206. http://doi.acm.org/10.1145/1592568.1592592
12. Broder A, Mitzenmacher M (2002) Network Applications of Bloom Filters: A Survey. In: Internet Mathematics. pp 636–646
13. RabbitMQ Core API Guide (2011) http://www.rabbitmq.com/api-guide.html
14. Trieloff C, McHale C, Sim G, Piskiel H, O'Hara J, Brome J, van der Riet K, Atwell M, Lucina M, Hintjens P, Greig R, Joyce S, Shrivastava S (2006) Advanced Message Queuing Protocol Protocol Specification. amq-spec, AMQP.org. [Version 0.8]
15. Samovskiy D (2008) Introduction to AMQP Messaging with RabbitMQ
16. Tan Y, Luo D, Wang J (2010) CC-VIT: Virtualization Intrusion Tolerance Based on Cloud Computing. In: Information Engineering and Computer Science (ICIECS), 2010 2nd International Conference on. pp 1–6
17. Puppet Labs (2011) http://www.puppetlabs.com/
18. XenAccess (2009) http://www.xenaccess.org/
19. libvirt Wiki (2011) http://wiki.libvirt.org/page/Main_Page#libvirt_Wiki
20. Garfinkel T, Rosenblum M (2003) A Virtual Machine Introspection Based Architecture for Intrusion Detection. In: Proc. Network and Distributed Systems Security Symposium

21. Berger S, Cáceres R, Goldman KA, Perez R, Sailer R, van Doorn L (2006) vTPM: Virtualizing the Trusted Platform Module. Research Report RC23879, IBM Research Division
22. Santos N, Gummadi KP, Rodrigues R (2009) Towards Trusted Cloud Computing. In: HOTCLOUD, USENIX
23. Berger S, Cáceres R, Pendarakis D, Sailer R, Valdez E, Perez R, Schildhauer W, Srinivasan D (2007) TVDc: Managing Security in the Trusted Virtual Datacenter. Research Report RC24441, IBM Research Division
24. RabbitMQ SSL (2011) http://www.rabbitmq.com/ssl.html
25. MacMullen S (2011) Who are you? Authentication and authorisation in RabbitMQ. http://www.rabbitmq.com/blog/2011/02/07/who-are-you-authentication-and-authorisation-in-rabbitmq-231/
26. Welsh M (2011) How can academics do research on cloud computing? http://matt-welsh.blogspot.com/2011/05/how-can-academics-do-research-on-cloud.html
27. Open Cirrus (2011) https://opencirrus.org/
28. Belov D (2011) 1 billion core-hours of computational capacity for researchers. http://googleresearch.blogspot.com/2011/04/1-billion-core-hours-of-computational.html
29. AWS in Education (2011) http://aws.amazon.com/education/
30. Michael M, Moreira J, Shiloach D, Wisniewski R (2007) Scale-up x Scale-out: A Case Study using Nutch/Lucene. In: Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International. pp 1–8

doi:10.1186/2192-113X-1-16
Cite this article as: TaheriMonfared and Jaatun: **Handling compromised components in an IaaS cloud installation.** *Journal of Cloud Computing: Advances, Systems and Applications* 2012 **1**:16.