Toward Agent Based Inter-VM Traffic Authentication in a Cloud Environment

Benzidane Karim(✉), Saad Khoudali, and Abderrahim Sekkaki

Faculty of Sciences Ain Chock, University Hassan II, Casablanca, Morocco
k.benzidane@live.fr, {s.khoudali,a_sekkaki}@yahoo.fr

Abstract. Ubiquitous simply means being everywhere. The concept of Cloud Computing (CC) further strengthens the idea of Ubiquitous computing. On the other hand, one of the key enablers of CC is Virtualization. However, with the many advantages of virtualization comes certain limitations, especially related to security. Virtualization vulnerabilities and more specifically isolation, creates new targets for intrusion due to the complexity of access and difficulty in monitoring all interconnection points between systems, applications, and data sets. Hence, without strict control put in place within the Cloud, guests could violate and bypass security policies, intercept unauthorized client data, and initiate or become the target of security attacks. This article discusses the security and the visibility issues of inter-VM traffic, by proposing a solution for it within the Cloud context. The proposed approach provides Virtual Machines (VMs) authentication, communication integrity, and enforces trusted transactions, through security mechanisms, structures, policies, and various intrusion detection techniques.

Keywords: Ubiquitous computing · Cloud computing · Virtualization · Security · Intrusion detection

1 Introduction

Ubiquitous computing is considered as a big wave of computing after mainframe and CC, where many computers serve one person. CC is an enabler of ubiquitous computing, giving the user an interaction with its resources at anytime and from anywhere, informational retrieval, and multimodal interaction with the user and enhanced visualization capabilities. In effect, giving extremely new and futuristic abilities for users to look at and interact with at any time and from anywhere. Such ubiquity of cloud resources hold the promise of enabling new service models, where resources are seamlessly utilized at the time and location that are best suited to the needs of the current workload, while at the same time optimizing business objectives such as minimizing cost and maintaining service quality levels [1]. CC has generated interest from both industry and academia over these years. As an extension of Grid Computing and Distributed Computing, CC aims to provide users with flexible services in a transparent manner. Though Providing security in a distributed system requires more than user authentication with

I. Stojmenovic et al. (Eds.): MOBIQUITOUS 2013, LNICST 131, pp. 537–548, 2014.
DOI: 10.1007/978-3-319-11569-6_42

passwords or digital certificates and confidentiality in data transmission, the distributed model of CC makes it vulnerable and prone to sophisticated distributed and internal attacks. Since virtualization is the fundamental of the CC, offering higher machine efficiency due to increased utilization, energy savings, and the flexibility to build or destroy virtual machines (VMs) on demand to meet changing organizational needs, those advantages comes with some concerning security issues especially in the cloud context. In this research, we are focusing our work on inter-VM threats and VM hopping by proposing an authentication mechanism of each request communicating between VMs, and rejecting the non-compliant ones. Inter-VM threats occur when a hacker takes control of a VM and then gains access to the underlying hypervisor, allowing him to access and control all of the VMs on the system via VM hopping.

The remainder of the article is structured as follows: in the next section, we discuss some virtualization security issues in CC. The third section presents the core of our approach by explaining its components. In Sect. 4, we present the IP packet handling and processing by highlighting the use of the frame tag and the agents in a CC environment. The last section contains a summary and conclusions.

2 Background

One of the outstanding properties of virtualization is its ability to isolate co-resident Operating Systems (OSs) on the same physical platform. While isolation is an important property from a security perspective, co-resident virtual machines (VMs) often need to communicate and exchange a considerable amount of data. Multi-tenant infrastructures typically offer scaled performance and services based on shared resources, including databases, other applications, and OSs. For some organizations, this leaves them open to a variety of threats in a virtualized environment [2,3], these fall into a number of categories, centered mostly on the hypervisor: Hyperjacking, VM escape, VM hopping, VM migration, VM theft and inter-VM traffic.

Hyperjacking: is subverting the hypervisor or injecting a rogue hypervisor on top of the hardware layer. Since hypervisors run at the most privileged ring level on a processor, it would be hard or even impossible for any OS running on the hypervisor to detect it. In theory, a hacker with control of the hypervisor could control any virtual machine running on the physical server.

VM Hopping/Guest jumping: is the process of hopping from one VM to another VM using vulnerabilities in either the virtual infrastructure or a hypervisor. These attacks are often accomplished once an attacker has gained access to a low-value, thus less secure, VM on the same host, which is then used as a launch point for further attacks on the system. Because there are several VMs running on the same machine, there would be several victims of the VM hopping attack. An attacker can falsify the SaaS users data once he gains access to a targeted VM by VM hopping, endangering the confidentiality and

integrity of SaaS. VM hopping is a considerable threat because several VMs can run on the same host making them all targets for the attacker.

VM mobility/migration: is enabling the moving or copying of VMs from one host to another over the network or by using portable storage devices without physically stealing or snatching a hard drive. It could lead to security problems such as spread of vulnerable configurations. The severity of the attack ranges from leaking sensitive information to completely compromising the OS. A man-in-the-middle can sniff sensitive data, manipulate services, and possibly even inject a rootkit. As IaaS lets users create computing platforms by importing a customized VM image into the infrastructure service. The impact on confidentiality, integrity, and availability via the VM mobility feature is quite large.

Inter-VM: In a traditional IT environment, network traffic can be monitored, inspected and filtered using a range of server security systems to try to detect malicious activity. But the problem with virtualized environments provides limited visibility to inter-VM traffic flows. This traffic is not visible to traditional network-based security protection devices, such as the network-based intrusion prevention systems (IPSs) located in network, and cannot be monitored in the normal way.

3 Our Approach

Virtualization is increasingly used in portions of the back-end of Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and SaaS (Software as a Service) providers. The benefits of virtualization are well known, including multi-tenancy, better server utilization, and data center consolidation. Cloud Providers can achieve higher density, which translates to better margins and enterprises, can use virtualization to shrink capital expenditure on server hardware as well as increase operational efficiency. But as mentioned in the introduction, a malicious user has a huge hack domain since the inter-VM communication is blind to the security appliances on the LAN, giving him the possibility to take control of other VMs via a compromised one. Thereby, the focus of our work is about controlling this particular inter-VM traffic, analyzing it, and then preventing the non-compliant one.

Relating to other works especially in intrusion detection [4,5], we find that all of the measures that should be taken must be distributed since it is a Cloud environment [6], but the work done so far is about detecting and preventing a malicious traffic [7]. Our approach aims to control and analyze a particular traffic which is the inter-VM traffic by introducing a security structure characterize by a frame called frame tag added via an agent in the payload of the IP packet ensuring a high-level of integrity, so that the receiving VM's agent will detect, analyze and authenticate the incoming traffic then respond by accepting or rejecting this IP packet according to the compliance of the information on the frame tag. A compromised VM can be a jumping-off point in order to send requests to other VMs, for instance retrieving information in a malicious way

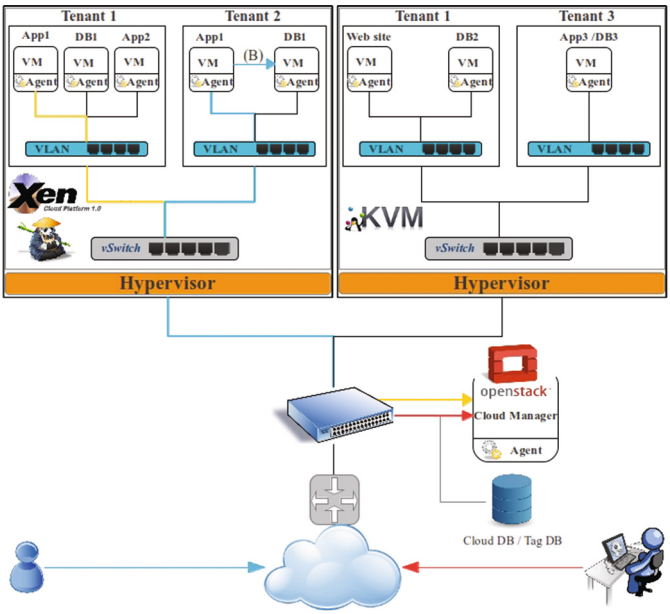

Fig. 1. A simple model for a Cloud Service Provider (Color figure online).

from a VM hosting a database, while in a legitimate scenario as depicted in Fig. 1, App1 hosted in a VM in tenant 1 sends a request to the DB1 hosted in another VM in order to get the requested information. Thereby, we are having two distinctive elements which are the tenant and the application. Hence, what we are trying to achieve with our approach is to authenticate each request communicating between VMs by encapsulating these two identifiers in the frame tag which will be an authentication factor for the IP packets by the sending VM' agent, then detected and analyzed by the receiving VM' agent and rejecting the non-compliant packets.

With this chapter, we provide a high level description of our proposed architecture, the frame tag, the security contract, policy management, how the agent works, and how they fit together. The goal of this description is to have a clear view of the overall approach, see how it fits into a Cloud environment, and to provide context for later chapter, describing a more detailed scenarios of packet handling and processing.

3.1 Frame Tag

Our approach is about identifying the application that sends the request (IP packet) to another machine in the same tenant. Hence, we propose a new structure of a frame called frame tag. The frame tag is generated between a pair of communicating agents, and it contains two fields: Tenant tag, and Application Tag.

The Tenant Tag Field: When a tenant is created an identity is assigned becoming the tenant tag and stored in a database. When a request is sent from a VM to another VM within the same tenant, the frame tag is generated and added by the agent in the payload of the IP packet and foreseeably the tenant tag, thereby ensuring the prevention of springboarding to another VM within another tenant, therefore adding a strong layer of tenant isolation.

The Application Tag Field: When an application is added and installed on a VM, the client has to certify the trustworthiness of this application by adding it via the console management, and then an ID is created to be the application tag and stored in a database. The use of the application tag in our approach, is that when App1 in Tenant1 sends a request to DB1 hosted in another machine within the same tenant and if the user did certify this application (App1) as a trustworthy one, the sending VM' agent will generate the frame tag in order to send this request by filling its fields with the right tenant tag and application tag.

3.2 Policy Management

In general, a Policy-based management (PM) is an administrative approach that is used to simplify the management of a given endeavor by establishing policies to deal with situations that are likely to occur [11].

In our case, the PM is the module responsible for handling outbound/inbound IP packets, in order to process them accordingly (such as processing the frame tag or the SC, inbound packets...). The output of the policy will be one of two actions -Discard, or Allow-. The protection afforded by the agents is defined by a database called the PM Database (PMD). This module will interact with the agent to update PMD and defines the processing of each outbound or inbound packet either needing a security service, and accepting, or dropping the packet.

IP packets are queued until being checked against the PM. If the policy indicates a packet needing to be dropped, then the inbound IP packet is expected to be discarded. When a packet is discarded, a reject policy is made so that the incoming packets from that host are dropped, because the host is considered compromised and a notification is made to the administrator. But if the policy indicates inbound/outbound inter-VM traffic, a special security processing is required. (Will be detailed in the next section)

3.3 Security Contract

The security and the integrity of the frame tag are critical. Hence, we introduced what we call a Security Contract (SC) (analogous to IPSec [8–10]). An SC is a uni-directional connection that affords security services to the frame tag carried by it. It is the establishment of a mutually agreed-upon security mechanisms and attributes (encryption algorithm, hash algorithms...) between two communicating VMs/agents to support a secure communication. To secure typical, bi-directional communication between two agents, a pair of SCs (one in

each direction) is required. If two VMs, A and B, are communicating, then the host A will have an SC, SC-A, for processing its inbound packets. The host B will also create an SC, SC-B, for processing its inbound packets. Data sets associated with an SC are represented in the SC Database (SCD) and shared between agents. Once an SC is created, it is added to the SCD and identified by a Security Contract ID (SC-ID) defining its encryption algorithm (including key length), hash algorithms, lifetime, and the quick mode status (meaning no encryption is being performed).

Security Contract ID: Is a field on the SCD representing a unique value generated by the sender and used by a receiver to identify the SC from the SCD to which outbound packets should be bound.

Lifetime: Is a lifetime value of the SC to stay operational. The lifetime is divided into three fields in the SCD; key regeneration lifetime, busy session lifetime and idle session lifetime. Key regeneration lifetime is the key lifetime used in an SC. Whenever a key lifetime is reached, the SC is updated with the new regenerated key. Busy and idle session lifetimes are used to determine the lifetime of a session before the current used SC is expired and renegotiating another one.

During the negotiation of an SC, the VM sends three important fields; the SCI, the frame tag and a session token. The session token is a hash of the used SCI, frame tag and a time stamp. It is only valid till the session lifetime expires, and it is used as an inner tail on the payload in order to be authenticated during the communication.

In order to orchestrate those SCs and PM in a homogeneous way, each VM is embedded with an agent responsible for handling and processing them.

3.4 The Agent

The agents include similarities for minimal and lighter IDPS or firewall functionality, since it is an essential aspect of access control at the IP layer, they are also responsible of generating all the required fields and applying the PM. The agents behavior is depending on whether if they are receiving/analyzing or sending/generating a request. As IP packets flow into a VM, the agent doesn't perform a whole packet analysis, but only targets specific fields of the IP packets and responds by an automated acceptance or rejection according to the PM.

Every tenant in a Cloud environment has some critical machines and resources with a restricted access. For this purpose, the agent has an attribute within its PM called trust level and can take two distinct values; high and low, and it comes to the administrator to define the trust level of his machines/agents (for example a database with sensitive data can take a high trust level), this value will affect the behavior of the agent regarding the frame tag in terms of the inspection. There are two types of agents; master/manager and slave/regular agents. When an agent is freshly deployed it is assigned to a prior known master agent for management purposes. At the administrator side, the master agent

plays the role of a front-end for the tenant in order to manage the deployed agents in terms of setting up their trust level, and to certify the trustworthiness of the applications running on the machines within the tenant, but also managing the SCs and the policies. Assigning the slave agents to their master agent means also having the right key pair in order to have encrypted and secured transactions between them, therefore preventing rogue requests to the master or having some VMs impersonating the master. The key pair could be changing periodically according to the administrators configuration. When a change of the key pair occurs the master agent sends a request to all his slave agents in order to update their keys.

As any intrusion detection and prevention system [12], the agent performs it analysis according to a set of rules in its PM, but in our case the agent has a self-generated rules defining what action should be taken regarding to its inbound IP packets. Besides the trust level component, the agent has another component called rules engine which is responsible for the self-generated rules based on the trust level value. The first task of the rules engine is to fetch the tenant tag and application tags, afterwards the rules engine checks the value of the trust level from the PMD in order to generate the proper rules set. When the trust level value is low, the generated rule will be is only about analyzing the tenant tag of the frame tag. On the other hand, if the trust level value is high this means that the generated rule is about analyzing both tenant tag and application tag of the frame tag (3).

In order to have a more accurate agent' behavior and targeted analysis, we introduced an inner header called a flag. The flag field is an indicator added by the sending agent at the beginning of the payload and it determines what action should be taken by the receiving agent depending on its value. The flag is a set of bits defining the type of data carried within the payload and can take several values: SC initiation flag, SC update flag, PM update flag, Data flag, Subscription flag, Rules and keys Update flag, and Trust level flag. The flag will be discussed more explicitly in later sections.

At the first run of a slave agent, it sends a request to its assigned master agent in order to retrieve its related tenant information. A tenant can contain several machines and accordingly several agents. Hence, a freshly deployed slave agent needs to recognize its neighboring agents within the same tenant, therefore comes the role of the flag field. When a slave agent is newly deployed, it sends a request to its master in order to subscribe its self and receive its tenant tag, applications tag and also information about its neighboring agents (IP address and trust level) and also populating the SCD with the SCs created so far. After this, the master agent sends specific frame tag with a particular flag called subscription flag to all the slave agents within the same tenant in order to have synchronization between the slave agents (the IP address and the trust level of the new slave agent). The trust level of a new slave agent is set by default to low but evidently it can be changed via the console management on the master agent by the admin if needed. When a change of a trust level occurs, the master agent sends a trigger to the designated slave agent in order to update its trust

level. The request holds a particular value on the flag field called trust level update flag, so that the receiving agent will update its trust level and triggers an update of its rules set meaning a re-execution of the rules engine workflow. As for the rule update flag is triggered, when the admin at the console management adds an application as a trusted one, the master agent updates its database and sends a request with a rule update flag to all the agents within the same tenant in order to add the new application in their rules set and create a rule for it.

The implementation of distributed agents comes to relieve the pressure of hypervisor. Hence, we will have an improved resource management, and also no direct impact on the scalability and the performance factors, since that the agents are embedded in the VMs, and it comes to the Cloud manager to provision VMs on the fly with the needed resources. The role of the agents is to elevate the level of security in a Cloud environment by adding another layer of authentication via the frame tags. It acts like a light weight security appliance or an IDPS, more specifically like a Stack based IDPS, where the packets are examined as they go through the TCP/IP stack. The protection offered by this approach is based on requirements defined by the security policy established and maintained by an administrator.

Most OS kernels enable extensibility by exporting a standard generic link layer device interface. This allows us to make link-layer drivers a non-intrusive loadable kernel module. Thus, our approach should be designed to be deployed without patching to the guest OSes or the VMM sources. Also, the integrity of those agents is primal, hence operating wise, they are executed in a privileged system domain, (Ring 0 or Ring 1) so that the agents would be protected from being controlled by any malicious application running within the VM or a unauthorized users, and also in order to avoid any kind of impersonation.

This approach is an end-to-end security scheme designed to provide interoperable and high quality security for the inter-VM traffic. Generally, it is not affected by fragmentation, because the IP packets are fragmented after the agent' processing and the fragments are reassembled before the IP layer invokes the analysis process for further processing. On inbound processing, the agent will only get a reassembled packet from the IP layer. The set of security services offered includes integrity, data origin authentication, detection and rejection of none compliant packets. These services are provided at the IP layer, offering protection in a standard fashion for all protocols in the TCP/IP suite, and due to its transparency to applications, its implementation does not need to be configured separately for each application that uses TCP/IP.

4 IP Packet Handling and Processing

Relating to the model depicted in Fig. 1 that we are basing our work on, and after laying out the components of our approach, this section provides a working scenarios description of how our proposed approach works, to provide the security services noted above. The goal of this description is to be able to "picture" the overall process and architecture, see how it fits into a Cloud environment, and how the traffic is processed by the agents and how it is handled.

Most of the Cloud service providers have OS images and templates ready to use. According to our approach, the agents need to be already embedded in order to have a working scenario. When a client chooses an image or a template to run, the agent is in a larval mode meaning that it has to do specific tasks in order to be up and running. We should bear in mind that the OS images contain primary default encryption keys which will be used to initiate new agents for the communication with its master. Those primary encryption keys are updated and managed by the admin. At its first run, the agent knows only five things, its IP address, the IP address of its master, its trust level which is set by default to low, the key pair for the encrypted communication, and a default rule to accept incoming frame tags from its master. Thereby, the first thing that the agent does (Fig. 1, 1) is sending a request to its master agent in order to subscribe its self as a new slave agent in order to receive its tenant tag, the applications tags within the tenant, but also the IP addresses of its neighboring agents, and the shared SCD. Then, the new slave agent will start self-generating its rules set and change to listening mode. After this phase, comes the subscription phase of the new slave agent alongside the neighboring agents, this task is done by the master agent by sending the subscription flag having as data the IP address of the new slave agent. The receiving agents will have to begin at first, the analysis process by making sure that the incoming traffic is either from the master agent or another VM/agent within the same tenant (since that every slave agent knows its master and neighboring agents) in order to begin the packet inspection process by capturing the IP Packet and going directly to the first bits of the payload to check the inner header which is flag in order to know the nature of the request, which is for subscription purposes in this case.

The master also handles the management of the SCs. For instance, when the lifetime of the SC expires, it cannot be used anymore and another SC has to be renegotiated between the communicating VMs. In order to avoid the problem of breaking the communication when the SC expires, the agent warns its master that the SC is about to expire allowing it to notify the admin for a redefinition of this SC before a hard deletion. When the administrator updates the SC, the master sends the updated SC along with the SC update flag to all the agents of the same tenant in order to update the shared SCD. The same thing goes for a newly created SC.

The PM database is empty by default and only populated when a new policy entry is created, by sending a PM update flag from the master for the appropriate agent.

As aforementioned, the trust level at the first run of a slave agent is set by default to low, but when a change of this value occurs (Fig. 1, red) on the console management, the master agent sends a trust level update flag to the designated agent in order to change its trust level and accordingly its whole rules set, but also a subscription frame tag flag with the IP address and the new trust level of the selected slave agent to all its neighboring agents in order to update trust level value. The same thing happens when an application is added as a trusted one within the tenant (Fig. 1, yellow), the master agent sends a rule update flag

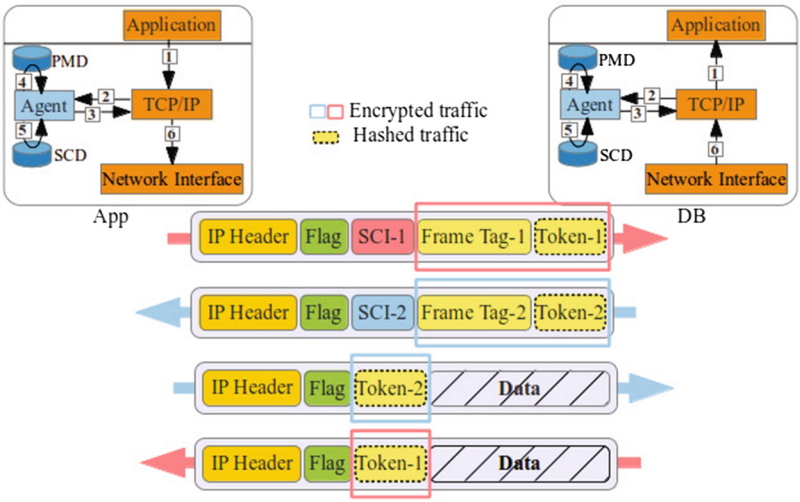

Fig. 2. The workflow of IP packet processing.

to all the slave agents in order to update their rules set by generating a rule for this application according to their trust level.

For a daily usage in a Cloud environment, a user can for example consult an application called App1 in tenant 2 (Fig. 1, blue), the incoming traffic from the user is considered an external traffic therefor the agent will not process it (according to its PM), but when the user tries for example to sign in by entering his login and password, evidently the App 1 will send a request to a database DB 1 in order to check the credentials. Let us consider the example network diagram shown in Fig. 2. For agent' processing (outbound and inbound), we will consider HTTP packets generated by App1 destined to DB 1 (both in tenant 2). First of all, the agent checks its PMD for a policy entry for this transaction. In our case, the policy on App1' VM mandates that packets destined to DB1' VM needs a security processing.

The basic outbound packet processing toward the DB1 occurs in the following sequence:

- App1 data is sent to the TCP/IP driver from the TCP/IP application.
- The TCP/IP driver sends an IP packet to the agent.
- The agent checks the packet for a policy match by looking up entries in the PMD.
- The agent checks the SCI by looking up on the SCD based on the policy entry, if there is not an appropriate SC, the agent chooses one randomly.
- The agent encapsulates the frame tag carrying the right credentials of the tenant tag of the tenant 2 and the application tag of App 1 (App 1 is a trusted application in tenant 2) within the SC initiation flag, the SCI and the token for the current communication session.
- Checksum recalculation.

– The IP packet is sent back to the TCP/IP driver.
– The TCP/IP driver sends the IP packet to the network interface.

Inbound processing is much simpler than outbound processing mainly because inner header construction is more complicated than header checking. The inbound packet processing from the App1 occurs in the following sequence:

– IP packets are sent from the network interface to the TCP/IP driver.
– The TCP/IP driver sends the IP packet to the agent.
– If the inbound packet contains an SC initiation flag.
– The agent extracts the SCI from the inner header.
– The agent then fetches the SC from the SCD using the sent SCI.
– If the SCD does not find the SC, an error is logged and the packet is dropped.
– If the SCD returns the SC entry, the agent decrypts the frame tag and the token.
– The agent authenticates the frame tag and stores the token for further communications.
– The agent sends that packet is sent to the TCP/IP driver.
– The TCP/IP driver performs IP packet processing as needed and sends the application data to the TCP/IP application.

After the initiation phase, the traffic between VMs is signed by the session token and instead of the SC initiation flag, the agent will send its requests with a data flag. Then the receiving end will have to decrypt and authenticate the token according to the negotiated SC. Hence, even if it is easy to make changes on the IP packet, it is very hard to change the payload to a desired value thanks to the randomness of the used SCIs (encryption methods, and hashing ...). Hence reducing the likelihood of payload tampering.

5 Conclusion

Cloud security is one of the buzz words in CC. Since virtualization is the fundamental of the CC, security and availability are critical for cloud environments because their massive amount of resources, simplifies several attacks to cloud services. In this paper, the focus of our work is about the visibility of the inter-VM traffic in a Cloud environment. Hence, we propose a new approach that gives an identity to this particular traffic, this identity is about the where and the who sends the request. Our approach relies on proposing a frame called frame tag that holds the proper credentials which are the tenant and the application that sends the IP packet, providing data origin authentication, and integrity, but also proposing an agent which is able to generate, capture and analyze this particular frame and respond to it by an automated acceptance or rejection, and also security mechanisms in order to ensure the integrity and security of this frame tag. The agent is the center-piece of our ongoing work. Therefore in our future work, we are trying to incorporate the autonomic computing characteristics on the agents functionality, in order to have not only a self-generating

rule engine but also a self-optimized agent and have an analysis at wire speed to meet security and performance since that those are the two key elements for a good Cloud service, and also study the integration of Deep Packet Inspection and Deep Content Inspection to see which one is the best fit for an efficient inter-VM traffic inspection, but more importantly in order to keep the system informant is to log every suspicious move in a centralized server for a correlation purposes or even for a Security Information Event Management.

References

1. Van der Merwe, J., Ramakrishnan, K.K., Fairchild, M., Flavel, A., Houle, J., Lagar-Cavilla, H.A., Mulligan, J.: Towards a ubiquitous cloud computing infrastructure. In: 17th IEEE Workshop on Local and Metropolitan Area Networks (LANMAN), Long Branch, pp. 1–6 (2010)
2. 3 Ways to Secure Your Virtualized Data Center, 29 July 2010. http://www.serverwatch.com/trends/article.php/3895846/3-Ways-to-Secure-Your-Virtualized-Data-Center.htm
3. A comprehensive framework for securing virtualized data centers, HP, August 2010
4. Schulter, A., et al.: Intrusion detection for computational grids. In: 2nd International Conference New Technologies, Mobility, and Security. IEEE Press (2008)
5. Schulter, K.: Intrusion detection for grid and cloud computing. IEEE J. IT Prof. **12**, 38–43 (2010)
6. Gul, I., Hussain, M.: Distributed cloud intrusion detection model. Int. J. Adv. Sci. Technol. **34**, 71–82 (2011)
7. Mazzariello, C., Bifulco, R., Canonico, R.: Integrating a network IDS into an open source cloud computing environment. In: IEEE Sixth International Conference on Information Assurance and Security (2010)
8. Security Architecture for the Internet Protocol, RFC 4301
9. IP Authentication Header, RFC 4302
10. IP Encapsulating Security Payload (ESP), RFC 4303
11. Irani, F.N.H.A., Noruzi, M.R.: Looking on policy and social policy in the context of public administration and management. J. Public Adm. Gov. **1**(1), 106–114 (2011)
12. Scarfone, K., Mell, P.: Guide to Intrusion Detection and Prevention Systems (IDPS). Computer Security Resource Center (National Institute of Standards and Technology) (2007)