# A Framework for Realizing Security on Demand in Cloud Computing

Pramod Jamkhedkar, Jakub Szefer, Diego Perez-Botero, Tianwei Zhang, Gina Triolo and Ruby B. Lee
Department of Electrical Engineering
Princeton University, Princeton, NJ, USA.
{pjamkhed, szefer, diegop, tianweiz, gtriolo, rblee}@princeton.edu

***Abstract*—In this paper we present our vision for *Security on Demand* in cloud computing: a system where cloud providers can offer customized security for customers' code and data throughout the term of contract. Security on demand enables security-focussed competitive service differentiation and pricing, based on a threat model that matches the customer's security requirements for the virtual machine he is leasing. It also enables a cloud provider to bring in new secure servers to the data center, and derive revenue from these servers, while still using existing servers. We show a framework where customers' security requests can be expressed and enforced by leveraging the capabilities of servers with different security architectures.**

***Index Terms*—Virtual Machine Security; Cloud Computing; Threat Models; Dynamic Provisioning; Hardware-Software Security Architectures; Live Migration; Trust Evidence.**

## I. Introduction

Just as cloud customers have different performance needs, they also have different security needs for their computations done in the cloud. For example, some customers insist on guarantees of the confidentiality of their sensitive data or proprietary code, while other customers are more concerned about the integrity of the programs and the computed results, while still others require availability of service as their top priority. Some customers worry about malicious virtual machines (VMs) being co-resident on the same cloud server as their VMs, while others would like the cloud server to help them defend against possible vulnerabilities in the commodity guest operating system (OS) that they may employ in their VMs. The present service level agreements (SLAs) and cloud implementations do not allow such flexibility in security specifications [1]–[3]. At best, they provide a rigid, small set of features [4], such as malware detection and monitoring [5] or encryption [6], which are applied uniformly to all customers.

This paper presents a *Security on Demand* (SoD) framework for infrastructure as a service (IaaS) clouds, which provides on-demand, customized VM security for the cloud customers. The framework is based on three primary design goals: 1) enable customers to request customized security for their VMs, 2) enable different secure server architectures to service these customized security requests, and 3) provide persistent security for each VM throughout its life-time in the cloud.

In the security on demand framework, we propose a novel mechanism for allowing cloud customers to choose among a range of security options, where each option is based on a specific underlying threat model. The capability to offer customized VM security requests is based on allowing the cloud customer to select his or her choice of threat model, which he or she finds most suitable for the VM. For instance, a customer requesting a VM for processing highly secretive financial data, will prefer the data to be protected from most entities in the cloud environment such as the hypervisor, other VMs, etc., and will require a threat model in which very few entities are trusted. On the other hand, a customer requesting VM for processing ordinary photographs may choose a less restrictive threat model.

Another feature of the security on demand framework is the manner in which we leverage existing secure server architectures to service these highly customized VM security requests. At present, there does not exist a single secure server architecture which provides desired security protections under all the different threat models. However, numerous secure hardware-software architectures have been proposed [7]–[19], and some are commercially available [20]–[22], which can provide security protections very well under different, specific threat models. Our SoD framework would enable a cloud provider to afford to have different types of secure servers in a datacenter, each capable of providing a desired type of security. A cloud provider can then select the most appropriate type of secure server, which can satisfy the security options selected by the customer. A key challenge in servicing such requests is to define a common representation to which various types of security options, and the security enforcement capabilities of different server architectures, can be accurately mapped. Once such a mapping is in place, a secure server type can be easily selected for a customer's security request by matching the appropriate parameters. In this paper, we define such a common representation and the manner in which the mapping can be carried out.

Finally, in order to provide lifetime VM security, we propose a novel use of live VM migration. We observe that the security enforcement process for a given VM can be disturbed, primarily by two events: 1) change in the security level requested by the cloud customer, and 2) an attack on the server on which the VM is running. When either of these events occur, the VM is remapped to a new server capable of satisfying the new security requirements, and it is live-migrated to the newly selected server. This process is carried out throughout the lifetime of the VM to ensure persistent security. In this paper, we analyze the feasibility of such an

approach by testing the average time and downtime required for live VM migration under different workloads. In order to demonstrate the practicality of the complete security on demand framework, we have implemented a prototype by modifying the OpenStack cloud system.

The security on demand framework offers numerous advantages in addition to tailored VM security. First, it empowers cloud customers to choose security based on their context specific requirements, and change the type of requested security as the requirements change. Second, it allows cloud providers to charge cloud customers based on the type of security service they request. Such a differential pay-by-service-type model is at the heart of the cloud computing paradigm, and it is only natural that it is extended to include security services. Third, it allows designers, vendors and researchers to identify the different types of customer security requirements within a common representation, and also get their existing architectures and products deployed in real cloud environments – perhaps in small numbers at first until their value results in increased customer demand.

The primary contributions of this paper are:

- a mechanism for requesting different types of VM security based on different underlying threat models,
- a detailed mapping of security features expected by customers to a set of threat models we define, and then to secure server architectures,
- a mechanism for managing trust in the cloud servers based on the hardware and/or software security mechanisms they provide, and
- a prototype implementation of the above features within the opensource OpenStack cloud software.

The remainder of the paper is organized as follows: Section II describes the security threat model for infrastructure as a service (IaaS) clouds. Section III describes the components and working of the security on demand framework. Section IV provides details of our prototype implementation in the OpenStack cloud system. Section V discusses security, while Section VI discusses performance, especially live VM migration. We briefly list related works in Section VII and conclude in Section VIII.

## II. Threat model for IAAS cloud

In an IaaS-based cloud computing setup, the security of a customer's application (i.e. code) and data, can be compromised by non-trusted entities in numerous ways. Attacks can either be launched by malicious entities that operate within the same VM, within the same server, within the cloud, or outside the cloud provider's setup. The attacks can compromise different security properties like *confidentiality, integrity, availability, privacy, anonymity*, etc. Table I shows potential attacks at each level. We categorize these attacks to build a comprehensive threat model, that forms the basis of our framework.

TABLE I
Attacking entities within and outside the cloud

| | VM-level Attacks | Server-level Attacks | Cloud-level Attacks |
|---|---|---|---|
| **Entities** | OS | Other VMs | Cloud Manager |
| | Other Apps | Hypervisor | Other Servers |
| | Same App | Hardware | Outside Entities |

### A. VM-level Attacks

**Attacks by the OS inside the VM.** The operating system running within the VM can be malicious or compromised to breach the security of customers' code and data. Since the OS has access to the full memory space allocated to the VM, it can see or modify a VM's code or data thus violating confidentiality and integrity.

**Attacks by Other Programs in the VM.** It is possible that the security of data and code within a VM is compromised by other programs running in the same VM. Since the programs may share the same physical memory space, a malicious program may be able to compromise the security of the sensitive data and code.

**Attacks by the Same Application.** It is possible that the very code that is operating on the sensitive data is malicious and compromises the confidentiality of the data, by illegally leaking the data to some third party. For example, there have been cases where patients' medical records or phone books have been leaked out and sold by authorized applications [23], without the knowledge of the user.

### B. Server-level Attacks

**Attacks by Other VMs.** In cloud computing, cloud providers run multiple VMs simultaneously on the same server. The VMs thus share caches, machine memory and other resources with other VMs. A malicious VM can therefore compromise the security of other VMs running on the same server.

**Attacks by the Hypervisor.** Hypervisors are powerful: they are responsible for allocating hardware resources, they run at the highest privilege level, and have access to the VMs' memory. Hence, a compromised hypervisor can easily carry out confidentiality and integrity attacks on a VM. A hypervisor is also responsible for scheduling the VMs and hence can easily cause a Denial-of-Service (DoS) attack on a VM, breaching availability.

**Hardware Attacks.** Physical attackers have the ability to directly manipulate hardware, probe it or even steal the hardware to analyze it at a later time [24]. Many security breaches have been due to physical attacks on hardware involving stolen (or improperly disposed of) hard drives that contain sensitive data, e.g., [25]. Other attacks, such as the cold boot attack [26], show that secrets can be recovered from volatile memory even after it is removed from a computer.

### C. Cloud-level Attacks

**Attacks by the Cloud Manager.** A cloud manager is responsible for allocating VMs to servers. A malicious cloud manager can compromise the security of a customer's code and data. A cloud manager can compromise the security of a VM by

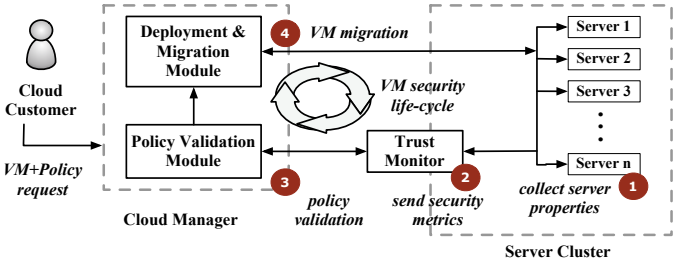


Fig. 1. Security on Demand Framework.

allocating it to the wrong set of servers. A cloud manager may also run at a higher privilege and access a server's memory, thereby breaching a VM's confidentiality and integrity.

**Attacks by Other Servers in the Cloud.** It is possible that a VM is attacked by other servers in the cloud. A malicious or compromised server inside the cloud perimeter may attempt to launch an attack on another server where the sensitive VM or applications are running; this would be an indirect attack. A malicious server could also attempt to attack the VM or applications directly if such a VM or applications have open connections to the network (e.g. running a web server).

**Attacks by Entities Outside the Cloud.** It is also possible that a VM is attacked by entities outside the cloud. The attacks possible in this scenario are similar to the attacks done by servers in the cloud, but these entities are outside the cloud perimeter and these attacks have the additional difficulty of having to breach any protections present at the perimeter of the cloud.

## III. Security on Demand Cloud Framework

### A. Overview

Our *Security on Demand* architecture, as shown in Figure 1, is based upon the IaaS cloud computing paradigm, in which VMs are leased to cloud customers. A customer initially requests a VM along with a security policy, which defines the desired level of security requested for the VM. The level of security requested depends on the cloud entities trusted by the customer.

For the initial deployment, the cloud manager determines the type of server that is best suited to match the security requirements of the VM. We assume that the cloud provider has different types of servers, each type capable of providing different types or degrees of security. Once the VM is deployed on an appropriate server, each deployed VM enters a security life-cycle, which ensures that the requested VM security is always maintained while the VM is running. The security life-cycle process operates in 4 steps: In Step ①, security properties of each server are collected by a trust monitor. In Step ②, the Trust Monitor translates servers' security properties it has collected into well-defined, meaningful security capabilities. In Step ③ the capabilities of each server are then provided to a Policy Validation module responsible for the validation of a virtual machine's requested security policies against these capabilities. Finally, in Step ④ a response mechanism is triggered if either the security requirements of the VM change, or the server on which the VM is deployed is no longer trustworthy. The response mechanism is typically a migration request which moves the VM to a suitable server capable of satisfying the security requirements of the VM.

In the remainder of this section, we will explain how different degrees of security are requested, how different types of secure server architectures are capable of enforcing different security needs, what properties need to be collected for maintaining trust, and finally the feasibility of live-migration as a response mechanism.

### B. Requesting VM Security

A cloud customer needs to be able to specify his or her security requirements for cloud services. A customer can choose between different security levels by specifying different entities that he trusts within the cloud. We propose an initial set of security options from which a customer can choose different VM security levels.

1) *Basic Security:* This is the basic security package in which the VM is protected against other VMs on the same server.
2) *Hypervisor-protection (or Host-OS protection for hypervisors like KVM):* This option will protect the VM against an untrusted hypervisor.
3) *HW-protection:* This option will also provide protection against HW attacks.
4) *Cloud Manager-protection:* This option will protect the VM against an untrusted cloud manager.
5) *Guest OS-protection:* In this option, the VM's code and data will be protected against an untrusted guest OS.
6) *Apps-protection:* In this option, the VM provided to the customer will allow the customer to specify trusted applications within the VM that will operate on customer-specified protected data, and the data will be protected against other applications within the VM.
7) *Same App-protection:* This option will provide post-access output control on sensitive data. The customer-specified sensitive data will be protected against the very application operating on the sensitive data, and prevent that application from illegitimately leaking out the data beyond this application.

A customer can combine different options specified from 1-7 to select the type of security for his leased VM. Based on the selections, our security on demand software fills out a security request matrix as shown in Table II. Once the options are selected, a customer can also specify whether he requires confidentiality protection or integrity or both. In future, other security properties like availability, provenance and audit logs can be added. For example, consider VM-A for which a basic security package (Option 1) is chosen with both confidentiality and integrity protection, and a VM-B for which options 1 and 2 are chosen for both confidentiality and integrity protection. For each of these VMs the security matrix is filled out correspondingly as shown in Table II. In addition, for some customers, default security packages can be created, e.g., "security for on-line medical records", and these are translated into the options above (e.g., options 1, 5, 6, 7).

TABLE II
SECURITY REQUEST MATRIX. THE CELLS WOULD BE FILLED ACCORDING TO THE SECURITY PACKAGE CHOSEN BY THE CUSTOMER.

| Protected Entity | Same VM | | | | | | Same Cloud Server | | | | | | Cloud Manager | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Same App | | Other Apps | | Same OS | | Other VMs | | Hypervisor | | HW attacks | | | |
| | C | I | C | I | C | I | C | I | C | I | C | I | C | I |
| VM-A | x | x | x | x | x | x | √ | √ | x | x | x | x | x | x |
| VM-B | x | x | x | x | x | x | √ | √ | √ | √ | x | x | x | x |

TABLE III
SERVER SECURITY CAPABILITIES: PROPOSED HARDWARE AND SOFTWARE SECURITY ARCHITECTURES AND SOME COMMERCIALLY AVAILABLE SECURITY ARCHITECTURES WHICH DEFEND AGAINST CONFIDENTIALITY AND INTEGRITY ATTACKS FROM DIFFERENT SOURCES. (*) REMOVES HYPERVISOR DURING RUNTIME. TSM = TRUSTED SOFTWARE MODULE WITHIN AN APP

| | Security Architecture | Same VM | | | | | | Same Cloud Server | | | | | | Cloud Mgr. | | Protected Entity |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Same App | | Other Apps | | Same OS | | Other VMs | | Hypervisor | | HW attacks | | | | |
| | | C | I | C | I | C | I | C | I | C | I | C | I | C | I | |
| Hardware | AEGIS [16] | No | No | Yes | Yes | No | No | n-a | n-a | n-a | n-a | n-a | n-a | n-a | n-a | Data+App |
| | XOM [15] | No | No | Yes | Yes | Yes | Yes | n-a | n-a | n-a | n-a | n-a | n-a | n-a | n-a | App |
| | SP [13], [14] | Yes | Yes | Yes | Yes | Yes | Yes | n-a | n-a | n-a | n-a | n-a | n-a | n-a | n-a | Data+TSM |
| | Bastion [12] | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No | Yes | Yes | No | No | Data+TSMs |
| | DataSafe [19] | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | No | No | No | No | No | Data |
| | H-SVM [11] | No | No | No | No | No | No | Yes | Yes | Yes | Yes | No | No | No | No | Full VM |
| | HyperWall [8] | No | No | No | No | No | No | Yes | Yes | Yes | Yes | No | No | No | No | Full VM |
| Software | NoHype [7], [27] | No | No | No | No | No | No | Yes | Yes | No* | No* | No | No | No | No | Full VM |
| | Overshadow [9] | No | No | Yes | Yes | No | No | n-a | n-a | n-a | n-a | No | No | No | No | Data+App |
| | HyperSafe [10] | No | No | No | No | No | No | Yes | Yes | No | No | No | No | No | No | Full VM |
| Comm. | TPM [20] | No | Yes | Yes | Yes | No | No | n-a | n-a | n-a | n-a | No | No | No | No | Data+App |
| | Cell Prssr. [21] | No | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | Yes | No | n-a | n-a | Data+App |
| | IBM 4758 [22] | No | Yes | Yes | Yes | No | No | n-a | n-a | n-a | n-a | Yes | Yes | n-a | n-a | Data+App |

### C. Collecting Server Capabilities

Once a customer's security request is clearly represented in terms of the Security Request Matrix shown in Table II, a cloud provider can appropriately allocate servers that can satisfy the requirements. The cloud provider keeps an updated table of Server Security Capabilities (Table III) which it can match with the customer's Security Request (Table II).

*1) Classifying Secure Server Solutions:* Cloud providers can incrementally add servers with different security features in our Security on Demand framework. Whenever a new server type is added to the cloud provider's infrastructure, its security capabilities and the threat model it defends against is added to the Server Security Capabilities table, as shown in Table III. Table III allows our security on demand software to find out which of the various security architectures are appropriate to use in a given scenario. The focus of the table is on presenting which entities (last column) are protected from various other entities in the cloud environment (middle columns). Because each architecture (first column) has been designed with a different threat model and assumptions, the table is a necessary simplification and does not capture all aspects of the architectures. Some architectures pre-date the "cloud" computing paradigm and hence do not provide virtual machine and cloud abstractions (as indicated by *n-a* in the corresponding table entries). The table's purpose is to illustrate how to map the architectures to the protections they offer.

Such a table needs to be filled in by architects for their respective architectures. Table III has been filled based on descriptions in publications describing the architectures. Each row shows whether (or not), the given architecture protects confidentiality (represented by 'C') and integrity (represented by 'I') from the column entity. The three categories of attacker entities (groups of columns in Table III) are: VM-level entities, server-level entities and cloud-level entities, as in Table II.

Hardware architectures have traditionally focused on adding new hardware to create isolated execution environments where trusted software modules (TSMs) could execute safely [12]–[18]. More recently, architectures have aimed to extend the protections to entire virtual machines [7], [8], [11], [27]. This is especially interesting given the contemporary interest in cloud computing where the basic unit of computation is the virtual machine. Also, software-hardware architectures like DataSafe [19], for protecting sensitive data used by unvetted applications, can provide Security Option 7 (Section III-B).

Software security architectures focus on using existing facilities and innovative software engineering techniques to protect the code and data [7], [9], [10], [27]. These can be implemented today on commodity hardware making them most likely to be the first to be deployed.

Commercial manufacturers also have deployed a number of systems [20]–[22] which can provide extra protections for software through dedicated hardware. These are either a co-processor such as TPM or the IBM 4758 crypto cards, or processors with built-in security features such as the Cell Broadband Engine's security vault architecture.

*2) Trust Monitor:* We introduce a Trust Monitor which collects, monitors and maintains the servers' current security capabilities. One of its roles is to maintain the equivalent of Table III for the provider's infrastructure.

A trust monitor needs to securely collect information about the properties of different servers and attest their capabilities in

enforcing protections. Security architectures such as TPM [20] or Bastion [12], [28] already support such type of attestations. Furthermore, architectures such as HyperWall [8], [29] provide trust evidence attestation that the hardware protections are indeed in force during runtime. The Trust Monitor also monitors dynamic changes in the security capabilities of the servers. If any of these attestations fail, the trust monitor informs the Policy Validation module accordingly to take appropriate action.

The Trust Monitor can be realized as a module of a cloud management software, for example OpenStack [30]. In OpenStack, all modules are self-contained and can run on the same or a different server. We propose that a specialized trust monitor overlooks and monitors a particular type of servers using custom attestation protocols. Such a distributed approach of collections of trust monitors will also address the issues of a single point of failure. All trust monitors then report to the policy validation module via a standard interface and protocol.

### D. Policy Validation and Secure VM-Server Allocation

We also introduce the Policy Validation module which ensures that virtual machines are running on servers which satisfy their security requirements. The Policy Validation module takes inputs from the trust monitors on the one hand (based on Table III), the VM security policy specification from the cloud customer on the other hand (Table II), and determines an appropriate server for the VM. The inputs from the trust monitor are the available servers and any trust evidence information about them. The inputs from the cloud customer have been mapped into a row of Table II that specify the protections the customer requested.

The Policy Validation module runs in two modes: deployment mode, and relocation mode. During the deployment mode, the Policy Validation module takes in a new request for a virtual machine from the cloud customer, and determines the servers that satisfy the security requirements of the virtual machine. Once the set of servers is determined, the Policy Validation module requests the Deployment module to deploy the virtual machine on the selected set of servers. Relocation mode is used when a decision is made to relocate a virtual machine from its present server or set of servers.

Relocation is triggered either by some fault in the server or a change in its properties, as a result of which it no longer can satisfy the security requirements of a virtual machine it is running. It can also be triggered by a change in the type of security requested by the customer for the virtual machine. In either case, the Policy Validation module remaps the virtual machine to other servers that satisfy the security requirements of the virtual machine. This process is carried out throughout the lifetime of the virtual machine.

### E. Enforcing Protections

In order to ensure lifetime protection, we propose to use virtual machine migration [31]. If the customer's requirements change, or attacks or other factors change the protections that can be provided by the server on which the virtual machine is currently running, the virtual machine needs to be relocated to a different server that meets the security requirements.

Various forms of migration, such as live migration [32], [33] are commonly supported by cloud management software such as OpenStack. Our security on demand framework adds software to determine when the migration is triggered. Our security-focused SLAs create new reasons for migration: *change in customer security requirements or change in security state of the servers*.

An additional trigger and strategy for migration that we propose is based on the "moving target defense". Virtual Machines are moved to other servers every now and then so that attackers cannot try to co-locate their malicious VMs on the same physical server, as in [34]

While live VM migration can counter against outside attacks, attacks from within a VM can be defeated using VM introspection (VMI) methods which typically make use of a trusted hypervisor [35]. Hence, a customer choosing such protections (e.g., options 5 and 6 in Section III-B) cannot also choose option 2.

## IV. Implementation on OpenStack Essex

We implemented the security on demand framework on the OpenStack Essex platform. We modified the existing Nova modules and datapaths, as shown in Figure 2, to add the features of the security on demand framework; We had to make four major changes to the existing Nova implementation:

1) enable security policy specification for the VM via Dashboard,
2) store the VM policy and server properties in Nova Database (nova.db),
3) collect server properties from Nova-Compute and send them to the Nova-Schedule (the VM scheduler), and
4) add a policy-based filter to Nova-Schedule

In order to specify the VM policy, we modified the Dashboard to include a security policy when a user requests a new VM. The customer is now allowed to choose from different security packages described in Section III-B. The security policy information is sent to the scheduler and the database along with other specification parameters in the VM request. The security request specification is then stored in the database *nova.db*. The security policy is stored as a key-value pair in the *instance_metadata* table of the database *nova.db*.

The Policy Validation Module (PVM) and the Deployment and Migration Module (DMM) were added to the scheduler to enable policy-specific VM deployment and a live VM migration response mechanism against security attacks. The PVM consists of a custom policy-based filter mechanism to allocate servers to the VMs. The filter carries out the task of mapping the security requirements of the VMs stored in the *instance_metadata* table to the security properties of the servers stored in the *compute_node* table of *nova.db*. The DMM module subsequently starts or migrates VMs to appropriate servers.

The Trust Monitor is split across the Nova-Compute nodes (TMc) and the Nova-Schedule module (TMs). In the current

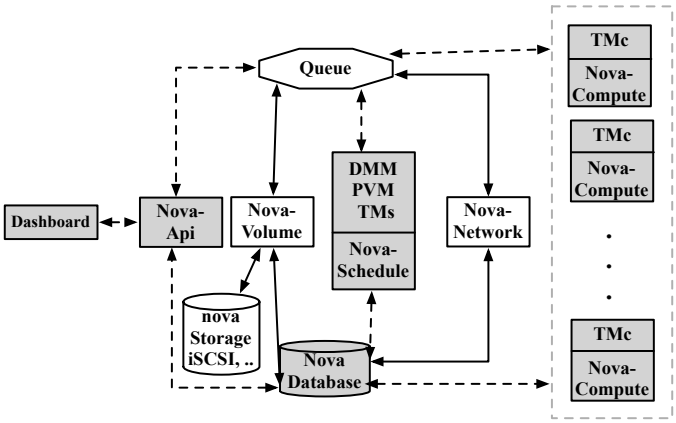


Fig. 2. Implementation of Security on Demand overlaid on Openstack Nova components. DMM is the Deployment and Migration Module, PVM is the Policy Validation Module, TMs is the Trust Monitor implementation within the Nova-Schedule Module, and TMc is the Trust Monitor implementation within each Nova-Compute Module. Grey colored boxes are modified modules and dashed arrows are modified communications

implementation, we simulate different compute nodes to act as some of the security servers described in Table III. To this effect we added a simulator, which acts as a part of the Trust Monitor on the compute nodes (TMc), to the underlying KVM hypervisor, to provide server properties to the Nova Database on bootup, and trust evidence while the server is running. Server properties are stored in the *compute_node* table of *nova.db* for each compute node. The server properties are passed on along with the other properties such as *running_vms, current_workload, cpu_info*, etc. At present the server security properties include *architecture_type* and *trust_evidence*. The *architecture_type* information is sent on the server bootup, while *trust_evidence* information is sent at regular intervals. In the present implementation we support architecture types: TPM, DataSafe and HyperWall, while trust evidence is sent only for the Hperwall architecture type in the form of the number of memory violations. The Trust Monitor implementation within the scheduler (TMs) collects these properties from the database and supplies them to the PVM.

## V. Security Discussion

The Security on Demand framework enables customers to select security options to customize the security requirements for their leased VMs. However, it is important that the operation of the framework is secure, and critical modules are protected. We assume the Cloud Manager is trusted (hence Option 4 in Section III-B is not currently supported). Beyond this, the framework needs to be secured for three additional aspects: 1) secure collection and servicing of the customer VM requests, 2) security of the process for establishing server security capabilities, and 3) secure execution of the response mechanism.

A customer must be able to request and change VM security in a secure manner, and it must be possible to collect, store and service these requests securely. Secure VM requests can be managed via authentication and authorization of cloud customers. The database where VM security requests are stored needs to be protected against unauthorized access and modifications. OpenStack currently provides both these features, along with access control to the Nova database via the OpenStack Identity Service (Keystone). Furthermore, the control module responsible for processing and enforcing VM security requests (i.e. the Nova-Schedule module), needs to be protected against illegal modifications compromising the integrity of its code. It is possible to secure the Nova-Schedule module code by running it on a server supported by a Trusted Platform Module (TPM), which can verify the code integrity during startup.

The process for establishing server security capabilities needs to ensure that the server properties are provided back to the scheduler via trustworthy attestation mechanisms and protocols. We envision these properties are collected at the startup from different servers by means of a security-focussed power-on self test (POST). Also, each security architecture provides its runtime security health, via trust evidence mechanisms.

The live VM migration process itself needs to be secured to guarantee data confidentiality and integrity to the user. In order to secure the VM migration process, the control mechanism handling the VM migration protocols, the communication channel between the servers, and the migration modules need to be protected against attacks. Finally, we assume that all the Nova-Compute modules along with our SoD modifications are securely started and provided with runtime integrity protections.

## VI. Performance Measurements

The key performance issue is to see whether VM migration is feasible as a real-time security response mechanism. Hence, we now study the time required to migrate a VM from one host to another. The act of carrying this out with negligible downtime is known as *Live VM Migration*. VMs use network-attached storage (NAS) devices for secondary storage, which does not have to be migrated with the VM. Live VM migration, thus, boils down to transferring VM memory state from one host to another. We will focus on the *pre-copy* algorithm because it is the one used by most commodity hypervisors to carry out live migration of VMs, including VMware, Xen, KVM, VirtualBox, Microsoft Hyper-V and OpenVZ.

The *pre-copy* algorithm proposed by [36] uses an iterative push phase, where dirty (i.e. modified) pages are transferred from the source to the destination machine iteratively, followed by a minimal stop-and-copy phase. In the stop and copy phase, the VM is stopped and the CPU state and any remaining modified pages are sent to the destination VM, leading to a fully consistent state.

To test the live VM-migration performance, we have selected a range of 7 different types of commonly used datacenter applications as listed in Table IV [37]. Our test bed is comprised of two hosts with identical hardware and software configurations. Each host comes with dual quad-core Intel Nehalem CPUs (1.6GHz), on top of which a KVM hypervisor is running. The Host OS is Ubuntu 12.04 LTS with kernel version 2.6.38.8, and the network connecting both hosts

TABLE IV
DATA CENTER WORKLOADS

| Workload | Benchmark |
|---|---|
| Mail Server | mstone [38] as remote SMTP client; smtp-sink [39] as SMTP server inside VM |
| App Server | Faban Benchmarking Framework [40] as remote client; Glassfish Server [41] with sample Java EE application inside VM |
| File Server | Dbench [42] inside VM |
| Web Server | Faban Benchmarking Framework [40] as remote client; Apache HTTP Server [43] inside VM |
| DB Server | Sysbench [44] inside VM |
| Stream Server | VideoLAN [45] inside VM; Wireshark [46] capturing stream packets remotely |
| Idle Server | No workload |

supports 1Gbps speeds. The VMs being migrated come with 1 GB of dedicated RAM and 1 CPU core. They run Ubuntu 12.04 LTS Server Edition, and their secondary storage disk images are roughly 5GB in size after installing all benchmarks.

For each workload, we ran 2-minute long benchmarks and performed migrations at five different migration points within that 2-minute interval (t=0, t=24s, t=48s, t=72s and t=96s). Such a multi-interval approach ensures fairness in the observed results and addresses the *problem of the migration point* described in [37]. We then averaged those results to get an overall measurement for metrics associated with each workload.

Table V gives performance measurements for the *total time*, *downtime* and *data sent* throughout the migration process for each type of application. The *total time* is the time elapsed between the moment when the *migrate* command is issued and the instant when the VM is resumed at the destination host after the stop-and-copy phase of the pre-copy algorithm. The *downtime* is a measure of liveliness and corresponds to the time taken by the stop-and-copy operation, which is when incoming packets are lost and no computational tasks are scheduled due to the fact that both the source and destination VMs are paused. The *data sent* column presents the amount of migration-related data that was transferred from one host to the other during the migration process.

Table V shows that live VM migration takes between 7 and 9 seconds to complete depending on the workload, which serves as evidence of how effective this reactive mechanism can be. With the exception of the Application Server, all workloads experience sub-second downtimes, so the amount of packets lost is negligible. Application Servers are exceptionally bad for live migration, since they have large page dirtying rates while leaving little spare bandwidth for migration purposes. These live VM migration tests were carried out in a non-OpenStack environment. Within an OpenStack implementation, migration triggering caused an additional average delay of 2.33 seconds, which increased the total live VM migration times by that amount, while the downtimes remained the same.

## VII. RELATED WORKS

Today's SLAs are focused on performance metrics [1]–[3]. They lack flexible security specification features and the security features which are included in them are static, rigid and limited to a small set of features [4].

TABLE V
MIGRATION PERFORMANCE WITH DATA CENTER WORKLOADS

| Benchmark | Total Time(s) | Downtime(ms) | Data Sent(MB) |
|---|---|---|---|
| Mail Server | 8.9 | 300 | 800.39 |
| App Server | 7.5 | 4050 | 728.21 |
| File Server | 7.2 | 250 | 697.87 |
| Web Server | 7.2 | 250 | 645.33 |
| DB Server | 7.4 | 450 | 717.52 |
| Stream Server | 8.0 | 650 | 716.21 |
| Idle Server | 2.6 | 200 | 212.34 |

Providers try to keep a high standard of protection by ensuring proper procedures are taken when handling customer code and data, access controls are in place, counter measures such as network monitoring are active, etc. [5], [6]. Such security guarantees in cloud computing offerings, however, are often applied to all customers equally. By employing Security on Demand as presented in this paper, different providers could differentiate themselves based on security offerings. The simple set of options (e.g. encryption or no encryption [6]) supported by some providers could be gainfully expanded.

We illustrate our Security on Demand framework (see Table III) using the different security architectures proposed in the academia, each focussing on specific security features and threat models [7]–[16], [19], [27]. Also, commercially, a few architectures are [20]–[22], or may soon be [17], [18] available.

We enhanced past work in virtual machine migration [31], especially live migration [32], to enable dynamic security provisioning in our SoD framework. As security breaches will likely affect many virtual machines at a time (e.g., all virtual machines on a server under attack), past work on gang virtual machine migration [33] will also be useful.

## VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented our vision for Security on Demand for cloud computing and presented a framework for realizing it. At the heart of our framework lies our comprehensive threat model for VMs in Cloud Computing, which forms the basis for understanding and analyzing cloud security from both the customer's perspective and the perspective of server security architectures. The framework builds on a detailed mapping of various hardware and software security architecture's properties to cloud security features expected by customers. Not only are the mappings central in realizing security-focused SLAs, but they will also help security architects explore new combinations of security features (possibly not available today) that customers desire – thus driving the design of new security architectures.

We have a prototype implementation of part of our framework on OpenStack, an open source cloud management software. We will extend OpenStack with a more sophisticated Trust Monitor Module and Policy Validation Module. We also plan to extend existing SLA frameworks to support customers' security requests.

## ACKNOWLEDGEMENT

This work was supported in part by the National Science Foundation under grant NSF CNS-1218817.

## REFERENCES

[1] *Web Service Level Agreement (WSLA) Language Specification*, IBM, Jan. 2003. [Online]. Available: http://www.research.ibm.com/wsla/WSLASpecV1-20030128.pdf

[2] *Web Services Agreement Specification*, Open Grid Forum, Mar. 2007. [Online]. Available: http://www.ogf.org/documents/GFD.107.pdf

[3] P. Patel, A. Ranabahu, and A. Sheth, "Service Level Agreement in Cloud Computing," in *Proceedings of the Workshop on Best Practices in Cloud Computing: Implementation and Operational Implications for the Cloud*, October 2009.

[4] "Review and summary of cloud service level agreements," IBM developerWorks, August 2010. [Online]. Available: https://www.ibm.com/developerworks/cloud/library/cl-rev2sla.html?ca=drs-

[5] "Security Whitepaper: Google Apps Messaging and Collaboration Products," Google security whitepaper, WP64-1109, 2011.

[6] C. Kaufman and R. Venkatapathy, "Windows Azure Security Overview," August 2010. [Online]. Available: http://go.microsoft.com/?linkid=9740388

[7] J. Szefer, E. Keller, R. B. Lee, and J. Rexford, "Eliminating the hypervisor attack surface for a more secure cloud," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS, October 2011, pp. 401–412.

[8] J. Szefer and R. B. Lee, "Architectural Support for Hypervisor-Secure Virtualization," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, March 2012, pp. 437–450.

[9] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow: A Virtualization-Based Approach to Retrofitting Protection in Commodity Operating Systems," in *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS, May 2008.

[10] Z. Wang and X. Jiang, "Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2010, pp. 380–395.

[11] S. Jin, J. Ahn, S. Cha, and J. Huh, "Architectural Support for Secure Virtualization under a Vulnerable Hypervisor," in *The Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture*, December 2011.

[12] D. Champagne and R. Lee, "Scalable architectural support for trusted software," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*, Jan. 2010, pp. 1 –12.

[13] R. B. Lee, P. Kwan, J. P. McGregor, J. Dwoskin, and Z. Wang, "Architecture for protecting critical secrets in microprocessors," in *Proceedings of the International Symposium on Computer Architecture*, ser. ISCA, June 2005, pp. 2–13.

[14] J. S. Dwoskin and R. B. Lee, "Hardware-rooted trust for secure key management and transient trust," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS, October 2007, pp. 389–400.

[15] D. Lie, C. Thekkath, M. Mitchell, P. Lincoln, D. Boneh, J. Mitchell, and M. Horowitz, "Architectural support for copy and tamper resistant software," *SIGPLAN Not.*, vol. 35, pp. 168–177, November 2000.

[16] G. E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Aegis: architecture for tamper-evident and tamper-resistant processing," in *Proceedings of the Annual International Conference on Supercomputing*, ser. ICS, June 2003, pp. 160–171.

[17] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP '13, June 2013.

[18] F. McKeen, I. Alexandrovich, A. Berenzon, C. V. Rozas, H. Shafi, V. Shanbhogue, and U. R. Savagaonkar, "Innovative instructions and software model for isolated execution," in *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP'13, 2013.

[19] Y.-Y. Chen, P. A. Jamkhedkar, and R. B. Lee, "A software-hardware architecture for self-protecting data," in *Proceedings of the 2012 ACM conference on Computer and communications security*, ser. CCS '12, October 2012, pp. 14–27.

[20] "Trusted Computing Group. TCG TPM Specification." [Online]. Available: http://www.trustedcomputinggroup.org/

[21] K. Shimizu, H. P. Hofstee, and J. S. Liberty, "Cell broadband engine processor vault security architecture," *IBM Journal of Research and Development*, vol. 51, no. 5, pp. 521 –528, September 2007.

[22] J. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, and S. Smith, "Building the IBM 4758 secure coprocessor," *Computer*, vol. 34, no. 10, pp. 57–66, October 2001.

[23] S. Chen, R. Wang, X. Wang, and K. Zhang, "Side-channel leaks in web applications: A reality today, a challenge tomorrow," in *Proceedings of the IEEE Symposium on Security and Privacy*, Washington, DC, USA, May 2010, pp. 191–206.

[24] A. Tereshkin, "Evil maid goes after pgp whole disk encryption," in *Proceedings of the 3rd international conference on Security of information and networks*, ser. SIN '10, September 2010.

[25] "Stolen BlueCross Hard Drives Lead to Credit Watch," memphis Daily News, Vol. 124, No. 233, November 27, 2009. [Online]. Available: http://www.memphisdailynews.com/editorial/Article.aspx?id=46337

[26] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten, "Lest we remember: cold-boot attacks on encryption keys," *Communications of the ACM*, vol. 52, pp. 91–98, May 2009.

[27] E. Keller, J. Szefer, J. Rexford, and R. B. Lee, "Nohype: virtualized cloud infrastructure without the virtualization," in *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ser. ISCA, June 2010, pp. 350–361.

[28] D. Champagne, "Scalable security architecture for trusted software," PhD Thesis, Princeton University, Princeton, NJ, 2010.

[29] J. Szefer, "Architectures for secure cloud computing servers," PhD Thesis, Princeton University, Princeton, NJ, 2013.

[30] "OpenStack Open Source Cloud Computing Software." [Online]. Available: http://www.openstack.org/

[31] M. Nelson, B.-H. Lim, and G. Hutchins, "Fast transparent migration for virtual machines," in *Proceedings of the USENIX Annual Technical Conference*, ser. ATEC '05, April 2005, pp. 391–394.

[32] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05, May 2005, pp. 273–286.

[33] U. Deshpande, X. Wang, and K. Gopalan, "Live gang migration of virtual machines," in *Proceedings of the 20th international symposium on High performance distributed computing*, ser. HPDC '11, 2011, pp. 135–146.

[34] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09, November 2009, pp. 199–212.

[35] T. Garfinkel, M. Rosenblum *et al.*, "A virtual machine introspection based architecture for intrusion detection." in *NDSS*, 2003.

[36] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, and A. Warfield, "Live migration of virtual machines," in *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation - Volume 2*, ser. NSDI'05, 2005, pp. 273–286.

[37] D. Huang, D. Ye, Q. He, J. Chen, and K. Ye, "Virt-lm: a benchmark for live migration of virtual machine," in *Proceedings of the second joint WOSP/SIPEW international conference on Performance engineering*, ser. ICPE '11, 2011, pp. 307–316.

[38] "mstone multi-protocol testing system," http://sourceforge.net/projects/mstone/.

[39] "smtp-sink(1) - Linux man page," http://linux.die.net/man/1/smtp-sink.

[40] "Faban Harness and Benchmark Framework," http://java.net/projects/faban/.

[41] "GlassFish - Open Source Application Server," http://glassfish.java.net/.

[42] "Dbench filesystem benchmark," http://dbench.samba.org/.

[43] "The Apache HTTP Server Project," http://httpd.apache.org/.

[44] "SysBench: a system performance benchmark," http://sysbench.sourceforge.net/.

[45] "VLC media player," http://www.videolan.org.

[46] "Wireshark: the world's foremost network protocol analyzer," http://www.wireshark.org/.