

# P-Cop: A Cloud Administration Proxy to Enforce Bipartite Maintenance of PaaS Services

Bruno Braga, Nuno Santos

*INESC-ID / Instituto Superior Técnico, Universidade de Lisboa*

*Lisbon, Portugal*

*Email: brunobraga@ist.utl.pt, nuno.santos@inesc-id.pt*

**Abstract**—Platform-as-a-Service (PaaS) infrastructures are highly dependent on cloud administrators. 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 access privileges on cloud nodes. However, this approach is overly inflexible for PaaS services, since superuser privileges are commonly required to apply a security patch, change a firewall rule, etc. This paper presents P-Cop, a system aimed to provide secure PaaS maintenance while preserving administration flexibility. To that end, P-Cop implements a *bipartite maintenance model* in which cloud administrator privileges can be elevated to superuser on a given node, but no sensitive guest computations can be allocated to the node until the issued command sequence has been endorsed by an auditor, i.e., a third-party trusted mutually by cloud provider and clients. P-Cop relies on a trusted proxy which supervises privileged commands issued by cloud administrators. Our current P-Cop design targets Docker-containerized PaaS services and leverages TPM hardware to enable remote attestation by external clients.

## I. INTRODUCTION

Cloud mismanagement is an often overlooked security risk that Platform-as-a-Service (PaaS) customers are faced with. To maintain the software systems of PaaS infrastructures, cloud administrators hold privileged access to cloud nodes, including to those where client computations take place. As a result, they have access to customer data. Although in the vast majority of cases cloud 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 cloud administrators from obtaining root privileges entirely. Systems such as CloudVisor [1], [2] or BrokulOS [3] consist of hardened hypervisor or operating system, respectively, 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) [4] hardware and cloud attestation services [5], [6] 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 cloud administrators from acquiring superuser privileges whenever necessary. The key idea to achieving this property is to enforce a bipartite maintenance model between cloud 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, cloud administrators have limited privileges in the common case. In the cases where superuser privileges are necessary, cloud administrators can request privilege elevation so that they can log into cloud nodes as root and execute arbitrary commands. In such cases, commands must first be endorsed by the auditor before they can be definitely accepted. To prevent data breaches, from the moment a cloud 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 cloud administrators and supports their respective endorsement by the auditor. To acquire privilege elevation on a given cloud node, a cloud 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 cloud administrator can execute arbitrary root commands (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 retrieve the log from the proxy in order to analyze and endorse the super-session commands if no security vulnerabilities have been introduced, or revert them otherwise.

P-Cop design addresses several technical challenges. 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 cloud administrators and their changes are reflected onto the PaaS clusters. The extent of that lag depends mostly on how tightly coupled cloud administrators and auditors operate.

## II. BACKGROUND AND GOALS

Containerized Platform-as-a-Service (PaaS) provides hosting cloud services for customer applications within *containers*. Containers are supported on a given server by software frameworks such as Docker [7], which modify the OS so as to enforce proper isolation between containers and manage their lifecycle. In a containerized PaaS infrastructure, the *compute* nodes (also named *minions*) is the place where guest containers execute. 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. The *monitor* nodes implement the logic that ties together the entire service: they process container deployment requests, allocate containers to minions, manage resources, etc.

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

The goal of our work is to enable the design of PaaS services that can prevent access to guest containers by untrusted cloud administrators. In spite of such restrictions, our solution must allow for flexible administration of the cloud by allowing cloud administrators to obtain superuser privileges whenever necessary. Thus, we assume that cloud administrators are untrusted. A cloud administrator can remotely reboot or power-cycle any of the cloud nodes to

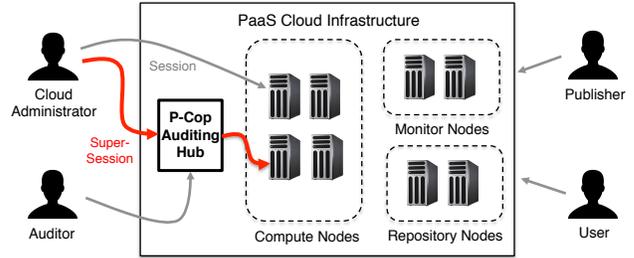


Figure 1. P-Cop architecture.

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 cloud 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. For example, the operating system can be hardened so that the administrator cannot obtain superuser privileges [3]. 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 and the respective public key is certified by the cloud provider. This certification ascertains the property of that chip (and respective server) by the cloud provider. The certification of such keys is performed when the hardware is deployed within the cloud provider's premises. We require that the TPM chip and the cryptographic algorithms used in our solution are correct.

## III. ARCHITECTURE

We present P-Cop (PaaS Cop), a PaaS cloud management system that provides container isolation from cloud administrators while providing flexible maintenance capability. In our solution, rather than permanently restricting administration privileges of compute nodes, we allow such privileges to be temporarily elevated in order to let administrators log into a particular compute 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 node software and therefore the node can be trusted to securely accommodate PaaS containers. Simply put, P-Cop enforces a bipartite maintenance policy between cloud 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 keeping track of the software state of the compute nodes and managing super-sessions. Super-sessions consist of tunneled SSL connections over which a cloud administrator can access a compute node with root privileges. In regular sessions, the cloud administrator does not have root privileges. Another component of P-Cop is the *node guard*, which is a software agent that runs on each compute node and implements local actions upon 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 compute nodes running a *trusted container runtime*. A trusted container runtime is a “containerized” software stack that satisfies container isolation properties. In particular, when such a software stack is installed on a compute node, cloud 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 compute 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 container runtime software. This is achieved by requiring the software to be validated and signed by the auditor and then leveraging TPMs to attest that such software is running on the cloud nodes. P-Cop provides the means for publishers 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 compute nodes that belong to this list eligible for hosting guest containers.

### B. Privileged Operation Mode

In case a cloud 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 trusted container runtime allowing the cloud 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 cloud administrator, and 2) removes the node from the verified list to avoid future instantiation of guest containers on compute

```
Nov 17, 2015 11:14:05 AM
ALL: Admin->Host:
service docker restart

Nov 17, 2015 11:14:06 AM
ALL: Host->Admin:
docker stop/waiting
docker start/running, process 6101
```

Figure 2. Example of a simple session log.

nodes that are currently controlled by the cloud administrator and, therefore, are potentially insecure.

When the cloud administrator obtains access to a compute 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 cloud administrator during the super-session. In particular, the log contains a record of all input commands provided by the cloud 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 cloud 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 compute 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, the auditor is responsible for certifying the software of the PaaS infrastructure. In addition to the validation of super-session logs, the auditor must validate the software of the trusted 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 cloud administrators. Instead, it must be assigned to a third party that is mutually trusted by the cloud provider, publishers, 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, 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. For symmetric keys, we drop the superscript. 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  $\{PCR \parallel n\}_{AIK}$ , where  $\parallel$  denotes string concatenation.

### A. System Initialization

To initialize the system, cloud administrators must first install the software of the auditing hub. The auditing hub provides a well defined set of interfaces and offers no special privileges to cloud administrators. However, by installing the software on the servers, a cloud administrator can easily manipulate the configuration of the auditing hub so as to escalate his privileges and undermine P-Cop operations.

To ensure the correct setup of the auditing hub, the auditing hub logic must be distributed as a software image that is certified and signed by the auditor. Such an image contains all the software components that are necessary in order for the auditing hub to be operational. The installation process is streamlined so that all software components are automatically installed and configured on the targeted server. In the bootstrapping process, a unique fingerprint is generated that refers to the original certified image.

Secondly, once the auditing hub boots for the first time, it requires an activation step that must be triggered by the auditor. This activation step includes a remote attestation process that collects the fingerprint of the auditing hub. 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 the *hub key* ( $HK$ ): a keypair that will serve as an identity of P-Cop for this particular PaaS service. To declare the authenticity of  $HK$ , the auditor issues a *hub key certificate*  $C_{HK}$ . This certificate plays an important role in most P-Cop protocols for authentication purposes.

### B. Bootstrapping and Attestation of Compute Nodes

Compute nodes are responsible for hosting guest containers of the PaaS service. After the auditing hub has been properly initialized, one of its main roles is to check the software configuration of the compute nodes and verify which of them have been properly set up with a trusted container runtime. Only such nodes can be part of the verified compute node list, which identifies the nodes that are authorized to accommodate guest containers.

Similarly to the auditing hub, the software of compute nodes must be installed by a cloud administrator, which can

potentially introduce vulnerabilities in their configuration and, consequently, introduce container isolation breaches. To mitigate such threats, P-Cop requires that the auditor certifies the images of trusted container runtime software. Then, every time the cloud nodes bootstrap, the auditing hub executes a remote attestation protocol to verify that the installed image is authentic and its integrity is intact. Only the cloud nodes that pass this test can be authorized to host PaaS containers.

To certify a trusted container runtime image, the auditor must validate the software configuration of the image and then sign a certificate that includes the hash value of the image. The auditor must identify all software components in the system, ensure that it includes P-Cop's node guard software, and that the operating system is hardened so as to enforce container isolation.

To attest a compute node, the auditing hub engages with the target compute node in simple remote attestation protocol. The auditing hub obtains a quote from the compute 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 trusted container runtime, then the attestation fails. Otherwise, the compute node is considered to be trusted. As a token of authenticity, the compute node creates a *node keypair* ( $NK$ ), whose private part is kept by the node guard in volatile memory and the public part is signed by the auditing hub using its  $HK$  private key. Signatures by the  $NK$  key demonstrate that the compute node has been properly attested. Since the private key is kept in memory, if the compute node is rebooted, the key is lost and a new attestation must be performed to reevaluate the software and generate a new key.

### C. Secure PaaS Operations

A typical container-based PaaS service implements a set of core operations aimed to support the lifecycle of guest applications. Publishers start by deploying their applications onto the service. The deployment process is coordinated by the monitor, which authenticates the publisher, receives an application package and a *docker file*, and stores them both on the repository cluster associated with the publisher's account. 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 compute node from the compute node pool, and uploads to it the application package and the docker file. The compute node must then create a new guest container for the application. To this end, the compute node must first build a container image by retrieving a copy of a base image from the repository and then extending that copy according to the publisher-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 publisher 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 to provide two guarantees. First, publishers must be assured that their applications can be instantiated only on trusted compute nodes, i.e., compute nodes properly configured with a certified trusted container runtime (G1). Second, users must be ensured that the applications they are accessing are authentic, can be properly identified, and execute on a trusted compute node (G2).

To provide guarantee G1, P-Cop involves the auditing hub and the node guard software running on trusted compute nodes. As we mentioned in the section above, trusted compute nodes have been attested by the auditing hub. As a result of that attestation process, they all have a keypair  $NK_i$  that can be used to authenticate the trusted computing node  $i$ . Authentication is possible because the private part of the key remains private to the trusted node and the public key can be validated as belonging to a compute node validated by the auditing hub. This validation can be performed because the auditing node issues certificates  $C_{Ni}$  which contain the public part of  $NK_i$  signed by the auditing hub’s private key ( $HK$ ). P-Cop uses  $NK$  keys to securely distribute an *application key*  $AK$  across all trusted compute nodes. The  $AK$  is a symmetric key that the publisher must use to encrypt the application package and the docker file before deploying the application. To instantiate the application, a compute node must first decrypt both application package and docker file, operation for which it needs to retrieve the application  $AK$  key. However, P-Cop only releases the  $AK$  key to the nodes that have been properly attested. The way it works is, upon deployment, the publisher generates an  $AK$  locally and sends to the monitor the message:  $\{App\}_{AK}\{AK\}_{HK}$ , which includes the application package and docker file encrypted with  $AK$  and the key  $AK$  encrypted with the public key of the auditing hub. The monitor forwards the encrypted  $AK$  to the auditing node. The auditing node decrypts the key and distributes the key to every trusted node  $i$  by encrypting  $AK$  with the public part of  $HK_i$  and sending it to the trusted node. The node receives the encrypted key, decrypts it using the private part of  $AK$ , and keeps it in memory for future instances of the application. Thus, the instantiation of the application cannot be performed on the nodes that are not running a trusted container runtime.

To provide the guarantee G2, P-Cop allows users to obtain a cryptographic token that enables them to check the authenticity and security of the PaaS application. Whenever the application is installed in the guest container, a new SSL

keypair is created for HTTPS connections initiated by the users, and the public SSL key is certified by the local node guard. The node guard creates a certificate  $C_K$  which includes identifying information of the application signed with the compute node’s  $NK$ . The first time the user connects to the application over HTTPS, he can verify the security of the application by obtaining a token which consists of a set of certificates:  $C_K C_{HK} C_{NK} C_{AK}$ . First, these certificates allow for determining the identify of the application and respective publisher by direct inspection of  $C_K$ . Second, it is possible to determine that the application host can be trusted by validating the chain of certificates. These certificates must link the public SSL key to a compute node ( $C_K$ ), whose configuration must have been attested by an auditing hub ( $C_{HK}$ ), which in turn is expected to have been validated by an auditor ( $C_{NK}$ ) owning a certified identity ( $C_{AK}$ ). If these conditions are met it is safe for the user to access the application.

#### D. Establishment and Endorsement of Super-Sessions

In order for cloud administrators to gain full access to the compute nodes as root they must open a super-session. To provide super-session support while preserving the invariant that applications are confined to trusted compute nodes, P-Cop must provide several guarantees. First, a super-session must only be granted to a cloud administrator once it has been assured that the targeted compute node is properly sanitized. In particular, the compute node can not leave remnants of publisher or user data from guest containers or sensitive P-Cop cryptographic material, namely the keys  $HK$ ,  $AK$ , and  $SK$  (G1). Second, P-Cop must ensure that applications cannot be instantiated on a compute 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 cloud administrators. Such a record trail is fundamental to ensure that the auditor can trace all configuration changes performed to the compute 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 compute nodes are hardened, any attempts to log into a trusted compute node with root privileges will fail. Instead, the cloud 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 compute node is properly sanitized before root access can be granted (G1). To this end, the auditing node sends an authenticated cleanup message to the compute node. The message receiver—the node guard—stops any existing guest containers, cleans up the remnants of publisher 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.

Before the super-session can be authorized, must in addition take proper actions to preserve the invariant that applications can be hosted only on compute nodes featuring a trusted container runtime. To preserve this invariant (G2), the auditing hub keeps track of the trusted nodes on the list of verified nodes, which identify the pool of attested compute nodes. 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 cloud administrator’s client and makes an SSH connection to the target compute node. The auditing hub then works like an SSH proxy which tunnels all messages exchanged between the client and the compute node during the super-session. Since the trusted compute 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 cloud administrator, the auditing hub makes a log of the super-session traffic. To preserve the order of the log history, when the super-session ends, the auditing hub computes the hash chain:  $h_i(s_i || h_{i-1})$ , where  $s_i$  is the hash of the most recent super-session log and  $h_0$  corresponds to the PCR hash of the trusted container runtime. By inspecting the hash chain and analyzing the content of the super-session logs, an auditor can then reconstruct the entire configuration history of the compute node and endorse or reject the changes.

## 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 container runtime 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 publisher, administrator, and auditor, and for issuing requests to the auditing hub and to the compute nodes. The PaaS repository is provided by the monitor and

the minions 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 measure 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 publishers deploy their applications on the PaaS service and the application becomes available to users, and 3) opening 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 compute 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 publishers, 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 cloud 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 compute 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 publisher publishes 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

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)

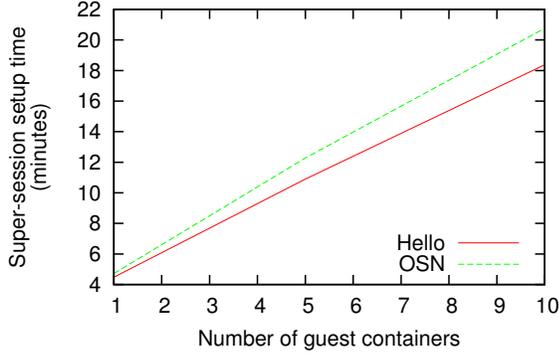


Figure 3. Administrative latency for  $n$  instances of Hello and OSN running

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 minions, 2) container setup, and 3) P-Cop operations (SSL connections, exchanged messages, and P-Cop specific cryptographic operations). The results 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 a cloud administrator needs to wait until he can log into a minion with superuser privileges, we deployed multiple instances of a test application on a given minion 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 3 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 publisher and the monitor. Table II presents the contributions of several sources to the opening time of a super-session to a minion running five application instances. Four components are indicated: 1) file transfers between monitor and minions, 2) redeployment of applications on alternative minions, 3) removal of application files and containers from current minion, and 4) P-Cop operations. We can see that the most significant fraction of the measured times were 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.

### B. Security Analysis

P-Cop provides confidentiality and integrity protection of guest containers for PaaS services. P-Cop protects publisher applications and user data from potentially malicious cloud 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 minions 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 publishers 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. [8], has led to the development of cloud attestation systems such as CloudVerifier [6] and Excalibur [5] 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 [1], [9], cloud attestation

Applications	File transfers	Application deployment	Minion 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)

systems enable customers to outsource their computations into the cloud while ensuring that the cloud 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. [10] 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 publishers that may deploy malicious or buggy applications onto the cloud, and not about untrusted cloud 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. [11] 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 [12] supports different levels of administrator privileges on the compute nodes, but depend on a fully trusted administrator to configure an underlying SELinux OS; P-Cop overcomes this restriction. Hardened operating systems such as BroKulOS [3] implement fine-grained privilege separation for Linux administrators and can be used for building trusted container runtimes for P-Cop.

P-Cop's auditing capability is also related with secure logging systems. Systems like H-One [13] are based on a hardened hypervisor to log administrator operations using information flow tracking when managing guest VMs. Sinha et al. [14] 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 cloud administrators while providing flexible maintenance capability. P-Cop prevents untrusted cloud 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 cloud 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] D. G. Murray, G. Milos, and S. Hand, "Improving Xen security through disaggregation," in *VEE*, 2008.
- [3] N. Santos, R. Rodrigues, and B. Ford, "Enhancing the OS against Security Threats in System Administration," in *Proc. of Middleware*, 2012.
- [4] T. C. Group, "TPM Main Specification Level 2 Version 1.2, Revision 130," 2006.
- [5] 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.
- [6] J. Schiffman, T. Moyer, H. Vijayakumar, T. Jaeger, and P. McDaniel, "Seeding clouds with trust anchors," in *Proc. of WCCS*, 2010.
- [7] "Docker," <https://www.docker.com>.
- [8] N. Santos, K. P. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *Proc. of HotCloud*, 2009.
- [9] 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.
- [10] A. Brown and J. S. Chase, "Trusted Platform-as-a-Service: A Foundation for Trustworthy Cloud-hosted Applications," in *Proc. of CCSW*, 2011.
- [11] S. Butt, H. A. Lagar-Cavilla, A. Srivastava, and V. Ganapathy, "Self-service Cloud Computing," in *Proc. of CCS*, 2012.
- [12] S. Bleikertz, A. Kurmus, Z. a. Nagy, and M. Schunter, "Secure Cloud Maintenance," in *Proc. of ASIACCS*, 2012.
- [13] A. Ganjali and D. Lie, "Auditing cloud administrators using information flow tracking," in *Proc. of CCS*, 2012.
- [14] P. England, L. Jia, J. Lorch, and A. Sinha, "Continuous Tamper-proof Logging using TPM2.0," in *Proc. of TRUST*, 2014.