P-Cop: Securing PaaS Against Cloud Administration Threats

Bruno Braga
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal
Email: brunobraga@ist.utl.pt

***Abstract*—Platform-as-a-Service (PaaS) infrastructures are highly dependent on the operational administrators. In fact, ill-configured software systems or compromising insider activity can result in serious data breaches for clients of PaaS services. A general security approach against untrusted administrators is to employ operating system hardening techniques to limit their access privileges on the cloud nodes where guest computations take place. However, such an approach is overly inflexible for PaaS services, since operational administrators commonly require superuser privileges in order to apply a security patch, change a firewall rule, etc. This paper presents P-Cop, a system aimed at enforcing secure PaaS maintenance while preserving administration flexibility. To that end, P-Cop implements a *bipartite maintenance model* in which the privileges of operational administrators on a given cloud node can be elevated to superuser, but no guest computations can be allocated to such a node until the maintenance commands have been endorsed by an auditor, i.e., a third-party trusted mutually by cloud provider and clients. P-Cop is materialized as a cloud administration proxy which supervises all privileged commands issued by operational administrators and that auditors must endorse. Our current P-Cop design targets Docker-containerized PaaS services and leverages trusted computing hardware to enable remote attestation of the software integrity by external clients.**

I. Introduction

The danger of cloud mismanagement is an often overlooked security risk that Platform-as-a-Service (PaaS) customers are faced with. In general, operational administrators are responsible for installing, configuring, and troubleshooting software systems of PaaS infrastructures. To perform such tasks, operational administrators hold privileged access to the cloud nodes of the infrastructure, including to those where client computations take place. As a result, they have access to customer data. Although in the vast majority of cases operational administrators use their privileges judiciously, such privileges can be misused by introducing security flaws that can potentially result in serious data breaches for customers.

To overcome such risks, a common practice is to segregate administration roles so as to reduce the number of individuals that know the root passwords of critical systems. Unfortunately, such individuals can still introduce critical security flaws or obtain unrestricted access to customer data. To prevent unauthorized access to customer data, some systems preclude operational administrators from obtaining root privileges entirely. Systems such as CloudVisor [1] consist of hardened hypervisor, which expose carefully crafted administration interfaces that allow for performing a large range of management tasks without compromising the security of customers' computations. Such systems are normally coupled with Trusted Platform Module (TPM) [2] hardware and cloud attestation services [3], [4] to allow for remote attestation of systems by the customers. However, enforcing a permanent reduction of administrator privileges is hardly tolerable in PaaS services because some critical maintenance tasks require root privileges, e.g., apply security patches, modify firewall rules, install software packages, etc.

This paper presents P-Cop, a system which aims to secure PaaS services against cloud mismanagement threats without precluding operational administrators from acquiring superuser privileges whenever necessary. The key idea to achieving this property is to enforce a bipartite maintenance model between operational administrators and an *auditor*, i.e., a third-party whose role is to validate software configurations and which is mutually trusted by both cloud provider and customers. Under this model, operational administrators have limited privileges in the common case. In the cases where superuser privileges are necessary, operational administrators can request privilege elevation so that they can log into cloud nodes as root and execute arbitrary commands. However, such commands must first be endorsed by the auditor before they can be definitely accepted. To prevent data breaches, from the moment an operational administrator enters a node as root until an auditor endorses all performed operations, no guest computations can be allocated to the node.

To implement a bipartite maintenance model, P-Cop relies on a cloud administration proxy to be deployed on the PaaS backend. This proxy mediates all superuser operations performed by the operational administrators and supports their respective endorsement by the auditor. To acquire privilege elevation on a given cloud node, an operational administrator requests a *super-session* to the proxy. After ensuring that no customer data is left on the node, P-Cop lets the administrator log into the target cloud node by establishing a tunneled SSL connection. Over that connection the operational administrator can execute arbitrary root com-

mands (i.e., UID = 0) while the proxy keeps on a local log a record of the command history of the super-session. Auditors can later access the log on the proxy in order to endorse the super-session commands if no security vulnerabilities have been introduced, or revert them otherwise.

P-Cop design addresses several challenges. For example, since the P-Cop proxy is installed on the cloud backend it must itself be shielded against potential attacks by untrusted administrators. P-Cop must keep track of the software configuration of each cloud node in a way that is capable of surviving cloud node reboots. Moreover, it is necessary to provide evidence to PaaS clients that the software of P-Cop proxy and cloud nodes has been endorsed by a trusted auditor, otherwise no security assurances can be given to PaaS clients. To address such challenges, P-Cop incorporates new security protocols, which leverage TPM chips deployed on the cloud nodes to be the root of trust.

Without loss of generality, we built P-Cop for Docker-containerized PaaS services. We implemented both the P-Cop system and a PaaS service prototype, and performed empirical evaluation of the system based on benchmarks. Results show that our system adds small performance overheads to the PaaS service. Due to P-Cop, a time lag exists since super-session operations are issued by operational administrators and their changes are reflected onto the PaaS clusters. The extent of that lag depends mostly on how tightly coupled operational administrators and auditors operate.

II. Background and Goals

A. Background on "Containerized" Platform-as-a-Service

Containerized Platform-as-a-Service (PaaS) provides hosting cloud services for customer applications within *containers*. Containers are isolated environments sharing a common underlying operating system (OS) interface but giving customers substantial flexibility for installing software according to the needs of their applications. Containers are supported on a given server by software frameworks such as Docker [5], which enforces isolation between containers and manages their lifecycle.

In general, a containerized PaaS infrastructure comprises clusters of three types of servers (or nodes). The *minion* nodes (also named minions) is the place where guest containers execute. These servers are typically configured with an operating system running Docker, for example. The *repository* nodes store container base images. Base images include pre-configured software (e.g., apache) which form the basis for customized container images to be instantiated on the minions. Customized images can include software packages freely chosen by the customers. The *monitor* nodes implement the logic that ties together the entire service: they process container deployment requests, allocate containers to minions, manage resources, etc.

For the sake of this paper, we highlight four main stakeholders: (1) the *cloud provider* owns the cloud infrastructure, (2) *developers* are the customers of the service and are responsible for deploying containerized applications onto the cloud infrastructure, (3) *end-users* access developers' applications instantiated inside guest containers as regular Web applications, (4) *operational administrators* are responsible for maintaining the software systems of the PaaS infrastructure.

B. Goals, Threat Model, and Assumptions

The goal of our work is to enable the design of PaaS services that can prevent access to guest containers by untrusted operational administrators. In spite of such restrictions, our solution must allow for flexible administration of the cloud by allowing operational administrators to obtain superuser privileges whenever necessary.

We assume that one or more operational administrators are untrusted but not all. An operational administrator can remotely reboot or power-cycle any of the cloud nodes to run an arbitrary operating system. He can then either mount the local file system and access persistent hard disk state, or install an arbitrary software stack on the node. If the node boots to the software stack installed on the local disk, the operational administrator is limited to accessing the node through the administration interface provided by that specific software stack, e.g., an SSH console or a Web interface. The privileges that the administrator can acquire through that interface depend on how the software stack is configured. An untrusted administrator can install network sniffers and similar software in order to eavesdrop the network traffic, modify, suppress, or inject packets. However, we exclude attacks that involve physical access to the hardware, attacks based on software exploits, and side-channel attacks.

Every server installed in the cluster is equipped with a Trusted Platform Module (TPM) chip. Each TPM has a unique keypair Attestation Identity Key (AIK) whose private key is bound to each chip. We require the TPM chip and the cryptographic algorithms used in our solution to be correct.

III. Architecture

We present P-Cop (PaaS Cop), a PaaS cloud management system that provides container isolation from operational administrators while providing flexible maintenance capability. In our solution, we focus on software administrators (hereby called operational administrators or just administrators), leaving the hardware ones for future work. In our solution, rather than permanently restricting administration privileges of minion nodes, we allow such privileges to be temporarily elevated in order to let administrators log into a particular minion node and perform operations that require root access. Then, through a process of endorsement, such privileged operations must be validated by an *auditor* in order to ensure that no vulnerabilities have been introduced into the

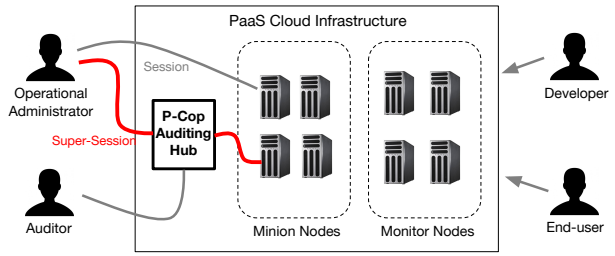

Figure 1. P-Cop architecture.

node software and therefore the node can be trusted to securely accommodate PaaS containers. Simply put, P-Cop enforces a bipartite maintenance policy between operational administrators and auditors.

Figure 1 represents the architecture of the P-Cop system when deployed in a PaaS cloud backend. The central component of P-Cop is the *auditing hub*, which is a cloud administration proxy provided by independent servers. The auditing hub is responsible for managing super-sessions. Super-sessions consist of tunneled SSL connections over which an operational administrator can access a minion node with root privileges. In regular sessions, the operational administrator does not have root privileges. Another component of P-Cop is the *node guard*, which is a software agent that runs on each minion node and implements local actions requested by the auditing hub, namely attestation requests and local state transitions. Finally, P-Cop provides *client* software that allow stakeholders to interact with the system.

A. Restricted Operation Mode

Essentially, P-Cop ensures that PaaS guest containers can be allocated and execute only on minion nodes running a software stack approved by both operational administrators and auditors. In particular, when such a software stack is installed on a minion node, operational administrators cannot acquire superuser privileges (e.g., log into the root account). Instead, they are restricted to using a limited management interface that allows them to maintain the node without compromising the confidentiality or integrity of guest containers, e.g., collect logs, set up resource management policies, monitor the resource consumption, etc. This management interface can be provided by a remote user session over secure channel protocols like SSH, HTTPS, or similar.

To supervise which minion nodes of the PaaS infrastructure can accommodate guest containers, P-Cop maintains a list of *verified nodes*. Such nodes are assured to be installed with trusted software stacks. This is achieved by requiring the software to be approved by the one or more auditors and operational administrators, and then leveraging TPMs to attest that such software is running on the cloud nodes. P-Cop provides the means for developers to instantiate their application on guest containers hosted by verified nodes and for users to process their data on such containers safely. P-Cop ensures that only the minion nodes that belong to this list are eligible for hosting guest containers.

B. Privileged Operation Mode

In case an operational administrator needs elevated privileges on a given verified node, P-Cop allows for opening a *super-session*. A super-session removes the restrictions imposed by security configurations on minion nodes, allowing the operational administrator to log into the system with root privileges. However, to prevent security breaches, before establishing the session, P-Cop: 1) stops all running guest containers and wipes their content so as to avoid leaving forensic traces of user data and application code that could be accessed by the operational administrator, and 2) removes the node from the verified list to avoid future instantiation of guest containers on minion nodes that are currently controlled by the operational administrator and, therefore, are potentially insecure.

When the operational administrator obtains access to a minion node through a super-session, he can perform arbitrary commands under root, e.g., install software, resize disk partitions, set up firewall rules, apply kernel patches, etc. Given that these operations can potentially leave the system in an insecure state (e.g., if a backdoor is left), the node cannot immediately be added back to the verified node list once the super-session ends. Instead, P-Cop adds the node to an *unverified node list* along with a log that provides a detailed account of all operations that were performed by the operational administrator during the super-session. In particular, the log contains a record of all input commands provided by the operational administrator and respective returned output. In order to return to the verified node list, such operations must be properly endorsed.

C. Endorsement of Super-Sessions

Endorsement of super-session logs aims to ensure that the input commands issued by the operational administrator have not introduced security breaches. This operation requires manual inspection of the logs by an *auditor*. The auditor is a trusted third-party that has special privileges on P-Cop for obtaining a list of super-session logs and approving or rejecting the operations performed within the super-session. If the super-session is approved, then the updated configuration of the minion node is deemed trusted. In this case, P-Cop removes the node from the unverified list and replaces it in the verified node list. Otherwise, if the super-session is rejected, a policy-based decision must be taken as to what to do next, which can be to revert the operations performed by the administrator, reinstall the software on the node, or other similar action. Such policy is defined by the auditor.

Essentially, one or more auditors and operational administrators are responsible for certifying the software of the PaaS infrastructure. In addition to the validation of super-session logs, the auditors must validate the software of the base container images and the software of the P-Cop system. Thus, in the interest of separation of privileges, the auditor role cannot be played by the operational administrators. Instead, it must be assigned to a third party that is mutually trusted by the cloud provider, developers, and users. The auditor may represent a collective entity that involve the participation of several stakeholders, e.g., cloud provider and customers.

IV. Design

This section describes the design of P-Cop, with particular emphasis on the security protocols implemented between auditing hub, monitors, node guards and P-Cop clients. To denote cryptographic protocols, we use the following notation. For asymmetric cryptography, K and K^P denote private and public keys, respectively. Notation $\langle x \rangle_K$ indicates data x encrypted with key K, and $\{y\}_K$ indicates data y signed with key K. We represent nonces as n. The TPM attestation keys are denoted by AIK. The TPM primitive *quote* $q(n)$ yields $\{PCR \mid\mid n\}_{AIK}$, where $\mid\mid$ denotes string concatenation.

A. System Initialization

To initialize the system, operational administrators must first install the software of the auditing hubs, monitors and minion nodes. During such procedure, a malicious or negligent administrator may undermine the security provided by the base software.

To ensure the correct setup of the full system, one or more auditors and operational administrators are responsible for attesting both monitors and auditing hubs. The monitors are, in turn, responsible for attesting the minion nodes. All the software supporting the base cluster must be distributed as software images that are approved by one or more auditors and operational administrators. The installation process is streamlined so that all software components are automatically installed and configured on the hosts.

Once the auditing hubs and monitors boot for the first time, they requires an activation step that must be triggered by one or more auditors and operational administrators. This activation step includes a remote attestation process that collects the fingerprint of the nodes (one or more auditors and operational administrators request $q(n)$ from nodes). If the fingerprint does not match the expected, then the installation was corrupted and therefore the server cannot be trusted. Otherwise, the activation proceeds with the generation of signed fingerprints ($\{MONITOR\_PCR\}_{K_A}$ and $\{AHUB\_PCR\}_{K_A}$, where K_A is the auditor's private key and, MONITOR_PCR and AHUB_PCR are approved fingerprints) that are sent to all attested nodes. These signed fingerprints allow the actors interacting with the auditing hubs and monitors a means to compare the nodes' fingerprints with the expected/trusted ones. The monitors also receive signed fingerprints for minion nodes since (as explained on the next subsection) monitors are responsible for attesting minion nodes upon boot to assess their eligibility to run application containers.

B. Bootstrapping and Attestation of Minion Nodes

Minion nodes are responsible for hosting guest containers of the PaaS service. After the monitor has been properly initialized, one of its main roles is to check the software configuration of the minion nodes and verify which of them have been properly set up with approved software. Only such nodes can be part of the verified minion node list, which identifies the nodes that are authorized to accommodate guest containers.

Similarly to the monitors, the software of minion nodes must be installed by an operational administrator, which can potentially introduce vulnerabilities in their configuration and, consequently, introduce security breaches. To mitigate such threats, P-Cop requires that the auditor certifies the images of trusted software stacks. Then, every time the minion nodes bootstrap, the monitor nodes execute remote attestation protocols to verify that the installed images are authentic and their integrity is intact. This is done by comparing the provided platform fingerprint with one or more (previously obtained) from the auditors and operational administrators upon system initialization. Only the cloud nodes that pass this test can be authorized to host PaaS containers.

To attest a minion node, the monitor node engage with the target minion node in simple remote attestation protocol. The auditing hub obtains a quote from the minion node's TPM and checks the AIK signature of the quote and the quote's PCR values. If the signature fails or the PCRs differ from the hash value of a certified runtime software, then the attestation fails. Otherwise, the minion node is considered to be trusted.

C. Secure PaaS Operations

A typical container-based PaaS service implements a set of core operations aimed to support the lifecycle of guest applications. Developers start by deploying their applications onto the service. The deployment process is coordinated by the monitor, which authenticates the developer, receives an application package and a *docker file*, and stores them. The docker file contains the configuration parameters of the container (e.g., the software packages to be installed in the container). Later, to instantiate the application on a guest container, the monitor selects a destination minion node from the minion node pool, and uploads to it the application package and the docker file. The minion node must then create a new guest container for the application. To this

end, the minion node must first build a container image. The base image may either be on the minion node or it may be obtained from a repository of trusted images (approved by one or more auditors and operational administrators). The base image is then extended according to the developer-defined configuration parameters specified in the docker file. The resulting container image can then be instantiated and the application deployed into it. At this point, the application becomes accessible to the users on a URL address assigned by the monitor until the developer removes the application from the service. While the application is in production, for scalability and fault tolerance purposes, the monitor can create multiple instances of the application in several guest containers possibly located in different nodes.

P-Cop aims must support the core PaaS operations while ensuring that the application instances available in production are properly secured. To this end P-Cop provides two guarantees. First, developers must be assured that their applications can be instantiated only on trusted minion nodes, i.e., minion nodes properly configured with certified software stacks (G1). Second, end-users must be ensured that the applications they are accessing are authentic, can be properly identified, and execute on a trusted minion node (G2).

To provide guarantee G1, P-Cop involves the monitor, auditing hub and the node guard software running on trusted minion nodes. As we mentioned in the section above, trusted minion nodes have been attested by the monitor. As a result of that attestation process, they all have approved configurations. In order to provide a means for end-users to verify that the applications are legitimate and running on legitimate nodes, developers must upload application certificates (i.e. SSL certificates) signed by well-known authorities (e.g. VeriSign). Since the certificates are signed by trusted authorities and the containers only run on minion nodes attested by monitors, which in turn are attested by one or more auditors and operational administrators, end-users can be sure that the applications they are accessing are to be trusted.

D. Establishment and Endorsement of Super-Sessions

In order for operational administrators to gain full access to the minion nodes as root they must open a super-session. To provide super-session support while preserving the invariant that applications are confined to trusted minion nodes, P-Cop must provide several guarantees. First, a super-session must only be granted to an operational administrator once it has been assured that the targeted minion node is properly sanitized. In particular, the minion node can not leave remnants of developer or user data from guest containers or sensitive P-Cop cryptographic material, namely the SSL and SSH keys (G1). Second, P-Cop must ensure that applications cannot be instantiated on a minion node of elevated administration privileges until the resulting configuration has not been properly validated and endorsed by an auditor (G2). Third, P-Cop must keep a complete record of super-session operations issued by the operational administrators. Such a record trail is fundamental to ensure that the auditor can trace all configuration changes performed to the minion node and thus spot any potential security vulnerabilities (G3). We explain how P-Cop provides these guarantees as we follow the super-session workflow.

The auditing hub acts like a gateway for super-session connections. Since the minion nodes are hardened, any attempts to log into a trusted minion node with root privileges will fail. Instead, the operational administrator's client must issue an "open super-session" request to the auditing hub in order to open a super-session. By intercepting all super-session requests, the auditing hub can then ensure that the minion node is properly sanitized before root access can be granted (G1). To this end, the auditing node sends an authenticated cleanup message to the minion node. The message receiver—the node guard—stops any existing guest containers, cleans up the remnants of developer application code and user data from the terminated guest containers, and erases the P-Cop keys maintained by the node guard. The cleanup operations are implementation-dependent and must be scrutinized by the auditor. Specialized wiping tools may be employed for zeroing memory pages and disk blocks, and encrypted disk partitions be used to prevent recovery of raw data from disk in case of reboots.

In order to provide replacement instances and avoid downtime, the auditing hub contacts one of the monitors to schedule the execution of replacement containers on verified nodes. This is possible because the monitors are responsible for keeping track of verified and unverified minion nodes. By keeping this information, it is possible to preserve the invariant G2. If a super-session request is received, the auditing hub removes the node from the list and places it in quarantine until the super-session is endorsed by the auditor.

The super-session request can be finally accepted. To establish the super-session, the auditing hub accepts an incoming SSH connection from the operational administrator's client and makes an SSH connection to the target minion node. The auditing hub then works like an SSH proxy which tunnels all messages exchanged between the client and the minion node during the super-session. Since the trusted minion nodes must only authorize incoming SSH connections to the root account from the auditing hub, this mechanism ensures that all traffic of the super-session is intercepted and logged by the auditing hub (G3). To keep a record of all operations performed by the operational administrator, the auditing hub makes a log of the super-session traffic. By inspecting and analyzing the content of the super-session logs, an auditor can then reconstruct the entire configuration history of the minion node and endorse or reject the changes.

E. Credentials Management and Distribution

In order to prevent impersonation or eavesdropping on communications between entities (e.g. nodes, actors), P-Cop employs SSL with mutual authentication on all software components. This allows P-Cop entities to communicate, while ensuring confidentiality, integrity and authentication. Aside from messages, P-Cop entities may exchange more complex data (e.g. monitors transfer applications to minions). To this end, in order to provide similar guarantees to SSL, SSH is employed.

Developers, operational administrators and auditors, as well as monitors need a way to assess the security of the platforms they interact with (i.e. know what software is running and if the software is approved). P-Cop relies on trusted computing to allow remote attestation on monitors, minions and auditing hubs. More precisely, developers attest monitors, operational administrators attest auditing hubs, monitors attest minions and, one or more auditors and operational administrators attest both auditing hubs and minions. The actors performing attestation assess the security of the platforms based on well-known configuration signed by auditors. All attestations are performed using SSH as the underlying communication protocol.

SSL certificates used by P-Cop entities are issued by the cloud provider. Also, in order to give stronger guarantees of trust on the provider, its root certificate should be signed by a well-known certification authority (e.g. VeriSign). The cloud provider should make available an authenticated signing service used by cloud nodes (e.g. monitors, minions, auditing hubs) and actors (e.g. developers, operational administrators, auditors) to sign their certificates which would be later used on SSL authentication. That service should use the cloud root certificate and due credentials to sign and issue the certificates. The signing service could be located/managed on/by an external and trusted company.

In order to protect the communications between cloud entities and the signing service, the service should employ SSL using a certificate signed by the provider's root certificate. The service must enforce some verification mechanism on the cloud entities to prevent malicious ones from getting trusted certificates. Cloud actors must have a combination of username and password while nodes must be attested by the service.

In order to protect credentials' integrity and secrecy, they should be protected using secure mechanisms for credentials storage (e.g. Java keystores), i.e., nodes and actors should keep their credentials in a secure storage, locally. Since such credentials' stores have protection mechanisms such as passwords, malicious users (e.g. operational administrators, hackers) may not access them if they don't know the passwords. Such aspect is specially important for minions credentials since operational administrators may access these keystores which are used by node guards. If the passwords protecting such keystores are kept on global variables, they may be erased before an operational administrator enter the node.

P-Cop employs SSH on remote sessions and file transfers. In terms of remote sessions, there may be administrative sessions (i.e. operational administrators connecting to auditing hubs and the latter connecting to minions) or auditing sessions (i.e. auditors access auditing hubs to read logs and commit/rollback administrative sessions). Auditors use their SSH sessions to connect to the auditing hubs and run P-Cop's auditing software that allows them to add SSH users (either operational administrators or other auditors) and access session logs.

Monitors and minions may have git repositories to host developers' applications. As in other cloud providers (e.g. Amazon), developers may upload their SSH keys through a web interface which will later allow them to access their computing instances. A similar approach is used on P-Cop where the keys will be used by the git repositories to receive files over SSH. Transfers between the monitors and the minions, their base images should have all the keys set up.

V. Implementation

We implemented the P-Cop system fully and a prototype of the PaaS software infrastructure. The P-Cop software components were developed using Java 8 with Java extensions for SSL. SSL credentials were generated using OpenSSL 1.0.1f and stored using Java keystores. Our trusted software setup is based on Docker 1.9 running in a hardened Linux CentOS 7.1.1503 with kernel 3.10.0-229.4.2.el7.x86_64. Interactions with the TPM to generate quotes are performed using Trousers 0.3.7. SSL connections are maintained using OpenSSH 6.6.1 servers. All generated credentials use RSA-2048 bits keys. The PaaS software consists of a simple monitor which is responsible for handling requests from the developer, administrator, and auditor, and for issuing requests to the auditing hub and to the minion nodes. The PaaS repository is provided by the monitor and the minion nodes which receive and send application packages using Linux scp.

VI. Evaluation

This section presents our evaluation of P-Cop in terms of performance and security.

A. Performance Evaluation

To evaluate the performance of our solution, we measured the time latencies of the operations that can most negatively affect the overall performance of the PaaS service: 1) attesting the infrastructure, which introduces delays in the system initialization that may affect the PaaS service uptime, 2) deploying applications, which can increase the time since the developers deploy their applications on the PaaS service and the application becomes available to users, and 3) opening

Case	File transfers	Container setup	P-Cop operations	Total
Hello	00:04	07:15	00:10	07:40
OSN	00:17	08:05	00:18	08:21

Table I
DISCRIMINATION OF DEPLOYMENT LATENCIES FOR FIVE INSTANCES OF HELLO AND OSN (MINUTES:SECONDS)

super-sessions, which introduce a delay until configuration changes are reflected into production.

Testbed. We evaluated P-Cop and our PaaS service prototype on a cluster that consists of: ten minion nodes, one monitor, and one auditing hub. All nodes have two Intel Xeon 3.00GHz, 2GB of RAM and an ethernet interface of 100Mbps. The base images run Linux CentOS 7.1.1503 with kernel 3.10.0-229.4.2.el7.x86_64. To simulate the network latencies experienced by the developers, application deployment requests are issued from outside our cluster in a home network featuring a download rate of 55.7 Mbps and an upload rate of 5.6 Mbps. Requests by the clients of operational administrators and auditors are issued within our cluster sharing a network bandwidth of 100Mbps.

Performance of attestation. Attestation is performed by auditors to the auditing hub and by the auditing hub to minion nodes. We use microbenchmarks to evaluate the performance of attestation. The measured time for both these operations is about 4 seconds. Nearly 24.6% of the overall attestation time is taken up by the quote operation of the TPM, while the remaining time is spent in cryptographic operations and network round-trip time of P-Cop protocol messages. Given that attestation operations have a relatively low duration and occur infrequently, they contribute with a residual impact in the initialization latency of PaaS services.

Performance of application deployment. To evaluate the deployment time of applications, i.e., the time since the developer developers an application and the application is instantiated on the cluster, we used two PHP applications of different sizes and complexities: *Hello* and *OSN*. Hello is a simple PHP application that consists of a single static web page (1.5Kb) and is representative of the baseline deployment time for Docker-based application. OSN is an open source online social network application (11MB) which is representative of a complex and heavyweight application. Both applications run on official Apache Web server (version 2.4) container. We test each application changing the number of container instances on the cluster.

Table I shows the overall deployment times for five instances of Hello and OSN broken up into three main components: 1) file transfers between developer and monitor, and between monitor and minion nodes, 2) container setup, and 3) P-Cop operations (SSL connections, exchanged messages, and P-Cop specific cryptographic operations). The results

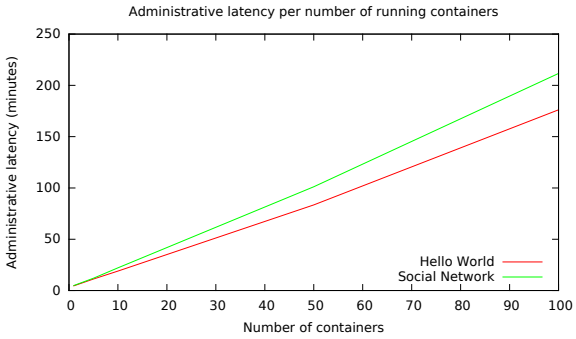

Figure 2. Administrtive latency for n instances of Hello and OSN running

show that the main source of overhead comes from file transfers and application deployments, i.e., SSL file copy and Docker daemon, leaving less than 25 seconds for P-Cop operations, which represent 5% and 2% of the total deployment time for Hello and OSN, respectively.

Performance of super-sessions. For space constraints, we focus on the super-session opening time, which can be quantitatively measured. To measure this value, which determines how much time an operational administrator needs to wait until he can log into a minion node with superuser privileges, we deployed multiple instances of a test application on a given minion node and initiated a super-session. Before the super-session can be established, the node is first sanitized and the applications are migrated to another node. In order to simulate the protection of the node against snooping and tampering, we used the Docker daemon to remove all Docker images and containers on the node (including the base images), and deleted all the application files sent by the monitor and used to build the containers. We repeated the same experiment for both Hello and OSN applications.

Figure 2 shows the measured super-session opening time. Similarly to application deployment, the time grows linearly with the number of instances on the node. The time difference between OSN and Hello is smaller than the respective deployment times (in the order of seconds) because no file transfers need to be performed between the developer and the monitor. Table II presents the contributions of several sources to the opening time of a super-session to a minion node running five application instances. Four components are indicated: 1) file transfers between monitor and minion nodes, 2) redeployment of applications on alternative minion nodes, 3) removal of application files and containers from current minion node, and 4) P-Cop operations. We can see that the most significant fraction of the measured times where spent by Docker in redeploying and deleting containers, namely 98% and 93% for Hello and OSN, respectively. Most of the time taken by the purging process comes from the Docker daemon.

Applications	File transfers	Application deployment	minion node purge	P-Cop operations	Total
Hello	0.085 seconds	05:00	05:54	00:13	11:07
OSN	0.23 seconds	05:45	05:45	00:14	11:44

Table II
DISCRIMINATION OF MANAGEMENT REQUESTS' LATENCIES FOR 5 APPLICATIONS OF HELLO AND OSN (MINUTES:SECONDS)

B. Security Analysis

P-Cop provides confidentiality and integrity protection of guest containers for PaaS services. P-Cop protects developer applications and user data from potentially malicious operational administrators of the cloud infrastructure. Such protection is attained by leveraging TPM attestation, cryptographic techniques, and operating system access control mechanisms in the design of P-Cop.

However, P-Cop relies on a set of assumptions which, if violated, may result in security breaches. P-Cop does not protect against software exploits in the minion nodes or in the software of the auditing hub. As a result, the auditor plays a critical role in order to rapidly identify faulty software and require well tested or even certified software. If the applications themselves are malicious, the users cannot also benefit from the protections offered by P-Cop: a malicious application with direct access to users' data can easily tamper with it or exfiltrate it. It is, therefore, essential for users to verify the identity and reputation of developers before using their applications. In P-Cop, the auditors are also a crucial pillar in the system's root of trust. Ill-intended or negligible auditors can seriously compromise the security assurances offered by P-Cop. To mitigate this risk, the auditor role must be performed by a collective entity in which the participation of independent individuals is required to avoid collusion. P-Cop cannot offer protection against an adversary with physical access to the hardware, and in particular that can subvert the TPM chip. We rely on out-of-band mechanisms to guarantee the integrity of the cloud hardware.

VII. RELATED WORK

The line of research that is mostly related with P-Cop involves using trusted computing techniques to enhance security and trust in the cloud. This general approach, which was introduced by Santos et al. [6], has led to the development of cloud attestation systems such as CloudVerifier [4] and Excalibur [3] which allow for cloud customers to remotely attest the software infrastructure of a cloud service based on TPM chips deployed on the cloud nodes. When coupled with hardened hypervisors [7], [1], cloud attestation systems enable customers to outsource their computations into the cloud while ensuring that the operational administrator cannot inspect nor interfere with the customers' computations. These systems, however, operate at the virtual machine abstraction layer and are too strict in the kind of privilege restrictions imposed to the administrators, making this solutions appropriated to IaaS, but limited for PaaS clouds, which is the focus of P-Cop.

Targeting exclusively PaaS services, Brown et al. [8] propose a trusted PaaS platform to address the lack of transparency by application end-users users. In contrast to P-Cop, however, their concern is about developers that may deploy malicious or buggy applications onto the cloud, and not about untrusted operational administrators. Thus, P-Cop provides complementary security assurances to this system.

Another related topic is about ways to improve the security of cloud administration. Butt et al. [9] propose a self-service cloud computing platform in which customers themselves can specify low-level configurations for the cloud infrastructure, but require a virtualized IaaS infrastructure. Secure Cloud Maintenance [10] supports different levels of administrator privileges on the minion nodes, but depend on a fully trusted administrator to configure an underlying SELinux OS; P-Cop overcomes this restriction.

P-Cop's auditing capability is also related with secure logging systems. Systems like H-One [11] are based on a hardened hypervisor to log administrator operations using information flow tracking when managing guest VMs. Sinha et al. [12] do not rely on hypervisors, but rather leverage TPM for protecting the integrity of logs. Such techniques are orthogonal to P-Cop and can be used to improve the security of P-Cop's auditing hub.

VIII. CONCLUSION

This paper presents P-Cop, a Platform-as-a-Service (PaaS) cloud management system that provides container isolation from operational administrators while providing flexible maintenance capability. P-Cop prevents untrusted operational administrators from accessing guest containers and therefore cause security breaches. To preserve the maintenance flexibility, P-Cop keeps access privileges restricted on cloud nodes, but allow operational administrators to elevate their privileges on cloud nodes. To preserve the security, P-Cop provides mechanisms that allow a third-party auditor to verify that the maintenance operations are safe. Although P-Cop was targeted toward Docker-containerized PaaS, it can generally be applied to other PaaS services.

REFERENCES

[1] F. Zhang, J. Chen, H. Chen, and B. Zang, "Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization," in *Proc. of SOSP*, 2011.

[2] T. C. Group, "TPM Main Specification Level 2 Version 1.2, Revision 130," 2006.

[3] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu, "Policy-sealed Data: A New Abstraction for Building Trusted Cloud Services," in *Proc. of USENIX Security*, 2012.

[4] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel, "Seeding clouds with trust anchors," in *Proc. of WCCS*, 2010.

[5] "Docker," https://www.docker.com.

[6] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proc. of HotCloud*, 2009.

[7] J. M. McCune, Y. Li, N. Qu, Z. Zhou, A. Datta, V. D. Gligor, and A. Perrig, "TrustVisor: Efficient TCB Reduction and Attestation," in *Proc. of IEEE S&P*, 2010.

[8] A. Brown and J. S. Chase, "Trusted Platform-as-a-Service: A Foundation for Trustworthy Cloud-hosted Applications," in *Proc. of CCSW*, 2011.

[9] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service Cloud Computing," in *Proc. of CCS*, 2012.

[10] S. Bleikertz, A. Kurmus, Z. a. Nagy, and M. Schunter, "Secure Cloud Maintenance," in *Proc. of ASIACCS*, 2012.

[11] A. Ganjali and D. Lie, "Auditing cloud administrators using information flow tracking," in *Proc. of CCS*, 2012.

[12] P. England, L. Jia, J. Lorch, and A. Sinha, "Continuous Tamper-proof Logging using TPM2.0," in *Proc. of TRUST*, 2014.