International Journal of Communication Networks and Information Security (IJCNIS) Vol. 11, No. 1, April 2019

Application-Based Authentication on an Inter-VM Traffic in a Cloud Environment

Karim Benzidane, Sâad Khoudali, Leila Fetjah, Said Jai Andaloussi and Abderrahim Sekkaki

Computer Science Department, Laboratory of Research in Computer Science and Innovation
University Hassan II, Faculty of Sciences Ain Chock, Casablanca, Morocco

Abstract: Cloud Computing (CC) is an innovative computing model in which resources are provided as a service over the Internet, on an as-needed basis. It is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet. Since cloud is often enabled by virtualization and share a common attribute, that is, the allocation of resources, applications, and even OSs, adequate safeguards and security measures are essential. In fact, Virtualization creates new targets for intrusion due to the complexity of access and difficulty in monitoring all interconnection points between systems, applications, and data sets. This raises many questions about the appropriate infrastructure, processes, and strategy for enacting detection and response to intrusion in a Cloud environment. Hence, without strict controls 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 shines the light on the issues of security within Cloud Computing, especially inter-VM traffic visibility. In addition, the paper lays the proposition of an Application Based Security (ABS) approach in order to enforce an application-based authentication between VMs, through various security mechanisms, filtering, structures, and policies.

Keywords: Cloud Computing, Security, Inter-VM, DPI, DPDK, Blockchain, Inter-VM traffic, Virtualization**.**

1. Introduction

According to ISO subcommittee 38, the CC study group, Cloud Computing (CC) is a paradigm for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable cloud resources accessed through services which can be rapidly provisioned and released with minimal management effort or service provider interaction [1].

It has successfully managed to advertise itself as one of the fastest growing service models. For organizations, the clouds per-use approach provides tangible relief from hardware or software investments by offering a pay-for-service model. As an extension of Grid Computing and Distributed Computing, CC aims to provide users with flexible services in a transparent manner. The benefits of CC include greater resource access, dynamic scaling, and improved costs, along with the ease of automated management for resources and performance. Consumers adopt cloud computing to reduce infrastructure overhead, adjust service levels to meet changing needs, and to quickly deliver applications. CC relies on multi-tenant environments where multiple clients are served by one software instance. It offers 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 both from inside the firewall, as in the case of a private cloud, and from outside. The major roadblock to full adoption of CC has been concern regarding the security and privacy of information. Furthermore, attackers can exploit the large amount of resources in a cloud for their advantage. Network security is an important subject that is defined by protection of valuable resources such as services and information in the network. An intrusion is a group of actions that try to affect this security and consequently damage confidentiality, integrity or availability of resources. Therefore, providing security in a distributed system requires more than user authentication with passwords or digital data transmission.

In fact, due to it distributed nature, CC makes it an easy target, vulnerable and prone to sophisticated attacks. Often the most utilized technology to implement a Cloud environment is virtualization with a massive multi-tenancy usage; it opens a door to a whole other level of security issues. The security factor of a CC infrastructure is an important one. In this case, we need to look into the Cloud environment (Network, Systems and applications) in a very deep manner to keep it informant, that will help to prevent security and performance disruptions which can destroy and compromise smooth transactions, and since the main concept of the CC is to allow end users to execute various applications in on-premise or off-premise resources, most of them are shared in a virtual environment creating new security and performance challenges leveraging attacks to be launched from compromised Virtual Machines (VM) that can damage the ability to serve all end users demands. In a virtual environment, there are several VMs hosted on a single physical server or hypervisor, where communication level issues can be identified either at the network level, host level and application level [2]. VMs generally inter-communicate with each other's via virtual switch without leaving the server, and this introduces the network blind spot letting any network security appliance set on the LAN blind to any communication between VMs and this could be within a single host and also across physical servers. If the traffic doesn't need to pass through that security appliance mostly a firewall, opening a loophole for all sorts of security attacks. Thus, the starting point of an attacker is compromising only one VM and using it as a springboard to take control of the other VMs within the same hypervisor (a technique called VM hopping or jumping) and this is generally done without being monitored or detected, giving the attacker a huge hack

domain.
The remainder of this paper is organized as follows: in the next section, we discuss some Cloud computing security issues, and more specifically in regards to virtualization, Software Defined Networking (SDN), and some basics of the blockchain. The third section presents the layout of the proposed approach, its various components, operations, and implementation. The last section contains a summary and conclusions.

2. Background

Cloud Computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications) delivered as a service. It is a disruptive technology that has a direct impact into enhancing collaboration, agility, scaling, availability, and provides the opportunities for cost reduction through optimized and efficient computing.

2.1 Cloud Computing Security Issues

As widely known, virtualization is the key technological driver behind the huge success of CC. Initially driven by the need to consolidate servers to achieve higher hardware utilization rates, boost operational efficiency, and cut costs, companies have implemented virtualization to get on-demand access to CC. This access can result in the creation of multiple VMs out of a single physical server. One of the outstanding properties of virtualisation 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. Additionally, Multi-tenant infrastructures typically offer scaled performance and services based on shared resources, including databases, other applications, and OSs.
Since each VM is itself a virtual server comprising a guest OS, middleware, application and data, virtualization comes with new challenges; new attack surface at the hypervisor level, as well as an important impact on network security. It should be noted that VMs communicate over a hardware backplane rather than a network. As a result, the standard network security controls are blind to the overlay traffic and cannot perform monitoring or in-line blocking. In terms of actual threats to a virtualized environment [3] [4], these fall into a number of categories, cantered mostly on the hypervisor, which leaves some organizations prone to a variety of threats [5] [6].

- **Lack of data interoperability standards:** It results into Cloud user data lock-in state. If a Cloud user wants to shift to other service provider due to certain reasons it would not be able to do so, as Cloud user's data and application may not be compatible with other vendor's data storage format or platform. Security and confidentiality of data would be in the hands of Cloud service provider and Cloud user would be dependent on a single service provider.
- **Cloud data confidentiality issue:** Confidentiality of data over Cloud is one of the glaring security concerns. Encryption of data can be done with the traditional techniques. However, encrypted data can be secured from a malicious user but the privacy of data even from the administrator of data at service provider's end could not be hidden. Searching and indexing on encrypted data remains a point of concern in that case.
- **Network and host-based attacks on remote Server:** Host and network intrusion attacks on remote hypervisors are a major security concern, as Cloud vendors use VM technology. DOS and DDOS attacks are launched to deny service availability to end users.
    - **ARP Attack:** ARP lacks very much when it comes to security, a malicious user is able to use a forged IP address of Layer 3 and MAC address of Layer 2, there is no way to verify those forged details in ARP. The malicious user identifies him as a legitimate user and starts to use resources available on the network. It’s even possible to transmit ARP packets to a device in a different VLAN using those forged details.
    - **VLAN Hopping Attack:** VLAN hopping works by sending packets to a port which should not be accessible. Basically, in VLAN hopping attack there are two types. The first one is switch spoofing happens when a malicious user tries to configure a system to spoof itself as a switch by matching itself to 802.1q or ISL. The malicious user is able to spoof the switch with help of (Dynamic Trunk Protocol) DTP signalling. The other type is double tagging, which is a method involves tagging transmitted frames with two 802.1q headers, one of the headers is used for Victim switch and another is used for the attacker’s switch.
    - **Private VLAN Attack:** A Private VLAN is a feature in Layer 2 which is used to isolate the traffic only at layer2. When a layer 3, device such as a router is connected to a Private VLAN, it supposed to forward all the traffic received by the router to whatever destination it’s meant for. Sometimes a malicious user might use it for his advantage.
    - **MAC Flooding Attack:** MAC flooding attack is one of the common attacks on a VLAN. In a MAC flooding attack, the switch is flooded with packets of different MAC address therefore consuming memory on the switch. During the MAC flooding attack, switch starts to behave like a “hub” where it starts to share the data with all the ports. Thus, a malicious user is able to use a Packet sniffer to extract the sensitive data.
- **Virtualization *security:*** Virtualization brings with it all the security concerns (see figure 1) of the operating system running as a guest, together with new security concerns about the hypervisor layer, as well as new virtualization specific threats, inter-VM attacks and blind spots, performance concerns arising from CPU and memory used for security, and operational complexity from “VM sprawl” as a security inhibitor. New problems like instant-on gaps, data comingling, the difficulty of encrypting VM images and residual data destruction are coming into focus. Virtualization has a large impact on network security. VMs may communicate with each other over a hardware backplane, rather than a network. As a result, standard network-based security controls are blind to this traffic and cannot perform monitoring or in-line blocking. Migration of VMs is also a concern. An attack scenario could be the migration of a malicious VM in a trusted zone, and with traditional network-based security controls, its misbehaviour will not be detected. Installing a full set of security tools on each individual VM is

another approach to add a layer of protection.

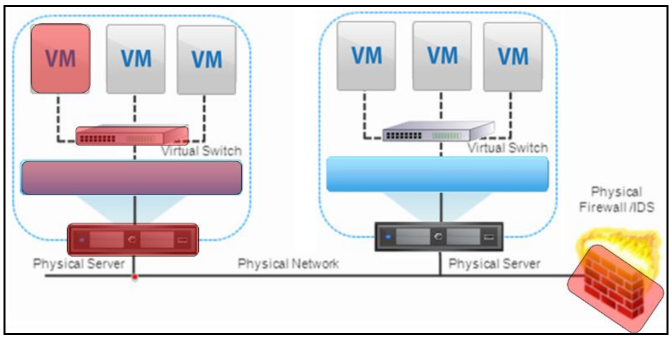


Figure 1. Virtual architecture, and its potential attack vectors

- **Hyperjacking:** 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. In theory, a hacker with control of the hypervisor could control any virtual machine running on the physical server.
- **VM escapes**: is a security attack designed to exploit a hypervisor, it allows a hacker to gain access over the hypervisor and attack the rest of the VMs. If the attacker gains access to the host running multiple VMs, the attacker can access the resources shared by the other VMs.
- **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 SS user's data once he gains access to a targeted VM by VM hopping, endangering the confidentiality and integrity of SS. VM hopping is a considerable threat because several VM's can run on the same host making them all targets for the attacker.
- **VM mobility/migration**: enables 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 IS 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.

2.2 Multi-tenancy

Multi-tenancy in its simplest form implies the use of same resources or application by multiple consumers that may belong to same organization or different organization. The impact of multi-tenancy is visibility of residual data or trace of operations by another user or tenant. Multi-tenancy in Cloud service models implies a need for policy-driven enforcement, segmentation, isolation, governance, service levels, and charge-back/billing models for different consumer constituencies [7].

2.3 Software Defined Networking

According to the Open Networking Foundation (ONF), “Software-Defined Networking (SDN) is an emerging architecture that is dynamic, manageable, cost-effective, and adaptable, making it ideal for the high-bandwidth, dynamic nature of today’s applications.”. Although both virtualization and Cloud predate SDN, the latter is now providing a reliable and effective foundation for the growth and success of Cloud business models. SDN is increasingly accepted as the path to "cloud networking," meaning the transformation of networks and services to support the use of cloud computing on a massive scale. Navigating the various missions and technology models of SDNs is critical to properly position cloud services and realize advantages of cloud computing [8]. The ONF lays out the architecture of SDN as an architecture that “decouples the network control and forwarding functions enabling the network control to become directly programmable and the underlying infrastructure to be abstracted for applications and network services”. According to the ONF, the SDN architecture is:

- **Directly programmable:** Network control is directly programmable because it is decoupled from forwarding functions.
- **Agile**: Abstracting control from forwarding lets administrators dynamically adjust network wide traffic flow to meet changing needs.
- **Programmatically configured:** SDN lets network managers configure, manage, secure, and optimize network resources very quickly via dynamic, automated SDN programs, which they can write themselves because the programs do not depend on proprietary software.
- **Open standards-based and vendor-neutral:** When implemented through open standards, SDN simplifies network design and operation because instructions are provided by SDN controllers instead of multiple, vendor-specific devices and protocols.
- **Centrally managed:** Network intelligence is (logically) centralized in software-based SDN controllers that maintain a global view of the network, which appears to applications and policy engines as a single (see figure 2), logical switch. SDN Controllers receive instructions from SDN Applications via Northbound APIs, and send other instructions to "below" Devices via Southbound APIs. Southbound APIs work in parallel with SDN Protocols, like it is depicted in figure 2.

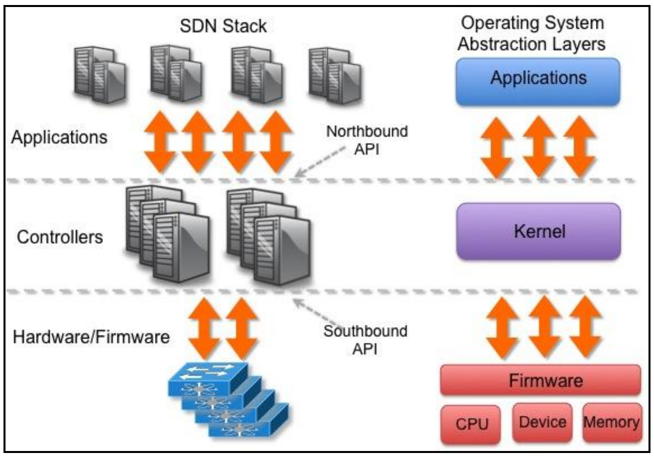


Figure 2. SDN architecture [9]

Open Flow Protocol is a standard and one of the ways of achieving communication between controller and switching infrastructure in SDN framework. Open Flow is standardized through Open Networking Foundation (ONF) to achieve the objectives of Increased Network Functionality, while lowering operational expenses through simplified Hardware, Software and Management. ONF through its working groups responsible for the development of protocol, configuration and interoperability testing [10].

2.4 Blockchain

Blockchain is a distributed ledger [11] (also called a shared ledger, or referred to as Distributed Ledger Technology) that is secured by different encryption techniques that provides an integrity and availability of an information record without the need for a verification by a centralized entity. As Blockchain has started, it delivered trust (in a product, a transaction or the integrity of data) far more efficiently and effectively than any existing technology. Although it has existed since 2008 (as the basis for the Bitcoin cryptocurrency), its application has expanded beyond the financial services realm towards other fields pretty quickly.

Blockchain can be seen as a series of data blocks hence the name [12], and each block is containing information about events, transactions or any type of entries. Therefore, once the block is recorded, the data in any given block cannot be altered retroactively without the alteration of all subsequent blocks. Each block is securely hashed and this hash is stored in the next block which makes it nearly tamper proof. These blocks are linked together into a chain and broadcast across the network to various nodes to store into their own copy of the ledger. This ledger is considered as shared/distributed, which is a digital record of ownership that differs from traditional database technologies, since it is replicated among many different nodes in a peer-to-peer virtual private network, and each transaction is uniquely signed with a private key with no central administration nor storage. Before adding a block to the chain, the block's validity must go through a consensus mechanism. The purpose of this mechanism is to ensures that all participants of the distributed ledger are on the same page. In a nutshell, a consensus is defined as the full-circle verification of the correctness of a set of transactions comprising a block. Thus, making it difficult for hackers to introduce untrusted transactions (as long as a majority of nodes are true), ensuring trust and integrity without the need for a central authority eliminating the risk of a single point of failure. In case of a breach occurring, its location can be determined and isolated quickly without impacting the rest of the network.

2.4.1 Consensus mechanisms:

Consensus mechanisms have been a topic of active research for nearly three decades now, and way before the upcoming of blockchain [13]. It is a mechanism that helps the update of a distributed shared state in a secured fashion. Therefore, distributing a shared state across multiple replicas in the network is one of the common techniques used for achieving fault tolerance in a distributed system but not its integrity. In contrast, adding and validating the replicated shared state should happen according to a pre-defined set of rules defined by the state machine that should be executed on all the replicas. This is what is called state machine replication, where replication of state guarantees that the state is not lost nor altered in one or more nodes. These replicas communicate with each other to build what is called a consensus and agree upon the irrevocability of the state after a state change is executed. In the blockchain world, the shared state is the distributed shared ledger and the state transition rules are the rules of the blockchain protocol.

In Blockchain [13], consensus is accomplished ultimately when a block's entries have met the explicit policy rule checks. These checks take place during the lifecycle of a block, and include the usage of endorsement policies to dictate which specific nodes must endorse a block, as well as a business logic to ensure that these policies are enforced and upheld. Prior to commitment, the peers will use these business logics to make sure that enough endorsements are present, and that they were derived from the appropriate nodes. After that, a versioning check will take place during which the current state of the ledger is agreed or consented upon, before any blocks are appended to the ledger. This final check provides protection against threats that might compromise data integrity.

In general, a consensus protocol has three major key properties based upon which its applicability and efficacy can be determined.

- **Safety/consistency** – A consensus protocol is determined to be safe if all nodes produce the same output and the outputs produced by the nodes are valid according to the rules of the protocol.
- **Liveness** - A consensus protocol guarantees liveness if all non-faulty nodes participating in consensus eventually produce a value.
- **Fault Tolerance** – A consensus protocol provides fault tolerance if it can recover from a byzantine node participating in consensus.

The most known use case of blockchain is bitcoin, which uses a consensus mechanism called Proof of Work [12] [14]. PoW is the original consensus algorithm in a Blockchain network where miners compete against each other to complete transactions on the network and get rewarded. Its approach is probabilistic and have to spend significant amount of time/computing solving a cryptographic puzzle. That is why, bitcoin has high transaction latencies and therefore a low transaction rate. On the flip side of bitcoin there is other cryptocurrencies and blockchains which are

using other types of consensus.
Proof of Stack [15] is one of them, where the key motivation behind it is that mining is done by stakeholders in the ecosystem who have the strongest incentives to be good stewards of the system. Therefore, the election of the creator of the next block is done via various combinations of random selection and wealth or age (i.e., the stake) rather than solving computationally an intensive puzzle to validate transactions and create new blocks.
Another consensus mechanism that has been on the rise lately is Proof of Authority [16] where blocks are validated by approved accounts, known as validators. It is an algorithm used in blockchains that have the need to deliver fairly fast transactions through a consensus mechanism based on identity as a stake.
Consensus in blockchain comes also to mitigate what is called Byzantine Generals Problem [17], where a byzantine node can lie, provide incorrect responses or mislead other nodes involved in the consensus network. Therefore, a consensus algorithm has to be able to function correctly and reach consensus in the presence of Byzantine nodes as long as their number within a distributed system are limited. One of the first practical solution to the achieving consensus in the face of Byzantine failures was Practical Byzantine Fault Tolerance, an algorithm proposed by Miguel Castro and Barbara Liskov [17]. PBFT uses the concept of state machine replication and voting by replicas for state changes, provides also some important optimization such as signing and encryption of messages between replicas and clients which reduces the overhead. The PBFT algorithm requires “3f+1” replicas to be able to tolerate “f” failing nodes, where the maximum number of nodes that I can be scaled to is 20 because the overhead increases significantly as the number of replicas increases. This consensus mechanism is used by Hyperledger; a project that allows developers to create their own digital assets with a distributed ledger built on the principles of BFT.

2.4.2 *Permissioned blockchain:*

Permissioned and permissionless blockchains are the two main types of the blockchain platforms. The most known cryptocurrencies such as Bitcoin and Ethereum are considered as permissionless or open blockchains since they are publicly available for use and any node can make transactions as well as take part in the consensus process. In contrast to permissionless platforms, there is permissioned blockchains (can also go by the name of 'consortium' blockchains.) such Hyperledger which are aimed at groups where participation is closed. Submission of entries can be done by any type of nodes within the group, but for the validation of blocks where the restriction comes at hand, which can be fixed to a set of peering nodes that run by consortium members. These groups are expected to be small in number and an access control layer is used to govern and vet who can have access to the network, therefore the consortium can employ alternative consensus mechanisms than proof of work for instance such as PBFT, PoA... It should be mentioned that there is no concept of digital currency on private permissioned distributed ledgers because the objective of this type of platform is different from a public one.

2.4.3 *Merkel Tree*

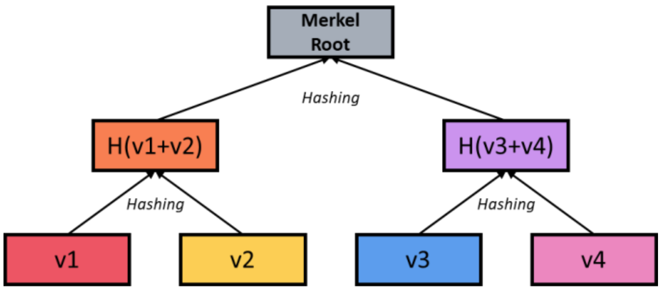


Figure 3. Merkle tree of 4 values.

In computer science and cryptography, a Merkle tree [18], as seen on figure 3, (or a hash tree) is a binary tree in which every leaf is labelled with a data block and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Hence, the tree is constructed by recursively hashing pairs of nodes until there is only one hash, called the root, or Merkle root. Merkel trees allow efficient and secure verification of the contents of large data structures such as a block in a blockchain platform. Hash trees are a generalization of hash lists and hash chains.

3. Our Approach

The emergence of Cloud Computing thrived immensely upon virtualisation that got shifted towards a world of on-demand scalability and service delivery over the Internet. Virtualization is one of the key enablers and key technologies to build upon a Cloud infrastructure. It is increasingly used in portions of the back-end of Infrastructure as a Service (IS), Platform as a Service (PS) and SS (Software as a Service) providers as well. Virtualization is also, naturally, a key technology for virtual desktops infrastructure (VDI), which often times are delivered from private or public Clouds. The benefits of virtualization are well known, including multi-tenancy, better server utilization, and data centre consolidation. Cloud Providers can achieve higher density, which translates to better margins, thus companies can use virtualization to shrink capital expenditure (Capex) on server hardware, as well as increase operational efficiency.
However, as it has been mentioned at 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 analysing this particular inter-VM traffic, authenticate/authorise it, and then preventing the non-compliant one.
Traditional network security mechanisms face new challenges to keep up with the cloud infrastructure. For instance, having virtualization as its foundation would bring in the security issues of the said underlying technology, such as virtual machine intrusion attacks and malicious user activities.

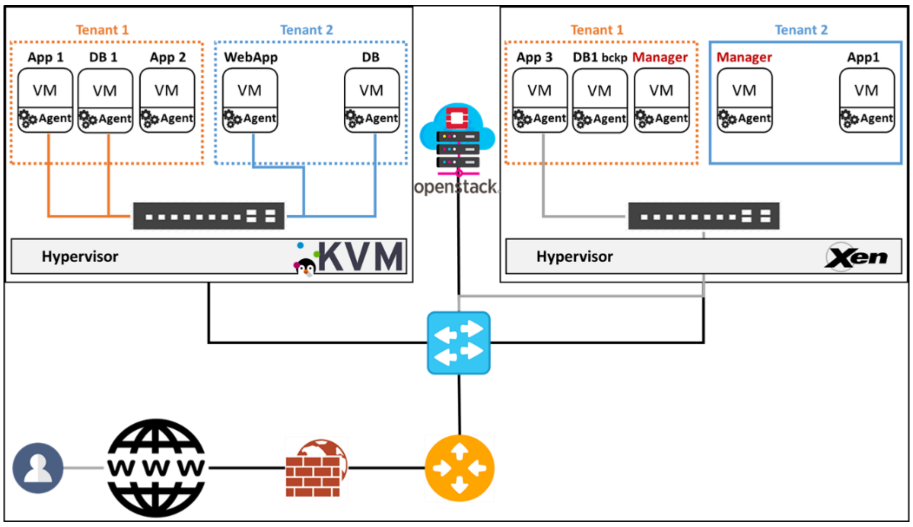


Figure 4. Basic Cloud architecture (testbed) Basic Cloud architecture (testbed)

New security methods are therefore needed to mitigate those new issues. Multiple research activities were introduced to address the issues of intrusion detection within cloud computing environments. [19] developed an intrusion detection model leveraging machine learning approach, by using 28 features subset without content features of Knowledge Data Discovery (KDD) dataset to build machine learning model and are most likely to be applied for the IDS. [20] proposes an IDS based on mobile agents (MAs). Its most important weaknesses are the performance and the security issues related to MAs. [27] has classified the deployment approaches of IDS in a cloud environment into several categories ranging from guest-based, VMM-based, Network-based, collaborative agent-based, to distributed. [21] uses an IDS sensor such as the version of Snort installed on VMware virtual ESX machine that sniffs in-bound traffic. [22] made a prototype of Cloud IDS inspired by Dendritic Cell mechanism, which mimics the activity and process of Dendritic Cell which is known for detecting and killing any pathogens that infected human tissue and cells. Snort matches in bound packets against several intrusion patterns. If a match occurs, all the traffic from the same IP address is dropped. No evaluation of the accuracy and performance of this solution is presented. Furthermore, network events are not correlated to discover attacks against several virtual zones. [28] has proposed a model that provides a security as a service at the infrastructure layer and analysis the alerts of users based on the system calls. [29] presents a design of a virtualization-based detection solution called VMFence to examine the network flow and integrity of a file and also to detect the real-time attacks. [30] has proposed an analysis solution based on k-means clustering for anomaly detection and integrated it with a frequent attack generation module using prior algorithms to detect recurrent attacks and find the signature of the attack within the cloud environment. Relating to previous works especially in intrusion detection [23] [24], we find that all of the measures that should be taken must to be distributed since it is a Cloud environment [25]. However, the work done so far is about detecting and preventing malicious traffic [26] outside of the wire i.e. the hypervisor.

Currently, cloud providers enforce data encryption for storage containers, virtual firewalls and access control lists [31]. The proposed framework builds upon the fact that new levels of security onto those already supplied by cloud providers are required. Therefore, the major contribution from our work is a about authenticating the access to the critical virtual machines, thus, by securing the inter-VM communication. We are basing our work on a simple model as depicted in Figure 4, containing the essential elements that we can find in a Cloud Service Provider or a Cloud Infrastructure: Cloud Manager, Hypervisor, Virtual Machines...

- **Cloud manager:** is a management and orchestration server for the admin/client in order to manage their Cloud resource infrastructure.
- **Hypervisor**: also called Virtual Machine Monitor (VMM) allows multiple operation systems, termed guests or VMs, to run concurrently on a host server. Actually, the hypervisor controls the host processor and resources, allocates what is needed to each OS in turn and makes sure that the guest VMs cannot disrupt each other.
- **Virtual Machine**: is a completely isolated guest operation system installation within a normal host OS, and as it was originally defined by Popek and Goldberg as "an efficient, isolated duplicate of a real machine". All the VMs in our architecture has an embedded agent, which will serve the purpose of our approach.

Our approach focusses particularly on the analysis and authentication of the inter-VM traffic by introducing a security structure characterized by a filed called frame tag. This field is introduced by a VM embedded agent at the beginning of the IP packet' payload. Thus, ensuring a high filtering level, by making the receiving VM/agent detect, analyse and authenticate the incoming traffic then respond by accepting or refusing this IP packet according to the compliance of the information on the frame tag or the IP packet in general. The idea behind this approach comes for the issue of network visibility in a virtual environment, which is a serious issue for the security appliances. Thus, 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 that should be detected by a security solution, from a VM hosting a database, while in a legitimate scenario as depicted in Figure 4, 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, meaning who sends the request and from where. Hence, what we are trying to achieve with our approach is to authenticate each request/IP packet communicating between VMs by encapsulating these two identifiers in the frame tag which will be an authentication credential for the IP packet by the sending VM/application, then retrieved and analysed by the receiving VM/agent acting like a light-weight intrusion prevention system or a firewall by refusing (DORP or REJECT) the non-compliant packets.

This approach creates a boundary for malicious traffic, between two communicating VMs. Traffic traversing this boundary is subject to the access controls specified by the policy of the receiving agent. In order to fulfil the authentication factor, which is the cornerstone of our approach we introduced some security mechanisms wrapping the frame tag. This security is manifested in a Security Contract between the communicating VMs, a policy management and of course encryption. Those mechanisms will be thoroughly introduced within the next sections.

This chapter provides a high-level description of the frame tag, the security contract, the agent, 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 infrastructure, and its implementation which is describes in more details.

3.1 Frame Tag

Users in a Cloud environment access their services by providing a digital identity. Commonly, this identity is a set of bytes related to the user. Based on this digital identity, a Cloud system can recognize what appropriate access this user has and what is allowed to do. Most of the Cloud platforms include an identity service management service (like keystone in OpenStack, or IAM in AWS). Following the same analogy, our approach is about identifying the application that sends the request (IP packet) to another machine in the same tenant. Hence, we propose a field called frame tag, as shown in the figure 5. The frame tag is generated between a pair of communicating agents, and its structure contains two main fields: Tenant tag, and Application Tag.

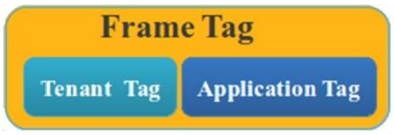


Figure 5. Structure of a frame Tag.

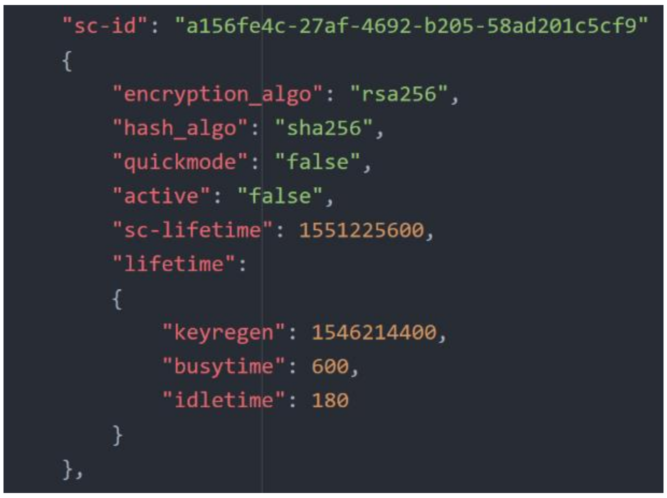
- **The tenant tag field:** When a tenant is created in a Cloud, the identity management service takes on the task of creating an ID for this tenant. Similarly, the same thing happens with the tenant tag which is a value generated holding the identity of the tenant. Thus, when a request is sent from a VM to another VM within the same tenant, the tenant tag is retrieved from the database and injected by the agent in the payload of the IP packet. This flied ensures the integrity of the inter-VM communication, but more importantly the prevention of spring boarding to another VM within another tenant, thereby adding a strong layer of tenant isolation alongside VLAN isolation.
- **The application tag field:** When an application is added and installed on a VM, the admin has to certify it as trusted by creating a signature to rely on for its detection by the agents. The use of the application tag in our approach would help authenticate for instance App1 in Tenant 1 that wants to send a request to DB1 hosted in another VM within the same tenant, provided that the admin has certified App1 as trusted. The sending VM will generate the application tag according to the detected signature of the application on the IP packet.

The frame tag in our proposed approach plays the role of an authenticator by encapsulating credentials (the tenant and the application) of the whereabouts and the identity of the application sending the request to another VM. Therefore, ensuring a level of authentication of the inter-VM traffic in a non-intrusive way and more importantly isolation between tenants. Therefore, having a monitored traffic by an agent acting as light-weight intrusion detection and prevention system and responds according to the frame tag field’s values.

3.2 Security Contract

Ensuring isolation and security in CC is a concerning issue for potential users and clients. Therefore, our approach is about giving a high level of trust and isolation within tenants, through the security and the integrity of the tenant tag and the application tag by introducing what we call a Security Contract (SC) which can be seen analogous to IPSec [32] [33]. An SC is a unidirectional 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. Therefore, with an SC is not only ensuring the integrity of the inter-VM traffic but also its authentication. 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. Hence, The SC-A and the SA-B will have the different security attributes. Data sets associated with an SC are represented in the SC Database (SCD). Though SCs are unidirectional, a shared SCD between agents is maintained for all SCs used for outbound and inbound processing. 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 as seen on figure 6 (meaning no encryption is being performed).

- **Security Contract ID:** is an ID identifying a security contract in the form of a generated UUID version 4. This ID helps the VM to know which SC is going to be used for outbound packets during a session. For example, multiple contracts might be used if a VM is communicating with multiple hosts simultaneously. This situation can occur when an VM is hosting a Data Base server for instance that responds multiple hosts. In this situation, the DB VM uses the sc-id to determine which SC is used for outbound packets in order to be appropriately processed by the agent accordingly.



```
    "sc-id": "a156fe4c-27af-4692-b205-58ad201c5cf9"
    {
        "encryption_algo": "rsa256",
        "hash_algo": "sha256",
        "quickmode": "false",
        "active": "false",
        "sc-lifetime": 1551225600,
        "lifetime":
        {
            "keyregen": 1546214400,
            "busytime": 600,
            "idletime": 180
        }
    },
```

Figure 6. Sample of a Security Contract entry.

- **Active*:*** is a field that takes three values; true, false and expired. When the value is true, it means that the SC is enabled and can be used as opposed to a false value which means that the SC is disabled temporarily. In contrast to these two values, the expired value means that the sc has reached its lifetime and it is disabled without the option of being enabled again.
- **Security contract lifetime:** is an UNIX epoch time that represents the expiration date of the security contract. When reached, the active field gets the expired value in order to disable the usage of the SC. Once a security contract is expired, it will be removed from the heap.
- **Lifetime*:*** is a lifetime value associated with each SC beyond which the SC can and cannot be 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 represented in the form of an POSIX time. Whenever a key lifetime is reached, the SC is updated with new regenerated keys. Busy and idle session lifetimes are used to determine the lifetime of a session in seconds before the current used SC is expired and renegotiate another one.

The negotiation of an SC is the security needed for a frame tag to be authenticated and processed properly. Therefore, the communication between VMs has to abide to the convened SC, by encrypting and encapsulating the frame tag accordingly. During this negotiation, 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 header on the payload in order to be authenticated during the communication.

3.3 Security Contract Database Blockchain

As it has been aforementioned, the SCD is a shared and distributed database wherein every participant has their own replicated copy where all the SCs are stored. Making any unwanted change on the SCD a huge vulnerability that a malicious user could take advantage of in order to send a malicious traffic towards another VM. Therefore, securing the integrity of the SCD from any type of unwarranted alteration is of the utmost importance, hence applying blockchain to it. Integrity is a way of avoiding any tampering at the security contract entries. Blockchain uses cryptographic hashing to ensure that the ledger remains tamper-proof. One of the key characteristics of this hashing function is that it is always one-way, which means it is logically impossible to get the data back from the hash result or from the message digest. It is also difficult to analyse the pattern of message digest and predict the original data as even a slight change in the actual message can result in a big difference. Therefore, the application of blockchain to the SCD would highlight its peer-to-peer distribution factor (i.e. a distributed ledger) and also make it cryptographically secure, append-only, immutable (extremely hard to change), and updateable only via consensus or agreement among peers.

With the implementation of blockchain, the SCD will consist of two records; SC entries and blocks. The block hold batches of SC entries that are hashed and encoded to a Merkel tree. Each block of course would include the hash of the prior block linking them together. As it was above-mentioned in the background section, the ledger' entries in blockchain are kept synchronized across the network and each block append is approved by the appropriate participants within the blockchain' network via an agreed upon algorithm called a consensus.

The consensus in the SCD blockchain will make sure that selected nodes will agree and validate the proposed SCs, which then will result an update of the ledger i.e. SCD. The proposed consensus mechanism for this blockchain will be a set of procedures and rules that will keep a coherent SCs state among the VMs on the same tenant.

3.3.1 Security Contract Block

The application of blockchain on the SCD comes as an obvious choice due to the nature of their respective distributed and shared implementation, but also to guarantee the integrity of the SCs within the SCD since they are a key component in the proposed approach. Therefore, once the SCs are validated, the SCD blockchain cannot be altered retroactively without the alteration of all subsequent blocks as depicted on figure 7.

Simply put, a block (see figure 7) is a selection of SCs bundled together with a block header. Its size of course may vary depending on the amount of SCs created on the master.

Each block within the blockchain is identified by a hash, generated using the SHA256 cryptographic hash algorithm on the header of the block. Each block also references a previous block, known as the parent block, through the "previous block hash" field in the block header. In other words, each block contains the hash of its parent inside its own header. The sequence of hashes linking each block to its parent creates a chain going back all the way to the first block ever created, known as the genesis block. A genesis block is the first block in the blockchain that was hardcoded at the time the blockchain was started. The master will create two default dummy security contracts just to populate this genesis block. On the other hand, the process of creating a block in the SCD blockchain stems from the creation of one or more security contract.

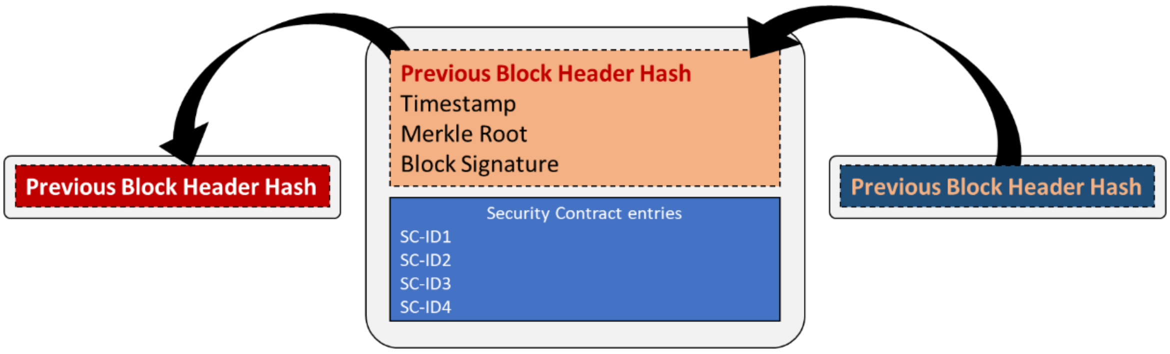
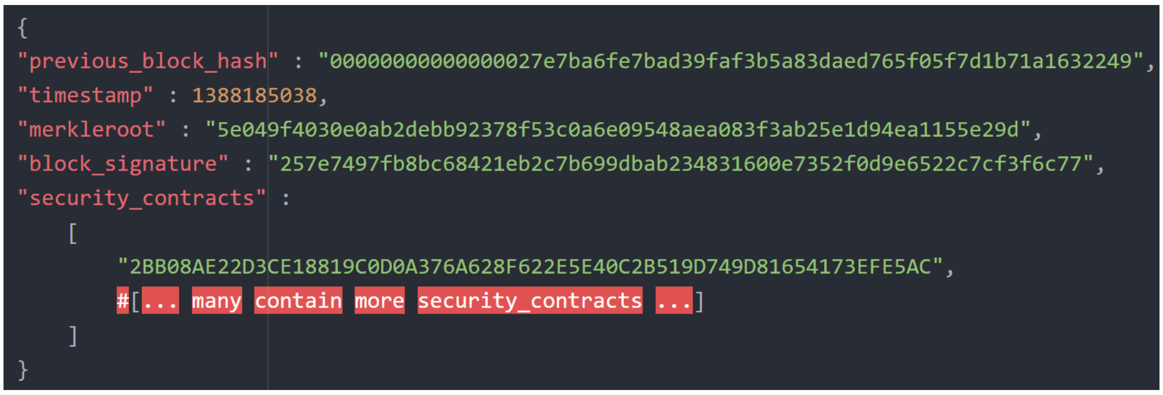

Previous Block Header Hash
Timestamp
Merkle Root
Block Signature
Security Contract entries
SC-ID1
SC-ID2
SC-ID3
SC-ID4
Previous Block Header Hash
Previous Block Header Hash

Figure 7. General architecture of a blockchain.



```
{
"previous_block_hash" : "000000000000000027e7ba6fe7bad39faf3b5a83daed765f05f7d1b71a1632249",
"timestamp" : 1388185038,
"merkleroot" : "5e049f4030e0ab2debb92378f53c0a6e09548aea083f3ab25e1d94ea1155e29d",
"block_signature" : "257e7497fb8bc68421eb2c7b699dbab234831600e7352f0d9e6522c7cf3f6c77",
"security_contracts" :
    [
        "2BB08AE22D3CE18819C0D0A376A628F622E5E40C2B519D749D81654173EFE5AC",
        #[... many contain more security_contracts ...]
    ]
}
```

Figure 8. General JSON Structure of a Block.

The SCs are either created manually or automatically on the master, and their number should always be binary since they are going to be hashed in a Merkle tree - For instance if only one SC is created manually, the master will create a dummy SC for the Merkle tree-. Then, the Merkle root will be sent for validation via our proposed agreement protocol (see the next section). Once validated, comes the commit phase where the block is being created. The block is made of a header, followed by the sub-block of the security contracts. The block header (see figure 8) consists of four sets of block metadata. First, there is a reference to a previous block hash which is a hash of the block header that can even act as an identifier, that connects the current block to the previous one. The second set of metadata, namely timestamp which is the epoch Unix time of the time of the block initialization. The third piece of metadata is the Merkle root, a data structure used to efficiently summarize all the security contracts in the sub-block. The last metadata field is the block signature, which is a cryptographic proof in the form of a hash that is made during the consensus phase in order to validate and commit the block in the SCD blockchain.

3.3.2 SCD Blockchain Consensus

It is known that the strength of blockchain comes from its immutability, and that is due to the chosen consensus which is considered the most crucial aspect that requires close attention when implementing any type of blockchain.

We have settled on the fact that the SCD blockchain is a permissioned blockchain, since all the participants of the network are known and already trusted (relatively). Therefore, the consensus mechanism for our particular blockchain would be an agreement protocol that will be used to maintain a shared and synchronized version of truth about the state of records on the SCD. Consequently, that will exclude any type of mining as all the participants are already know each other and there is no requirement for mining to secure the network.

Consensus is basically a distributed computing concept that has been around for a long time, and it has been used in blockchain as well in order to provide a means of agreeing to a single version of truth by all peers on the blockchain network. In our case, some of these nodes will verify and, if appropriate, validate the proposed security contracts according to an agreed-upon consensus process.

The consensus architecture of our SCD blockchain is a master/slave type of architecture, where the master is the starting point of the validation process of a block. This process goes through several phases that are inspired by existing consensus mechanisms.

The security contracts are created at the master level, which is considered as a block generator. At each SCs creation, the master is triggered to assemble them into a block as well as compute the Merkle root from the created SCs. The Merkle root will be digitally signed by the master in order for it to be considered as a candidate in the consensus process, then broadcasted over the blockchain network which will be picked up by a selected leader node to begin the validation process.

The leader selection as depicted on figure 9, is a two-fold process. The first one is a pseudo-random selection of a node to prevent any selective censorship attack, which is done during each block validation phase. The second part is the identity validation of said selected node in order to confirm

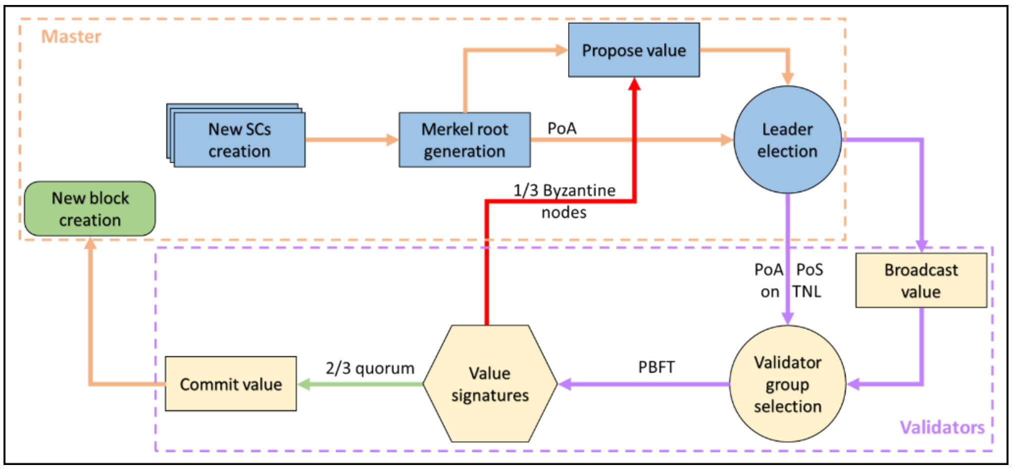


Figure 9. Workflow of the consensus process in the SCD Blockchain.

its election then become a leader. In order to help with the identity validation, each node that joins the SCD blockchain must create an ECDSA keypair. Since this is a permissioned network, all nodes will have to register their identity with the master in order to access the blockchain network. The master will make use of the PKI to support identity management and authorization operations during the whole block validation and commit phase.

After the leader selection and proving its identity, the system will switch to proof of stake. The master will send a Trusted Nodes List (TNL) to the leader based on their stake in the network. The stake in the SCD blockchain is represented by the compute resources (RAM and CPU) of a node, and also a security score which is a score attributed to a VM/node according to the hosted trusted applications in it. Based on the sent said list from the master, the leader will choose a random group of validators (ensuring that no validator can predict its turn in advance) from it to validate and sign the new proposed block. At this final stage, the validation becomes leader-less and only the chosen nodes will participate in it. A node is defined as an individual VM that holds a replicated copy of the SCD, and identified by their public keys. All nodes are capable of sending and receiving messages to and from each other. Nodes can be honest, faulty, or malicious. A node that can exhibit arbitrary behaviour is also known as a Byzantine node. This arbitrary behaviour can be intentionally malicious, which is detrimental to the validation phase.

Since that the SCD network is a permissioned one, this led us to use a small consensus group over the need to achieve the decentralization of open and public blockchains such as Bitcoin or Ethereum. Therefore, the best consensus that fits our network is PBFT. It is an effective consensus protocol for providing high-throughput transactions without needing to worry about optimizing the platform to scale to large consensus groups. With the application of PBFT at this validation stage, the elected validators will have to commit new blocks in the blockchain. These validators participate in the PBFT consensus protocol by broadcasting votes which contain cryptographic signatures signed by each validator's public key. The signature is based on ECDSA scheme and makes use of the SECP256k1 curve, and it will also help identify the source of the exchanged message since that each node publishes their public key.

The application PBFT on the SCD blockchain works on three stages in which nodes broadcast messages to each other. First, the pre-prepare stage consisting of a leader selection that has the Merkel root to commit. This stage has already been done by the master during the aforementioned election phase of the whole consensus. Next, the prepare stage broadcasts the Merkel root replica to be validated. All the state machine replication techniques require two major things on their replicas, that applies also on the broadcasted Merkel root replica in the SCD network. Replicas are required into being deterministic, where the execution in a given state must always produce the same result. Adding to that, Replicas must always start in the same state. With those two requirements, the PBFT guarantees the safety property by ensuring that all none byzantine nodes agree on a total order for the execution of requests despite failures. Finally, the commit stage waits for more than two third quorum of all the validators in a partially asynchronous model to confirm the proposed value before announcing that the value is committed. Once the leader has received two third endorsements from the validators for the master' proposed Merkel hash, this value gets committed and the leader sends it to the master. The master in its turn will generate the block header, i.e. the timestamp and the block signature from the ones sent by the leader, making the block full-fledged to appended on the chain and committed to the ledger. It should be mentioned that PBFT can only tolerate up to a one third of Byzantine nodes, where failures can include arbitrary or malicious behaviour, thus validators will never commit conflicting blocks at the same height and the SCD blockchain will never fork. In the case of the one third faulty nodes are exceeded; the consensus process will fail and the master will be notified by the leader testifying that an agreement couldn't be reached.

3.3.3 SCD Blockchain synchronization

As a new block is validated and added to the blockchain, it triggers a need of an update process at the level of the SCD and the SCD blockchain on the nodes. The master will send the new block and also the new created security contracts with its signature to nodes of the SCD network. The SCD update process arranges the nodes in a linear fashion so that each node will only receive the message from its predecessor and send it to its successor. This process balances the load among the nodes making the replication achieve the best throughput possible. When a node receives the incoming message from the network, it will valid block according to its signature and then link it to the existing blockchain. To establish the link, the node will examine the incoming block header and look for the previous block hash to link it to.

Consistency and integrity checks are done periodically by nodes to guaranty that the SCD is in sync and have the same entries, especially when the network may fail to deliver the attended message or act in a malicious way. Therefore, the consistency and verification checks use a kind of Byzantine fault tolerant model where the node sends a request for the last replica verification to several nodes, and receives a respond with only the digest of the result. The digest helps the node to check the correctness of the result against the digest its replica while reducing network bandwidth and CPU overhead. If the node computes a different result from the designated requested nodes, it will send a notification to the master requesting the up to date SCD.

3.4 Application signature and identification

As aforementioned, the application tag (App-Tag) represents an identification injected on the payload of an egress packet of the application that sent a request/response to another VM over the network. This tag is considered as a key field alongside the tenant tag to authenticate the outgoing traffic, so that it gets accepted or denied accordingly by the receiving VM. For a packet to contain this App-Tag, the application needs to be identified when it sends a request or a response over the network. This identification process needs to be done before sending any traffic out of the VM. Therefore, an approach was set for this purpose to identify the application from its signature on the payload of the egress traffic, then and once identified, create an App-tag for the purpose of authenticating and authorizing the ingress traffic by the receiving VM.

The administrator is the one responsible of adding and attesting that an application is trustworthy. The process of adding an application to the database of trustworthy applications i.e. AppDB, is an iterative process that needs refining to get the application identification right through its signature. An application signature is simply a pattern within the outgoing packet from an application or a task. Therefore, the process of identifying the App signature starts with the analysis of the 6-tuple of the outgoing packet using layer 3 and 4 inspection, and going even to the seventh layer based on the application's unique characteristics, in order to achieve granularity of visibility and control over the egress traffic. The five tuples are a reference to six different values that comprise a TCP/IP packet, which are; source IP address, destination IP address, source port number, destination port number, and payload. The engine behind the AppID tag is driven by a series of pre-determined contexts. These contexts use decoders to help identify applications and ensure the success of proper layer 7 inspection at the packet load level. The identification of the application layer protocol adopts characteristics provided by the packet inspection module, which is mainly based on regular expressions of match recognition which not only improves the matching speed but also increases the accuracy of matching.

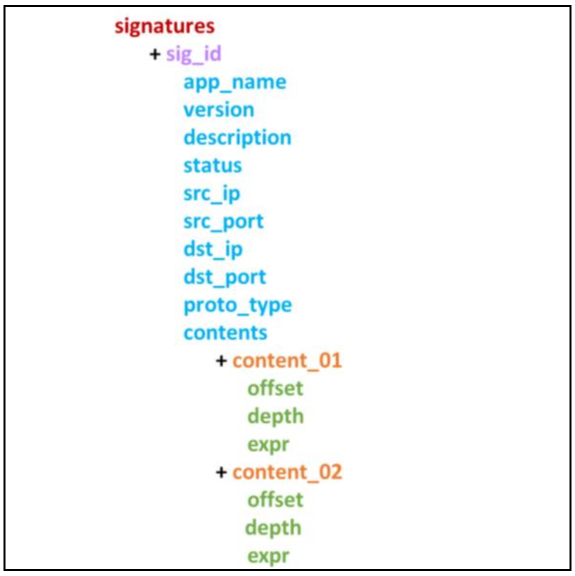
The packet inspection module relies heavily on Deep Packet inspection (DPI). DPI combines signature matching technology with the analysis of data in order to determine the impact of that communication stream, and identify the contents of each and every packet flowing through the network. It also takes packets apart to examines the data part of the packet, comparing it with a set of criteria, searching for pre-defined characteristics, making a decision based on the detected content, and then re-assembles the packet. In most use case of DPI, successful pattern matches are reported to a managing application (in our case the master agent – see the next section) for any appropriate further actions to be taken. The packet inspection process in our case is split in two phases. The first one is a shallow inspection or what is called also a stateless inspection. It focuses on a simple detection technique by only analysing the IP packet's source IP address, destination IP address, port source, port destination, protocol type. This phase renders the infrastructure's visibility limited only from layer to 2 to layer 4. Thus, the introduction of the second phase which is DPI, stems from the need of filling the void that the shallow inspection has, by pushing the visibility up to layer 7, and being able to see and understand the traffic up to that level. This is done adding functions that analyses the application layer which can identify the various applications and their contents on that bases, making this phase more like an application centric inspection.

As aforementioned, the detection of an application relies on the detection of its signature on the packet. In their most broad sense, signatures are pattern recipes which are chosen for uniquely identifying an associated application (or protocol) [34]. When a new application or protocol is added, it is analysed and an appropriate App-tag is generated and added to the AppDB. Those pattern recipes that represents a signature of a particular application are detected through a search for known sequences of bytes of for regular expression matches on the packet. In order to make this search faster, it is limited to only specific parts of the packet. The analysis of the packet is done in two ways; analysis by numerical matches and analysis by string matches. The analysis by numerical matches involves the search of numerical characteristics within a packet such as IP addresses, port numbers, payload length, etc. On the other hand, the string-matching analysis is a search for a sequence of textual characters, numerical, or even several strings distributed within the content of the packet, such as the protocol type or the payload.

The process of detecting and identifying an application should be as fast as possible, since a lot of packets will be flowing through this module. [35] has showed that almost all application signatures begin and end at the first 32 bytes of the payload, hinting that a lightweight approach using only a small portion of payload could be viable. The approach of lightweight payload inspection is not something new: In [36], the authors talk about the traffic classification approach of NetPDL [37] as "lightweight".

Our main focus for this module is at the payload level where the inspection is similar to what is done by the libprotoident library. The four bytes of the payload will be compared against a known signature for the application, making a rule that will include specific characters for all four bytes. However, the particularity of this phase is to only inspect the first outgoing packet and attempting to match it to the rules that have been set as a signature of a particular application. In fact, [35] has mentioned that the inspection of only few bytes in the first packet can still be successfully used for traffic classification by shifting the verification where the application protocol injects its headers.

As aforementioned, writing an application signature is an iterative process. Ann application signature should be precise and cover all the needed scenarios. Therefore, an administrator is required to test the crafted signature meticulously before committing it. In our case, the application signature comes in the form of a json file as shown on figure 10 that comprises the following fields:



```
signatures
  + sig_id
      app_name
      version
      description
      status
      src_ip
      src_port
      dst_ip
      dst_port
      proto_type
      contents
        + content_01
            offset
            depth
            expr
        + content_02
            offset
            depth
            expr
```

Figure 10. Structure of an app signature.

- **Sig_id:** Represents a unique ID for each signature application. Generally, this filed is generated automatically.
- **App_name:** Represents a simply the name of the application. It is also used in the logs to identify the application when it is being detected.
- **Version:** Represents the version of the signature. It is used as a revision for version control.
- **Description:** Represents a textual description of the application.
- **Status:** Represents the status of the signature. This fields can take three values:
  - o***Enabled***: If set, it means that the signature is enabled, and the agent is allowed to parse it.
  - o***Disabled***: If set, means that the signature is disabled for some particular reason. Therefore, the agent will not parse the signature since it won't find it on the scoped signature list. However, the signature will not be deleted.
  - o***Decommission***: If set, it means that the signature of an application is out of service. Therefore, the agent will never parse this signature again.
- **Src_ip:** Represents the source IP address that is being used by the application.
- **Src_port:** Represents the source port that is being used by the application.
- **Dst_ip:** Represents the destination IP address that is being used by the application.
- **Dst_port:** Represents the destination port that is being used by the application.
- **Proto_type:** Represents the protocol type that is being used by the application.
- **Contents:** Represents what should be detected within the payload that defines the application. The contents section may have as many content sub-sections as needed. The content entry is defined by the following parameters:
  - o***Offset***: Represents which byte will be the starting point on the payload where the search for a content should begin.
  - o***Depth***: Represents the offset where on the payload the search should stop.
  - o***Expr***: Represents the expression that the module will match against on the payload. It can be a normal expression as well as a regular expression. The usage of the offset and depth parameter allows a very specific matching that would help process the regular expression in a faster pace on a small section of the payload rather than the whole payload.

The application signature can be fine-grained to the extent of making an application signature specific to a certain type of traffic. For instance, a signature can be made specifically for a download traffic going towards a web application. Therefore, upload requests won't be going towards the web application, since they don't have their own signature, hence no App-tag created for this traffic.

It should be noted that the AppDB implements also the same features of the SCD blockchain. However, the database is only shared between the master(s) and the appropriate VM that hosts those trusted applications, rather than being shared with the whole VMs within the tenant.

3.5 Policy Management

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

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 Security Contract, inbound packets...). The output of the policy will be one of two actions -Allow, or Deny-. If the output of the policy is discarded, the packet is dropped or rejected according to the configuration done by the admin. Otherwise, the packet is passed up to the next layer for further processing. The protection afforded by the agents is defined by a database called the PM Database (PMdb). This database is maintained by an administrator who interacts with it for all policy-related managements. This module is checked by the agent to define the required processing of each outbound or inbound packet. This processing is generally, either choosing a particular security contract for an egress traffic, processing a traffic as an inter-VM one, allowing the traffic to its destination, or applying deeper processing. In order to reduce the overhead of checking those policies, there is a policy cache where most recent loaded polices are stored.

When a traffic occurs, IP packets are queued until being checked against the PM, where the following actions are performed:

- If the policy indicates a packet needing to be denied, then the inbound IP packet is expected to be dropped or rejected. When a packet is refused, 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 sent to the administrator. Dropping the traffic can also be done at the vSwitch level, by send a rule add to the control of the SDN network if implemented.
- If the policy indicates that the packet can be transmitted without any extra security processing, this means that the traffic is not inter-VM and the policy engine will allow the traffic to be transmitted without any processing nor frame tag injection.
- If the policy indicates an inbound/outbound inter-VM traffic, the agent will be required to process the traffic accordingly, which will be detailed in the next section.

Unlike SCD, the PMdb is not a shared database, but rather local and private to each agent/VM.

3.6 The Agent

The goal of the proposed approach is mainly to target some specific fields within the incoming/outgoing packets, in order to be analysed and act on them accordingly. Therefore, we thought about using a comprehensive agent implementation embedded in every machine of interest [38]. Those agents are the backbone of the proposed approach in order to have a complete working cycle of an inter-VM exchange. The agents include similarities of a minimal and lighter IDS or firewall functionalities, since it has an essential aspect of access control as well as filtering. They are primarily responsible for enforcing the policies set by an administrator within the PMdb, and also crafting all the needed fields accordingly. As IP packets flow into a VM, the agent doesn't perform a whole packet analysis, but rather targets only specific fields of the IP packets and responds by an ACCEPT, DROP, REJECT or FORWARD, according to the PMdb and the AppDB.

The architecture of the proposed approach can be can be viewed as a manager (master)/slave kind of architecture. However, the manager is not considered as a single point of failure, since the slave agents can function without any problem even though the manager is unreachable. Hence, the role of the manager can be considered as frontend offering an entry point to manage the slave agent cluster, its different features, databases, and blockchain. Consequently, when a worker agent is freshly deployed, it is configured to make a first contact with its manager and launch the discovery and synchronization process with the other agents. The communication between the manager and the slave is done in a secure channel manage by a passwordless ssh connection, in order to prevent rogue requests to the manager or having some VMs impersonating the manager. Key rotation is used as a standard mechanism to ensure security of the key pairs, by periodically changing the encryption keys to thwart any attacks if the keys have been compromised. When a change of those key pair occurs, the manager sends a request to all of his slave agents in order to update their keys. The communication between the manager and its slaves is different from communication of the slave between them. When the agent receives a request from the manager, the packet holds a special management frame tag containing a masterTenant tag and matserApp tag, and there is no SC negotiation due to the encrypted channel between them. The manager has a mapping service that is used to document all the information about the slave agents. This service publishes IP addresses, identity certificates and the mode of each agent. All the slaves announce their presence by registering to this service when they first start up.

Additionally, the agent has a within its PMdb a mode that is shared with its neighbouring agents within the same tenant via the manager’s mapping service. This agent mode can in fact take two distinctive values; full and quick. The administrator is responsible of defining the right mode for each machine/agent. This mode would have a direct impact primarily on the processing of an outbound traffic. Because, when a receiving machine/agent has is in quick mode for instance, this means that the process of deep packet inspection for the application signature detection would be skipped and the agent will only send only a tenant tag with the added frame field. Thus, making the inter-VM communication much faster. Conversely, if the agent is in full mode, the processing of the outbound packet will be in done in full and phase will be skipped.

In order to have an optimal agent' behaviour on what action should be done when a packet is received, we introduced a flag at the beginning of the payload. This flag is a set of bits that is passed between agents in order to perform various actions depending on its value. Thus, it defines the type of data carried within the payload by taking one of these values:

- **Enrolment flag** – occurs when a new agent is added to the cluster. The manager will try to sync the slave agent with all the needed data.
- **Discovery flag** – is exchanged between agents when enrolled in order to be discovered by other peers on the network and be added as well on their respective lists.
- **Update flag** – is generally sent by the manager to signal an update is occurring. This can be at the level of the policy management database, security contract database, etc
- **Handshake:** is considered as a flag initiating a new communication, letting the receiving agent what fields to expect within the payload.
- **Data flag:** is simply a flag indicating that the payload holds an application data.

At the first run of a slave, it will send a request to its manager in order to retrieve its related information and be in sync. A tenant can contain several machines and accordingly several agents. Hence, a freshly deployed slave agent needs to recognize its neighbouring agents within the same tenant, as well as the sync of the shared ledger and databases. Therefore, comes the role of the flag field. As mentioned before the flag affects the agent' behaviour, when a slave agent is newly deployed it sends a request to its manager in order to enrol itself and receive its tenant tag, information about its neighbouring agents (IP address and trust level),

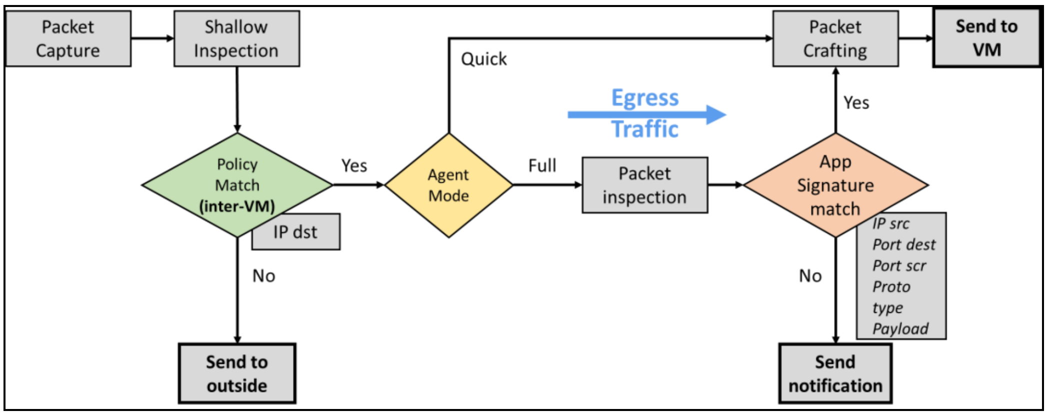


Figure 11. Egress traffic processing workflow.

populating its SCD with the SCs created so far and sync its blockchain consequently. The sync basically is done via gossip protocol to ensure that data is routed to all members. After this, the new salve agent sends a message with a discovery flag, so that the neighbouring agents within the same tenant add it to their list of agents. The mode of a new agent is set by default to quick mode but it can be changed via the console management on the manager agent by the admin if needed. When a change of the mode occurs, the manager agent sends a message with an update flag to the designated slave agent in order to update its mode. Consequently, this designated slave by its turn will send a message to its peers with a discovery flag so that they update their list of agents.

The implementation of distributed agents comes to relieve the hypervisor from a bottleneck filtering, introspection which is a technique that helps monitor the state of VM's running on a hypervisor [39], and other security mechanisms. Hence, adding another security level, but with an enhanced focus on the tenant and its VMs, by adding another layer of application-based authentication via the frame tags. The proposed agents act like a light weight security gateway or an IDS, more specifically like a Stack based IDPS as if it was an independent device, where the packets are examined as they go through the TCP/IP stack and, therefore, it is not necessary for them to work with the network interface in promiscuous mode. The protection offered by this approach is based on requirements defined by the security policy (trusted applications, mode, SC, PMdb) established and maintained by an administrator, which afterward is translated to an allow or a reject of the traffic accordingly. The integrity of the agents is of the utmost importance. Hence operating wise, they are executed in a privileged domain which the Ring 0. Subsequently, the agents would be safer from being controlled by any malicious application running within the VM or any unauthorized users, and also enabling high performance input/output at the network level.

To sum up, the agent interacts closely with the transport, network layer and application layers. It is designed to efficiently implement the following capabilities:

- Ability to inject fields at the beginning of the payload to outbound packets via packet crafting.
- Ability to analyse and inspect packets via DPI then decapsulate the payload from the injected filed(s) in the case of inbound traffic and send it to the appropriate application.

3.6.1 Egress/ingress inter-VM traffic workflow processing:

When an outbound traffic flows within an agent (see figure 11), its workflow goes through several stages which are governed by a set of policies that needs to be enforced.

The first station that the packet arrives to through the packet capture is the shallow inspection. This inspection targets only the IP header, and more specifically the IP destination. The IP destination matching has two scenarios; either the traffic is going outside of the network of the agent or it is an inter-VM traffic i.e. an IP destination address of another agent within the same tenant. In the case of the later, the agent will check the mode of the destination agent. If the receiving agent is in quick mode, the traffic will go directly to the packet crafting, where only the tenant tag will be injected. Then, the packets get to be sent to the destination VM. However, if the agent is in full mode, the traffic will go the packet inspection stage, where the application within the traffic will be matched against the application signatures database. If matched with a signature, the traffic will go to the packet crafting in order to inject the tenant and the application tags unlike the quick mode. Then, the traffic gets to be sent to the destination VM. Moreover, if the application didn't match any signature during the packet inspection, the traffic will be dropped, logged and a notification will be sent to the admin for further investigation, which mostly can be due to an application signature testing, a new application, a malicious application or simply a false positive, etc.

Similar to the outbound traffic, inbound traffic (see figure 12) goes through the same main station as well to enforces different policies. First, and via the packet capture, the traffic will flow through the shallow inspection to verify its origin by verifying its IP source. In the case of a traffic coming from the outside of the agent' network, the packets will be sent directly to their destination. However, in the case of a traffic coming from a peer agent within the same tenant, the traffic goes through the next phase which is packet inspection. At first, the inspection goes through injected payload flag and check its nature. In the case of a flag value different from the data flag, the packet will be sent to the

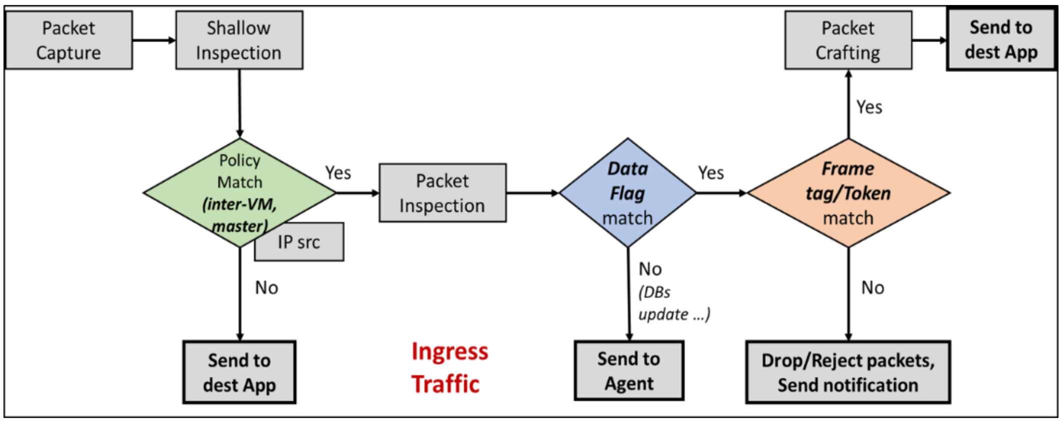


Figure 12. Ingress traffic processing workflow.

agent. Otherwise, the inspection will translate to the next field, which is the frame tag or session token to be matched. The session token matching is pretty straightforward, where the token will have to match to the saved current token session. As for the frame tag, the inspection in this case depends on the agent' mode. Since during the quick mode, the injected field in only the tenant tag, thus it will be the only one to be verified and matched. However, in the case of a full mode, the inspection will have to go through the two fields of the frame tag; the tenant and application tag. Once a successful matching occurs, the traffic will go to the packet crafting phase, where the packets will get stripped of the injected fields at the level of the payload. On the other hand, if the frame tag or the token didn't match, will result with a refusal of the inbound traffic and send a notification to the admin. Hence, the packets and according to the admin' configuration, can be dropped or rejected. When rejected, the agent sends an explicit notification back to the sending agent of a closed connection. However, when the packets are dropped, the agent will simply discard the packets and sends no response to the sending agent. If DROP is chosen as a strategy for refusing noncompliant packets, the packets and with the help of SDN, can be dropped at the level of the vSwitch, thus reducing the overhead of processing knowingly refused packets.

3.6.2 Packet crafting:

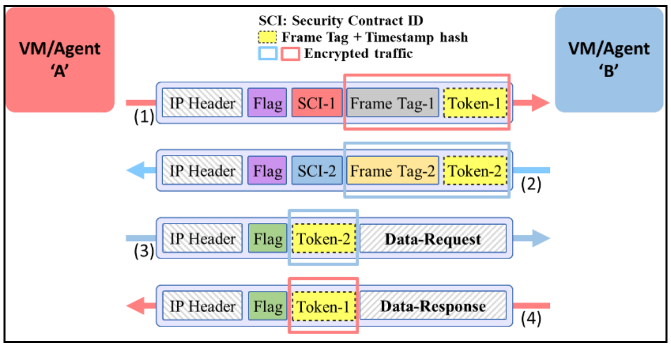


Figure 13. Inter-VM Packet crafting processing.

The packet crafting phase as shown on figure 13, is not straightforward as its preceding ones, since its processing can be a bit different depending on the inter-VM exchange stages. When an inter-VM communication occurs, there is the initiation of two distinct handshakes from the communicating agents that needs to take place first. Those handshake (Figure ,13 -1-) (Figure ,13 -2-) are in the form of a payload that encompasses relative fields and data in regards to the current opened communication session. The first injected field during the handshake phase is a flag. The value that this flag is going to hold in this case is 'Handshake', since it is a communication initiation between the two agents with the negotiated information. After that, comes the security contract ID, which is an ID of the SC chosen (randomly or according to the policy) by VM/Agent 'A' letting the VM/Agent 'B' know which encryption key and algorithm that will be used to communicate with it. As aforementioned, the SCs are stored in a shared database between the agents, therefore the VM/Agent 'B' will refer to the SCI-1 and use it to encrypt the communication with VM/Agent 'A' accordingly. It should be noted that the VM/Agent 'B' will also inject its own security contract ID so that the VM/Agent 'A' know in its turn which encryption keys and algorithm that will be used to communicate with it. The injection of the frame tag comes next, which its structure depends on the agent 'B' mode. In the case of a quick mode, only the tenant tag will be injected, however if it is a in full mode, then both tenant and application tag will be injected as the frame tag filed. Once those three first fields are created, they will be combined with a time stamp and get hashed to serve as the token for the current session. When the token is created, this triggers the encryption of the frame tag and token according to the chosen security contract (SCI-1). Similarly, and since it is a bi-directional handshake, The VM/Agent 'B' will encrypt its generate frame tag and token according to the chosen security contract (SCI-2). The randomness of choice of the security contract is reflected directly on encryption of the frame tag and the token, which are considered as key fields for the integrity of the communication between the two agents. This randomness would also help to hinder the likelihood of payload tampering by a malicious user or a man in the middle.

When an agent receives a packet having a handshake flag, it triggers the process of a communication initiation for the receiving agents. Consequently, the agent will then determine which security contract has been chosen in order to decrypt properly the frame tag and the token, then authenticate the frame tag and store the token for a further verification.

Once a successful handshake is reached, the agent 'A' will send the application' request data to VM 'B'. However, this time the injected flag will hold a different value which will be DATA, since it is simply data that is being communicated from a trustworthy application hosted on a VM to another

VM. In addition to the flag, the packet crafter will encrypt the previously generated session token (token-1) as a signature and inject it to the payload. This will help the receiving agent to not only authenticate the traffic but to authorize it as well by checking the value of the session token with the stored and exchanged one from the handshake phase. Once verified and validated, the agent will decapsulate the payload from the flag and the token, then send the data to the application of interest. The communication between the two VMs/Agents from now on will take the same format, which is injecting the DATA flag and their respective session token to the payload.

The lifespan of the session is determined by the 'busy/idle' lifetime of the security contract which was determined during its creation. Once a lifetime is reached, a connection handshake must be renegotiated. In order to have an optimal communication session between the VMs/Agents, it is recommended to configure the policy of the agent on what security contracts to be used (i.e. an optimal busy/idle lifetime) for a specific traffic going towards a particular agent/VM/application.

3.6.3 Agent implementation:

The agent is the cornerstone of the proposed approach since it does implement the different blocks of packet processing and make them stick together in a cohesive environment. The implementation of the said agent should take under consideration a huge factor, which is traffic latency because all the ingress/egress traffic will flow through the agent.

In a general sense, each packet is processed layer by layer. For instance, in an ingress scenario, the network flow goes through the network card first so that it will be sent to RX, or a receive queue. Then, the packet gets copied to the main memory via the Direct Memory Access (DMA) mechanism. After that, the system has to be informed that a new packet is going through, and pass the data onto an allocated buffer. This is a special buffer that Linux systems allocate for every packet, and in order to do so, Linux uses an interrupt system which is generated several times when a new packet enters the system. The network stack in Linux offers a complete implementation of the TCP/UPD protocols, as it was showcase more in details of this implementation in [40]. After going through the transport layer, the packet can finally be delivered to the userspace by taking on the data that got copied from the buffer using the NAPI [41], last visited: July 2014 to start a poll loop if one was not running already, in order to have a full access to that data by the application that should receive the packet.

As aforementioned, the implementation of the agent has to take under consideration major latency issues during the processing of flowing packets. However, by looking at the normal flow of a packet and how it is processed, it became apparent that there are some issues that might hinder the proper functioning of the agent. For instance, the network card works in interrupt mode that might affect severely the overall agent performance as well as the system. When a packet enters the network interface, it registers itself in a poll queue and disables the interrupt. Consequently, the system periodically checks the queue for new devices and gathers packets for further processing. As soon as the packets are processed, the card will be deleted from the queue and interrupts are again enabled. Additionally, in a virtual environment, the VM Kernel is relying on the physical device to generate these interrupts to process network inputs/outputs. Therefore, the processing in a VM will suffer from additional delays on the entire data plane from the physical network interface to the guest machine. Another issue, is related to the buffer that is allocated for each packet and becomes free each time a packet enters the userspace. This operation does consume a lot of bus cycles, since there is a frequent data transfer from the CPU to the main memory. Adding to this, the Linux network stack was designed to be compatible with as many protocols as possible, which makes all of their metadata get included in the buffer for the purpose of processing the packet. All of these metadata are not necessary for processing specific packets making it slower than it could be. In addition to this, context switching also affects negatively the performance. This context switching happens when an application in the userspace needs to send or receive a packet. The application has to execute a system call, which means a switch to the kernel mode and then back to the user mode.

The implementation architecture of the agent has to take under consideration the aforementioned issues and mitigate them. Therefore, one of the architectures taken under consideration for the agent is fast path architecture, where the data plane is split into two layers. The first layer is called fast path. It is a layer that processes the majority of the ingress traffic that is coming outside of the OS, without suffering from the OS overheads that decrease the overall performance. The second layer resides on the OS networking stack, and take on only the packets that require complex processing. This layer performs the necessary and needed operations on the packet.

This fast path architecture influenced the implementation architecture of the agent to be split in tow: an agent backend and an agent frontend as depicted on figure 14.

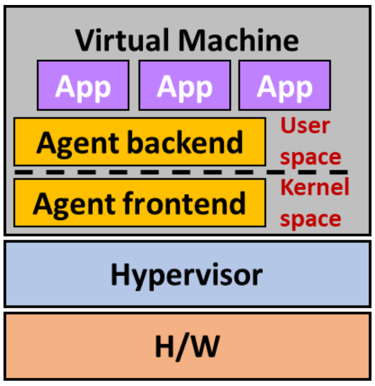


Figure 14. Agent (front and back end) placement.

The frontend agent primarily performs inputs/outputs operations via the physical device and also accepts inputs/outputs requests from the backend agent. Additionally, it reduces the per packet system call and also focuses on moving packets from the kernel space towards the userspace i.e. the backend agent. In contrast, the backend agent accepts inputs/outputs requests from the kernel space i.e. the frontend agent, as well as transferring them back to it. It is also responsible for preforming packet processing as aforementioned in the section before, and packet generation according to the enforced policies against the egress/ingress traffic.

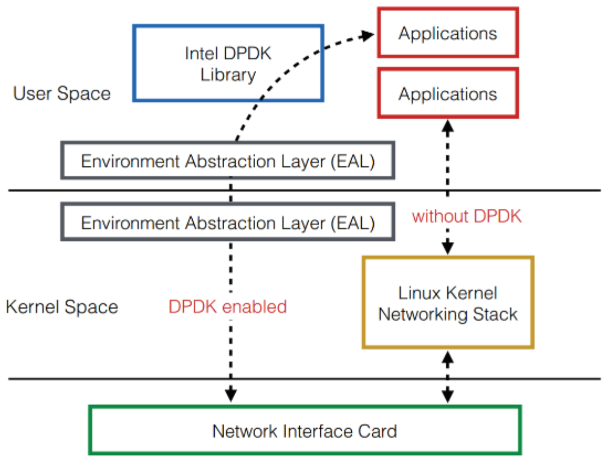


Figure 15. Architecture of Intel Dataplane Development Kit (DPDK). [42].

To help with implementation of the fast path architecture on the agent, the choice has fallen on Intel Data Plane Development Kit (DPDK) [43] [44]. In fact, DPDK enables fast packet processing for data plane applications. As it is shown in figure 15, its architecture provides a set of data plane libraries and network interface controller drivers to have basic Linux network stack functions for fast packet processing, as well as the optimization for memory/buffer allocation and mapping. DPDK provides a programming framework for several processor architectures, and enables faster development of high-speed data packet networking applications. This can help to write networking applications i.e. the agent that work entirely in the userland with no system calls in order to bypass the heavy layers of the Linux kernel networking stack and communicating directly to the network hardware. Additionally, DPDK implements a low overhead run-to-completion model for fast data plane performance and accesses devices via polling to remove the latency of interrupt processing at the trade-off of higher CPU consumption. It should be noted that polling is very advantageous when interrupts are frequent, since there will be a noticeable overhead associated with each interrupt due to the aforementioned back and forth switching from the user mode to the supervisor mode. In the context of a VM, the driver and the application will constantly busy-looping for an input/output to be available. Therefore, an application in the guest OS can process the Inputs/Outputs near realtime instead of waiting for an interrupt to happen. This in fact will enable a lower latency and a higher Packer Per Second rate. On the other hand, each poll is generally only a check for a value on a specific memory address. Consequently, the agent implements via DPDK the Poll Mode Drivers (PMD). The PMD consists of APIs that configure the devices and their respective queues. It accesses the RX and TX descriptors directly without any interrupts to receive, delivers and process packets in a faster pace.

By implementing DPDK in any application, the massive network traffic will usually be handled through Environment Abstraction Layer to have fast access to hardware and memory. The Environment Abstraction Layer (see figure 15), or EAL is the main concept behind the DPDK. EAL is a set of programming tools that let DPDK gain access to lower-level resources such as hardware and memory space. It offers a generic interface that hides the details of the environment and provides a standard programming interface. It is also responsible of a frequent initialization to decide how to allocate resources such as memory space, PCI devices, timers, etc. Common use cases are around special solutions for instance Network Function Virtualization and advanced high-throughput network switching that we can find in SDN. Technically, EAL achieves physical memory allocation by using mmap () in hugetlbfs through the usage of huge page sizes to increase performance. In fact, it is what binds DPDK to applications, since they must include its header files. The most commonly of these include:

- **rte_lcore.h** — manages processor cores and sockets;
- **rte_memory.h** — manages memory;
- **rte_pci.h** — provides the interface access to PCI address space;
- **rte_debug.h** — provides trace and debug functions (logging, dump_stack, and more);
- **rte_interrupts.h** — processes interrupts.

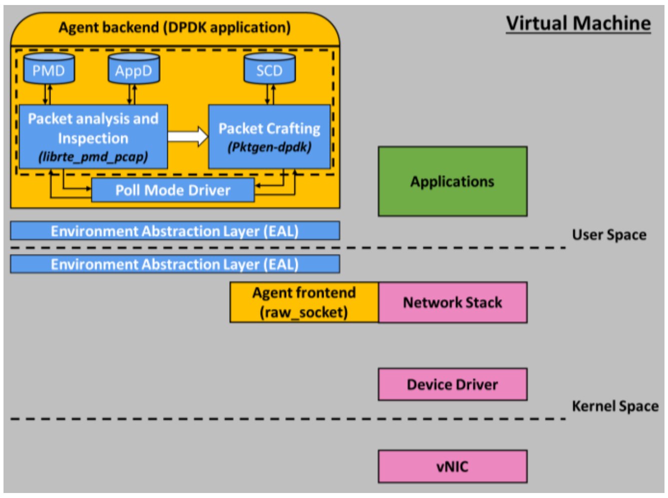

Agent backend (DPDK application)
Virtual Machine
PMD
AppD
SCD
Packet analysis and Inspection (librte_pmd_pcap)
Packet Crafting (Pktgen-dpdk)
Poll Mode Driver
Applications
Environment Abstraction Layer (EAL)
User Space
Environment Abstraction Layer (EAL)
Agent frontend (raw_socket)
Network Stack
Device Driver
Kernel Space
vNIC

Figure 16. Agent implementation through DPDK in a VM.

The main part of the agent as shown on figure 16, runs in the userland using the pthread library. Additionally, PCI information about devices and address space are discovered through the /sys kernel interface and also via kernel modules such as uio_pci_generic, or igb_uio, etc. The sheer amount of processing that the agent has to do is backed by the DPDK, since it can achieve a very low latency by completely bypassing the kernel layer, where the PMD quickly delivers them to the agent making the TX path as well the RX path are equally fast.

The first station that a packet goes through in the agent is the ring buffer that acts as a receiving queue, where the agent periodically checks that buffer for new incoming packets. Packets received in the DPDK are also sent to a queue implemented on the rte_ring library. In the case of packet descriptors existing within the said buffer, the agent will refer to DPDK packet buffers in the specially allocated memory pool using the pointers in the packet descriptors. However, in the case of an empty buffer, the agent will queue the network device under the DPDK and then refer to the ring again.

The adoption of DPDK for the agent helps tremendously to implement its two major backend functionalities: packet inspection and packet crafting. The packet capture/analysis relies on a libpcap-based PMD, and more specifically on the librte_pmd_pcap as shown on the figure 16, that reads and writes packets using directly libpcap. When the agent wants to start capturing packets, the library registers a callback-function into the PMD to capture from, whether it is the RX or TX PMDs or together. This is where resides the aforementioned packet inspection that enforces the detection of application according to their signatures. When an inter-VM traffic occurs via a trusted application, the packets need to go through the packet crafting module in order to unencapsulate the payload from the frame tag. This feature is implemented as shown on the figure 16 via pktgen-dpdk, which is a traffic generator that is powered by the DPDK fast packet processing framework. The pktgen-dpdk is capable of generating 10Gbit wire rate traffic with 64-byte frames in sequence by iterating IP addresses, MAC or ports destination or source. It also can handle packets with several protocols such as MPLS. GRE, TCP, UPD, ARP, ICMP, etc. At the frontend agent side, there is the usage of raw_socket to receive data packets and send them to the backend agent by bypassing the normal TCP/IP protocols, offering a sort of a fast lane for packets to be sent and processed by the backend agent.

4. Conclusion

CC has emerged as a promising IT services provisioning paradigm. It encompasses many technologies including networks, virtualization, operating systems, resource scheduling, databases, transaction management, etc. Therefore, security issues for many of these systems and technologies remains very much current issues in a CC environment. For instance, VLAN isolation could be bypassed via several techniques. For those techniques, we could find VLAN hopping, which is basically attacking a host on a VLAN to gain access to traffic on other VLANs that would normally not be accessible. There is also VM jumping, that exploits vulnerabilities in hypervisors that allow malware or remote attacks to compromise VM separation protections and gain access to other VMs. Consequently, our proposal addresses mainly the inter-VM traffic visibility and authentication by proposing a protocol that processes and controls such traffic.

The approach relies on introducing a frame structure at the payload, to fill the security gaps where mostly the isolation breach occurs. This 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. The processing of such frame tag is done through embedded agents within the machine of interest which are able to generate, capture and analyse this said frame and respond to it by an automated acceptance or refusal. The implementation of the agent was done with DPDK to ensure a high throughput due to the amount of traffic that needs to be processed by it. Additionally, to ensure the integrity and security of the frame tag, security mechanisms such blockchain and security contract we put and place, complimented by a policy directory to govern the overall all process. Since the authentication in our proposal is primarily application centric, we also introduced a way of detection applications in an exchanged IP packet relying on application signature mechanisms. In order to make the approach informant, the agent logs and events can be fed to a Security Information Even Management [45] for a thorough and better response in the case of breaches or malicious behaviour. An adaptation of this approach can be envisioned as future work for not only VMs but rather extending it to containers as well.

References

[1] ISO/IEC JTC 1 SC38: Cloud Computing Overview and Vocabulary. International Organization for Standardization, Geneva, Switzerland.

[2] R Bhadauria, S Sanyal. Survey on security issues in Cloud Computing and Associated Mitigation Techniques. Int J Comput Appl (0975-888); 47(18), 2012

[3] Irfan Gul, M. Hussain, “Distributed Cloud Intrusion Detection Model”, International Journal of Advanced Science and Technology, Vol. 34, September, 2011

[4] 3 Ways to Secure Your Virtualized Data Center, Jull 29, 2010, http://www.serverwatch.com/trends/article.php/3895846/3-Ways-to-Secure-Your-Virtualized-Data-Center.htm, Retrieved 2016-05-20.

[5] A comprehensive framework for securing virtualized data centers, HP,Aug 2010.

[6] S. L. and Z. L. and X. C. and Z. Y. and J. Chen, S.Luo, Z. Lin, X. Chen, Z. Yang, and J. Chen, ”Virtualization security for cloud computing service” in International Conference on Cloud and Service Computing (CSC), pp. 174-179, 2011.

[7] “Cloud Security Alliance Guidance Version 3.0 ”, Cloud Security Alliance, 2011

[8] SDN: Development, Adoption and Research Trends, Lav Gupta, Washington University in St. Louis, Retrieved 2018-10-20. Retrieved 2017-02-17.

[9] The Northbound API- A Big Little Problem, http://networkstatic.net/the-northbound-api-2/,

[10] Magic of SDN in Networking, Calsoft Labs, Santha Rami Reddy, Retrieved 2017-02-17.

[11] Restructuring and Innovation in Banking. Scardovi, Claudio. Springer. p. 36, 2016

[12] A. Antonopoulos, Mastering Bitcoin. O'Reilly. 2014

[13] R. Chan, Consensus Mechanisms used in Blockchain, https://www.linkedin.com/pulse/consensus-mechanisms-used-blockchain-ronald-chan/, Retrieved 2018-04-16.

[14] D. Cawrey, How Consensus Algorithms Solve Issues with Bitcoin’s Proof of Work, http://www.coindesk.com/stellar-ripple-hyperledger-rivals-bitcoin-proof-work/, Retrieved 2018-05-22.

[15] V. Buterin, Proof of Stake: How I Learned to Love Weak Subjectivity, https://blog.ethereum.org/2014/11/25/proof-stake-learned-love-weak-subjectivity/, Retrieved 2018-03-12.

[16] A. Naumoff, Why Blockchain Needs ‘Proof of Authority’ Instead of ‘Proof of Stake’, https://cointelegraph.com/news/why-blockchain-needs-proof-of-authority-instead-of-proof-of-stake, Retrieved 2018-01-11.

[17] Practical byzantine fault tolerance.Castro, & Liskov. 3rd Symposium on Operating Systems Design and Implementation, pp173-186. 1999

[18] G. Becker, R. Bochum, Merkle Signature Schemes, Merkle Trees and Their Cryptanalysis.Becker, p. 16, 2008

[19] B. Wahyudi1, K. Ramli, H. Murfi, Implementation and Analysis of Combined Machine Learning Method for Intrusion Detection System, International Journal of Communication Networks and Information Security (IJCNIS) Vol. 10, No. 2, August 2018

[20] A. Vahid Dastjerdi, A. Kamalrulnizam, Sayed Gholam Hassan Tabatabaei, Distributed Instrusion Detection in Clouds Using Mobile Agents, Third International Conference on Advanced Engineering Computing and Application in Sciences, October 11-16, 2009

[21] A. Bakshi, Yogesh B. Dujodwala, ”Securing Cloud from DDoS Attacks Using Intrusion Detection System in Virtual Machine”, Proceedings of the 2010 Second International Conference on Communication Software and Networks(ICCSN ’10), P 260-264, 2010.

[22] A. Azuan, I. Norbik Bashah, Mohd Nazri Kama, CloudIDS: Cloud Intrusion Detection Model Inspired by Dendritic Cell Mechanism, International Journal of Communication Networks and Information Security (IJCNIS) Vol. 9, No. 1, April 2017

[23] A. Schulter et al., Intrusion Detection for Computational Grids, Proc. 2nd Intl Conf. New Technologies, Mobility, and Security, IEEE Press, 2008

[24] Kleber, schulter, Intrusion Detection for Grid and Cloud Computing, IEEE Journal: IT Professional, 19 July 2010

[25] Irfan Gul, M. Hussain , Distributed Cloud Intrusion Detection Model , International Journal of Advanced Science and Technology , Vol. 34, September, 2011

[26] C. Mazzariello, R. Bifulco, R. Canonico, Integrating a Network IDS into an Open Source Cloud Computing Environment, IEEE sixth international conference on Information Assurance and Security, 2010

[27] P Mishra, E S Pilli, V Varadharajan et al., "Intrusion detection techniques in cloud environment", Journal of Network & Computer Applications, vol. 77, no. C, pp. 18-47, 2017

[28] P. Deshpande, SC. Sharma, SK. Peddoju, S. Junaid, HIDS: a host-based intrusion detection system for cloud computing environment. Int J Syst Assur Eng Manag 9(3):567–576, 2018.

[29] H. Jin, G. Xiang, D. Zou, S. Wu, F. Zhoa, M. Li, A VMM-based intrusion prevention system in cloud computing environment. J Supercomput 66(3):1133–1151. 2013.

[30] P. Padmakumari, K. Surendra, M. Sowmya, M. Sravya, Effective intrusion detection system for cloud architecture. ARPN J Eng Appl Sci 9(11):2135–2139

[31] A. V. Dastjerdi, K. Abu Bakar, and S. Tabatabaei,” Distributed Intrusion Detection in Clouds Using Mobile Agents,” in Third International Conference on Advanced Engineering Computing and Applications in Sciences,p. 175-180, 2009.

[32] S. Kentl, K. Seo, Security Architecture for the Internet Protocol, RFC 4301, 2005

[33] A. Freier, P. Karlton, The Secure Sockets Layer (SSL) Protocol Version 3.0, RFC 6101, 2011

[34] Digging Deeper into Deep Packet Inspection (DPI), Datakom, 2017

[35] G. Aceto, A. Dainotti, W. de Donato, and A. Pescap, “PortLoad: Taking the Best of Two Worlds in Traffic Classification,” in IEEE INFOCOM 2010, 2010

[36] F. Risso, M. Baldi, O. Morandi, A. Baldini, P. Monclus, “Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation,” in ICC’08, pp. 5869–5875, 2008.

[37] Computer Networks Group at Politecnico di Torino, “The NetBee Library,” http://www.nbee.org/, Retrieved 2018-02-27.

[38] E.H, Spafford and D. Zamboni, Intrusion Detection Using Autonomous Agent, Computer Networks, vol.34, issue 4, 2000.

[39] A, Tapaswi S, Virtual machine introspection: towards bridging the semantic gap. Journal of Cloud Computing, 2014.

[40] M. Rio et al.: A Map of the Networking Code in Linux Kernel 2.4.20, in: Technical Report DataTAG-2004-1, 2004

[41] Linux man page: socket, http://linux.die.net/man/7/socket, Retrieved 2017-04-15.

[42] Chen-Nien MaoMu-Han HuangSatyajit Padhy,Minimizing Latency of Real-Time Container Cloud for Software Radio Access Networks, Conference: 2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom), 2015

[43] D. Scholz, “A look at Intels dataplane development kit,” Network, vol. 115, 2014

[44] “DPDK web page,” http://dpdk.org/, Retrieved 2017-12-08.

[45] L. Fetjah, K. Benzidane, H. El Alloussi, O. El Warrak, S. Jai-Andaloussi, A. Sekkaki, Toward a big data architecture for security events analytic, 2016 IEEE 3rd International Conference on Cyber Security and Cloud Computing (CSCloud), 2016