PALANTIR: Zero-trust architecture for Managed Security Service Provider

PALANTIR: Zero-trust architecture for Managed Security Service Provider

Maxime Compastié [1], Silvia Sisinni [2], Supreshna Gurung [3], Carolina Fernández [1], Ludovic Jacquin [3], Izidor Mlakar [4,5], Valentino Šafran [4], Antonio Lioy [2] and Ignazio Pedone [2]

[1] *i2CAT Foundation, C\ Gran Capità 2-4 Edifici Nexus I, Barcelona, Catalonia, Spain*
[2] *Politecnico di Torino, Corso Duca degli Abruzzi, 24, 10129, Turin, Italy*
[3] *Hewlett Packard Enterprise, 1 Enterprise Park, Long Down Avenue, Stoke Gifford, BS34 8QZ, Bristol, UK*
[4] *University of Maribor Faculty of Electrical Engineering and Computer Science, Koroška cesta 46, 2000 Maribor Slovenia*
[5] *Sfera IT d.o.o., Beloruska ulica 7, 2000 Maribor, Slovenia*

Abstract

The H2020 PALANTIR project aims at delivering a Security-as-a-Service solution to SMEs and microenterprises via the exploitation of containerised Network Functions. However, these functions are conceived by third-party developers and can also be deployed in untrustworthy virtualisation layers, depending on the subscribed delivery model. Therefore, they cannot be trusted and require a stringent monitoring to ensure their harmlessness, as well as adequate measures to remediate any nefarious activities. This paper justifies, details and evaluates a Zero-Trust architecture supporting PALANTIR's solution. Specifically, PALANTIR periodically attests the service and infrastructure's components for signs of compromise by implementing the Trusted Computing paradigm. Verification addresses the firmware, OS and software using UEFI measured boot and Linux Integrity Measurement Architecture, extended to support containerised application attestation. Mitigation actions are supervised by the Recovery Service and the Security Orchestrator based on OSM to, respectively, determine the adequate remediation actions from a recovery policy and enforce them down to the lower layers of the infrastructure through local authenticated enablers. We detail an implementation prototype serving a baseline for quantitative evaluation of our work.

Keywords

Trusted Computing, Zero-Trust, Integrity Measurement, Remediation, Managed Security

1. Introduction

The recent years have witnessed a flourishing diversity of cyber-attacks and techniques jeopardising organisation's activity and assets. While large companies and public bodies promptly reacted by acquiring necessary skills, processes and tooling to handle such risks, SMEs and microenterprises are facing fierce obstacles due to their limited investment capabilities and manpower to allocate. For those actors, contracting a Managed Security Service Provider (MSSP) has become a practical option to delegate the prevention and the management of cybersecurity incidents. In this context, H2020 PALANTIR project [1] is an innovation action from the European commission and ambitions at conceiving and delivering a solution for MSSPs and organisations' internal usage. The platform exploits security enablers offered as extended virtual network functions (VNF) to leverage the detection and

C&ESAR'22: Computer & Electronics Security Application Rendezvous, Nov. 15-16, 2022, Rennes, France
EMAIL: maxime.compastie@i2cat.net (M. Compastié); silvia.sisinni@polito.it (S. Sisinni); supreshna.gurung@hpe.com (S. Gurung); carolina.fernandez@i2cat.net; (C. Fernández); ludovic.jacquin@hpe.com (L. Jacquin); izidor.mlakar@um.si (I. Mlakar); valentino.safran@um.si (V. Šafran); lioy@polito.it (A. Lioy); ignazio.pedone@polito.it (I. Pedone)
ORCID: 0000-0001-7399-709X (M. Compastié); 0000-0002-0877-7063 (S. Sisinni); 0000-0002-0877-7063 (S. Gurung); 0000-0003-1865-7177 (C. Fernández); 0000-0002-0877-7063 (L. Jacquin); 0000-0002-4910-1879 (I. Mlakar); 0000-0002-3664-3564 (V. Šafran); 0000-0002-0877-7063 (A. Lioy); 0000-0002-0877-7063 (I. Pedone);

mitigation of malevolent activities on the customer's premise. To ensure PALANTIR extensibility, they are proposed in an as-a-service market open to contributions from third-party developers.

However, if not properly controlled, inviting external contributions exposes PALANTIR platform to the risk of being weaponised against their customers: not only a security service could deliberately be converted into a threat vector (e.g. disguised malware), but developed appliances remain subject to unintended software flaws likely to be exploited by a malevolent actor. These defects can appear at design time as a result of limited quality assurance and secured development practices, but also during their operation, as software vulnerabilities are discovered. Another risk item is induced by the PALANTIR infrastructure hosting the deployed security enablers: as they encompass distributed resources hosted by both the MSSP (e.g. cloud and edge environments) and the customers (e.g. customer premise equipment), they represent attractive targets. This is so since their disruption will impact the reliability of deployed security enablers (as stated by governmental cybersecurity agencies [2]), while offering a tremendous opportunity for lateral movements to access several organisations at once. In practice, if an attacker manages to gain control of a security enabler instance or the physical node hosting it and compromise its components (for example, configuration, executable files, sections of the platform's firmware or kernel), this impacts the reliability of the security services offered to SMEs and MEs, undermining their cyber-protection capabilities and exposing their assets to further threats. Attacks on critical infrastructures are nowadays considered unrelenting and increasingly sophisticated while PALANTIR platform carries a wide exposure to numerous stakeholders and potential malevolent actors. Consequently, PALANTIR's security paradigm does not assume that the infrastructure nodes and service instances are trusted a priori but it enforces that they prove their identity and integrity before being deployed, as well as periodically, throughout the entire life cycle of their operations.

Given the prevalence of the MSSP concept in the delivery models offered by PALANTIR and the potential distribution of some elements in the stack, it is necessary to define a trust model that can assess the integrity level of any asset under the PALANTIR protection, disregarding the location of its deployment. The effective enforcement of such model is also contemplated, considering security orchestration techniques. This paper delves into the main question on how to elaborate a trust model for a distributed MSSP. The contributions of this work span (i) the definition of the trust model for an MSSP deployment, (ii) the assessment strategies that ensure continued integrity of the assets, (iii) the orchestration techniques and interactions in place to enforce these strategies, and finally details (iv) the implementation and evaluation of the technical stack contributing to the fulfilment of the integrity assessment in the ZTA architecture within the project.

The remainder of this paper is organised as follows: Section 2 discusses previous trust assessment research initiatives in the cloud-to-edge continuum, Section 3 comprehensively exposes the foundations of PALANTIR's ZTA architecture. We present our implementation and evaluation work in Section 4. Section 5 introduces the current limitations of our approach and proposes several options to tackle them. Finally, Section 6 concludes this paper and provides paths for future work.

2. Related work

The management of trust in distributed environments has been already largely explored in the literature. Specifically, the software network community has been confronted early to the involvement of multiple parties sharing the same infrastructure with possible competing objectives. For instance, the European Telecommunications Standards Institute (ETSI) has published a threat analysis [3] affecting NFV networks from the perspective of different deployment models and stakeholders. This work identifies several areas of concerns including the trusted boot technologies and the user/tenant authorisation and authentication; but does not provide specific guidelines to tackle them. The management of multiple parties bringing their own constraints represents a certain complexity to cope with, as involved parties are becoming more and more technically diversified and their number increase. Darabseh et al. [4] have proposed an initial framework to decouple the security decision-making process from the enforcement on network devices. This approach permits to optimise the decision-making process to adapt it with the variability of the network topology. In PALANTIR, we consider several threats identified by ETSI, since Security Capabilities (SCs) are extending VNF design and aligned with some proposed deployment model. We also separate the security management from the enablers

in the PALANTIR architecture and apply this principle to the whole management of cybersecurity incidents by exploiting programmable SCs.

At the scale of a single node in distributed environment, virtualisation technologies can play a key role to maintain trust through resource isolation. In [5], the authors analyse the vulnerabilities and possible attacks at different levels of the system architectures. Their threats models include an attacker controlling the host & virtualisation layer or initially subduing a virtualised resource. They conclude on the necessity of (i) integrating security mechanisms in resources needing protection (ii) minimising their attack surface and (iii) leveraging an adequate security management aligned with the security posture. Specifically, as per the integration of security mechanisms, involving a hardware root-of-trust has proven to leverage trust assessment from the system layer perspective or from the virtualised resource perspective. In the latter case, Haven [6] is a solution defending application against malevolent actors having control of the operating system and lower system layer by leveraging Intel SGX enclaves. In opposition to vendor-locked solutions, the Trusted Computing Group (TCG) sustains industry standards for vendor-agnostic hardware root-of-trust specification, noticeably the Trusted Platform Module (TPM) [7] and associated methods, such as TPM 2.0 Keys for Device Identity and Attestation [8], or protocols, such as the Trusted Attestation Protocol [9]. To shrink the attack surface, several works explore the development of sensitive resources. Initial efforts such as SecureUML [10] and Model-driven security approach [11] aim at an extensive specification of security constraints on the application conception to deliver strong security guarantees, but these approaches are component specific and requires a significant effort to be established. Recently, the software supply chain has gained more visibility as a threat vector. Cappos et al. [12] highlight the criticality of package managers in spreading compromised artefacts. The work presented in [13] exploits the unikernel architecture to produce and operate security-constrained resources not needing in-situ package management, contributing to reducing supply-chain risk to the design phase of the resource only. More generally, ENISA has published a report [14] detailing the threat landscape of supply chain attacks. PALANTIR complements these approaches by leveraging an adequate security management not only for the assets of the MSSP's subscribers but also to the security enablers in charge of their protection.

Since 2010 [15], the term of "Zero-Trust" has emerged to refer the idea that no participant in a network should be trusted. However, the concept that a participant in a computer network has its access systematically mediated has been introduced by Saltzer and Shroeder [16] since 1975. Recently, the conception of system architecture applying Zero-Trust principle by default in opposition to perimeter-based trust management has gained momentum and has been explored by both academia and engineering community. NIST [17] has published in 2020 a comprehensive report detailing the founding concepts of Zero-trust architectures and their expected benefits regarding the exposure of systems to the threat landscape. It retained seven main tenets: (i) services and data sources are equally considered as resources, (ii) communication shall be secured disregarding specific network location, (iii) access to resources is granted on a per-session basis and (iv) evaluated dynamically based on the current attributes of the subject, (v) assets are continuously evaluated on their integrity and security posture, (vi) authentication and authorisation are systematically checked before access is granted, and (vii) the constant collection of data to evaluate the security posture of a system. Buck et al. [15] complement this work by reviewing the current state of knowledge on ZTA from both academia (peer-reviewed literature) and practice (grey literature), underlining an unequal adhesion of the aforementioned principles among communities. From a more practical standpoint, several work has been carried to elaborate Zero-Trust strategies when applied to specific technical context and verticals: the work presented in [18] introduces a security framework for 5G healthcare extending the commonly access control scheme relying on “subject” (humans, devices or apps) and “object” (resources) with the dimensions of “behaviour”, based on the approach based on User and Entity Behaviour Analytics (UEBA), and the “environment”; both incorporating historical data gathered by external intelligence and monitoring. The work in [19] reviews the situation in the vehicular networks and identifies common attacks, the means to enforce trust between components, relevant environment-related attributes to exploit (such as similarity, familiarity, frequency, and duration), and where to favour the trust posture (e.g. data-centric focusing on data accuracy, and legitimacy, whereas entity-centric relies on reputation). More specific work has covered the application of Zero-Trust tenets to specific environment. For instance, Vanickis et al. [20] explored the Zero-Trust-Networking (ZTN) via the proposition of access control framework sanctioning the access to the network zones. The work elaborates a domain-specific

language to specify access control policy complying with several Zero-Trust guidelines, and a monitor to enforce the decisions by dynamically adapting network configurations. Due to their inherent exposure, cloud infrastructures have become an application field of choice for Zero Trust. The contribution from [21] details a didactic model for constructing a Zero-Trust services in cloud environments. Our work capitalises on the experience shared from the literature to provide an architecture and implementation suited for MSSP usage, abiding with diversified deployment environments for security enablers such as cloud, edge, and on-premises infrastructures.

3. PALANTIR Zero-trust architecture

3.1. Threat model and prerequisites

PALANTIR envisions delivering a SecaaS platform to enable a service provider to oversee the security of the subscribers' assets. Security is enacted by granular enablers, i.e. security capabilities (SCs), deployed on an infrastructure under custody of a contracted provider. The SCs are conceived by third-party developers and made available through a marketplace. We propose the following threat model and assume the following actors and behaviours:

The **PALANTIR provider** and its operators can be trusted: they actively contribute to the security of the platform by applying proactively and reactively mitigations covering the complete spectrum of identified threats. This assertion is acceptable since the provider is the main beneficiary of the platform exploitation whose economic position and reputation are at stake. Specifically, we consider it is enforcing the necessary measures to protect the platform against both external and insider threats. Thus, we assume the platform and its infrastructure are part of the trusted computing base and can confidently support the features needed to maintain the whole deployment in a secured state while offering no exposure to an attacker for exploitation.

The **infrastructure provider** is deemed semi-honest: this actor applies protective measures when contractually obliged and audited but will not necessarily act beyond this scope. In fact, the infrastructure servicing is covered by agreements guaranteeing the level of service. Yet, the PALANTIR provider has no control over the contractor personal and cannot prevent an insider malicious actor. Moreover, as public hosting solutions are typically multi-tenant, the PALANTIR provider has no option to prevent an adversary from using this service and attacking other tenants or the infrastructure. An attacker can therefore target SC instances by tampering the infrastructure layer as (i) an insider of the provider company or as (ii) another subscriber given access to the same infrastructure.

The **SC developer** is also considered as semi-honest. Albeit producing security services to gain a revenue, the PALANTIR provider cannot assess the compliances with secure development practices. This implicates the software may contain flaws susceptible to be exploited. Furthermore, these vulnerabilities may stem from introduced dependencies carrying their own flaws. Therefore, the management of the supply chain by the involved developers represents risk factors. In this situation, an attacker can act as a neglectful developer introducing flaws by deficient quality evaluation, or as an actor of the supply chain inserting vulnerable code in the designed SCs, jeopardising their operation.

The **subscriber**, its collaborators and assets are inherently of lesser trust, as they request the protection services and lack the necessary tooling and practices to prevent or counteract cyberattacks on their own. When being subscribers to a MSSP solution, they may choose not to apply for a full-fledged protection for their IT (e.g. due to budget constraints), maintaining a vulnerable surface area. They represent common targets for threat actors, seeking to access their assets, or to rebound over SCs instances to target the MSSP.

In this context, SCs will seek to access resources on the customer information system and their hosting infrastructure while not being trustable due to their design and exposure. Our approach reverses the traditional use case of ZTA and proposes to entangle ZT principles with the management of SCs instead of the users: their instances and communication are scrutinised to evaluate their security postures and the PALANTIR platform acts as a mediation layer for their interaction with the customer resources.

3.2. Architecture

To handle those prerequisites, the PALANTIR project focuses on the seven tenets on ZTA proposed by NIST in the standard SP 800-207 [17], and specifically on those relating to the continued assessment of the trust status for the asset inventory. The architecture is depicted in Figure 1.

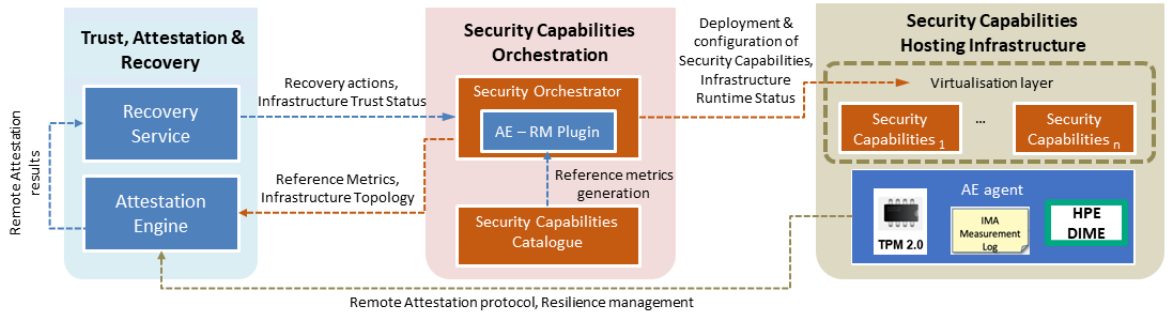


Figure 1: PALANTIR's Zero-Trust architecture

The Trust, Attestation and Recovery (TAR) component continuously monitors the infrastructure and SCs to detect signs of attacks or erroneous behaviour. The TAR is also leveraged by the Security Orchestrator to ensure no untrusted node or capability is used to enforce the PALANTIR SecaaS solution. The TAR comprises the Attestation Engine (AE) and the Recovery Service (RS). The AE carries out the remote attestation of the nodes and SCs. The RS supervises remediation procedures that unify, correlate and automate event handling across the end-to-end physical and virtual infrastructure.

The Security Orchestrator (SO) subcomponent controls the lifecycle of the SC instances and provides context information about their static and runtime data. It ensures their placement, initial deployment setup, configuration and deallocation based on the customer requests and subscriptions. It also serves as the enforcer for countering the activity of compromised resources, thanks to the deployment and configuration of specific capabilities that fulfil the mitigation decisions from the RS.

The Security Capability Hosting Infrastructure (SCHI) represents the set of assets deployed in the PALANTIR platform (physical nodes provided by the infrastructure provider and SC instances running on them) that need to be protected through the ZT security model. The AE is responsible of continuously monitoring the integrity status of all the components in the SCHI, thus verifying that their security posture has not been compromised. When the AE detects an integrity failure, it notifies this event to the RS, which will enforce a suitable remediation in order to recover the security posture of the infrastructure.

3.3. Security orchestration for decision enforcement

Managing the lifecycle and configuration of the security capabilities across different infrastructures is key to offer an automated manner of applying specific decisions, such as the enforcement of specific actions or configurations. SO provides the intermediate orchestration logic that receives the decisions to enforce on the running SCs. To do so, it is first contacted by the upper decision-taking logic provided by RS. Then, the SO coordinates with a third-party Management and Orchestration Software (MANO) named OSM (Open Source MANO) to instantiate the SCs as containers on the SCHI, as well as configure them. The configuration process emits adequate action requests to the SC endpoints, which have the logic to interpret and ultimately apply the action. Besides the instantiation and configuration features, the SO facilitates specific decision enforcement, as well as overseeing and exposing the behaviour of the infrastructure (SCHI) and the SC instances through its multi-layer monitoring; whose values can be used to assess the current status and complement the decision enforcement. These values come from the (i) container runtime and from (ii) the running SC instances themselves, both running on the SCHI. Specifically, and in order to identify the specific nodes subject to integrity measurement (described below), the AE uses a subset of data from (i); whilst other architectural components can request custom measurements from (ii).

At the initial attestation, right after receiving a request to deploy a specific SC, the SO fetches a subset of runtime data from the infrastructure and feeds it to the AE, which in turn performs the integrity check. If the attestation fails, the SO allows a set of actions that can alleviate the risk introduced by the potentially compromised SC instance (i.e. deployment, termination, or specific actions to each SC) and actions to restore the security posture (e.g. redeployment of an equivalent SC). This process also takes place during the periodic attestation of the SC instance, throughout its lifetime; where the AE continuously compares the reference measurements.

The ZT paradigm requires securing communications disregarding the network location. This also applies to the interfaces used by OSM and SO to communicate with the SCs that run on the SCHI; as these require mutual authentication, authorisation, and encrypted communication channels. The following ETSI NFV-compliant interfaces are secured at the SC or SCHI level, respectively: (i) between the VNF Manager (VNFM) and the NFs (Ve-Vnfm-em in ETSI terminology) to be configured through actions; and (ii) between the NFV MANO with the SCHI (Or-Vi in ETSI terminology) to manage the virtual resources subject to orchestration.

3.4. Integrity measurement for Security situation evaluation

The AE uses the Trusted Computing paradigm to continuously monitor SCHI, along with new methods to perform hardware attestation, runtime monitoring and containerised workload attestation. The AE follows the TCG specification to comply with the ZTA foundation; applying the TPM 2.0 Keys for Device Identity and Attestation standard when creating cryptographic keys and certificates used to perform attestation (Initial Attestation Key – IAK) or authentication (Initial Device IDentity – IDevID). The IDevID certificate is signed by the platform manufacturer, and it is complemented by an infrastructure-provided Local Device IDentity (LDevID) that cryptographically identifies the platform's deployment.

The AE also leverages TCG Platform Certificates, issued during manufacturing to establish an authenticated baseline for the hardware when it is first registered with the AE. A hardware measurement capability is added through hardware fingerprinting in UEFI. In the current implementation of the AE, UEFI fingerprints platform hardware by reading serial numbers from the devices including DIMMS, PCIe cards and power supplies. This ensures that the hardware components have not been changed since manufacture – unless an authorised hardware modification happened. In the near future, with the adoption of the Security Protocols and Data Models (SPDM) [22] by the industry, hardware authentication will be done using cryptographic identities.

While Trusted Computing mainly focuses on boot- and load-time measurement, the AE also supports runtime verification. The Distributed Intrusion Monitoring Engine (DIME) [23] kernel module leverages the memory inspection capability of the platform, located on its Baseboard Management Controller (BMC), to detect any unexpected change of code or data already loaded in memory. These measurements along with their corresponding physical memory addresses are monitored continuously to detect any insertion of new code into the kernel, or changes to critical configuration such as the syscall table, that tampers the existing kernel code and critical data structures. When a mismatch is detected, DIME notifies the AE, which alerts the RS to apply the correct recovery policy through the SO.

The Remote Attestation based on TCG's principles is a well-established process for attesting physical nodes, yet it presents several challenges to attest virtual entities. In PALANTIR, the SCs are deployed as containers, following the current trend of lightweight virtualisation techniques, which offer considerable advantages in the management of microservices and guarantee near bare-metal performance. In order to create an attestation solution covering all layers of the SCHI, it has been necessary to tackle the remote attestation of containers, a problem still open for the scientific community. Our solution allows attesting each SC deployed on the SCHI and its host system without depending on specific container runtimes since it relies on properties owned by the containerised processes, which are valid for several containerisation technologies currently in use. Moreover, the solution complies with TCG guidelines as it is based on the TPM chip and the Integrity Measurement Architecture (IMA) module of the Linux kernel and is highly scalable since it does not limit the number

of SCs to concurrently run on a platform. We integrated this solution into the PALANTIR AE which, together with the hardware, firmware and runtime attestation techniques presented above, provides an attestation solution capable of monitoring the entire software stack of a node, thus ensuring the security posture of all assets that are part of the PALANTIR infrastructure.

4. Implementation and evaluation

4.1. SC Integrity measurement

Implementation: The AE keeps monitoring agents distributed throughout the SCHI to assess the trustworthiness of the system. Each monitored node in SCHI hosts an Attestation Agent that is responsible for forwarding attestation information and alerts used by the AE. For example, it extracts the measurements stored in the TPM, signed with a TPM attestation key, and sends them to the AE used to compare against known baseline values. When new SC packages and images are made accessible to the PALANTIR platform, these values are automatically generated by a Reference Measurement plugin within the SO.

The AE provides continuous verification of hardware, firmware, OS, and workload from initialisation and through operation. The verification is based on the principle to build a secure chain of trust based on integrity measurements starting from the hardware root of trust which is a BMC in a server. The BMC Root of Trust provides the initial assurance that the platform starts in a known-good state. It is then followed by firmware integrity measurement leveraging the Measured Boot feature of UEFI and the bootloaders (e.g. Shim, Grub2). Each component, including the Linux kernel, is measured to create a chain of measurements, which are recorded in tamper proof storage, provided by the TPM. The measurements extension in the TPM provides a way to authenticate the chain of measurements when the AE verifies the state of the server. There are malwares that are known to disable security services such as secure boot to maintain persistence in the compromised nodes. A malware can extend new measurements in the TPM but cannot remove its measurement. Since each measurement is recorded and verified, such attacks can be detected by AE during platform initialisation.

The Linux kernel provides the IMA module in order to extend the chain of trust after the boot phase, up to the application layer. IMA implements the Measured Boot principles by making measurements on the dynamic executable content (applications and kernel modules) and the configuration files loaded at runtime; this allows the AE not only to verify that the platform booted in a trusted way, but also that all the applications and kernel modules loaded at runtime are trusted. The IMA module stores the sequence of the measurement events in a measurement list maintained in the kernel memory.

The AE Agent sends the IMA measurement log to the AE, together with the measurement aggregates signed by the TPM, at each attestation cycle. This allows the AE to determine the integrity level of each SCHI node at runtime; for example, if the AE detects a software component on a node that is not present in the whitelist for that node, or if the measurement of a component does not match its expected reference value provided by the SO, then the AE will mark the node as untrusted and will notify the RS, which is in charge to select the appropriate remediation actions that will be enforced by the SO (e.g. ring-fencing the compromised node, removing it from the cluster, applying security patches).

Our goal is to make the AE capable of attesting containers individually in order to identify any compromised SC; this allows to stop only the untrusted SC and to replace it with a new instance of an equivalent SC without the need to restart the entire platform, thus avoiding the disruption of the security service provided by uncompromised SCs. Containers are processes running on the host system: this means that the measurement events they generate are detected by the IMA module in the same way as those triggered by non-containerised processes and are stored all together in the same measurement log. This implies that, to attest containers individually, the AE should be able to determine if a given measurement event of the IMA measurement log belongs to a container or to the host system and, in the former, to which specific container it is associated. However, the built-in templates provided by the IMA module do not contain fields that allow to make this kind of distinction.

To overcome this issue, we defined and integrated in the kernel of the SCHI nodes a new IMA template that enables the attestation of individual containers; this template, in addition to the fields provided by the default template (i.e. the digest of the file content and the file path-name), provides

other metadata that take into account some properties of the process triggering the measurement event, in particular its control groups and the list of its dependencies:

- the list of dependencies field allows the AE to establish with certainty whether an entry of the measurement log was generated by a container or by the host system; and this is because, if the measurement event is related to a container, then the shim process that manages the execution of the container is always present in the dependencies field, otherwise the AE would attribute the measurement event to the host system;
- if the entry belongs to a container, the control group field allows the AE to attribute the measurement to a specific container, and therefore to a particular SC, the measure has to be attributed; in fact, if the process that generates the measurement event runs in a container, this field contains the full-identifier of that container, allowing the AE to associate the measurements with the container.

In this way, the AE, by checking the IMA measurement log, can continuously monitor all the executable code and configuration files that are loaded into a specific SC at runtime and promptly react as soon as a measurement mismatch is detected, informing the RS on the integrity failure related to the SC.

Testbed: To test the functionality and performance of the AE, two test benches were created: one to evaluate the attestation capability of hardware, firmware, kernel runtime, and host system (i.e. the PALANTIR infrastructure), and a second one to test the integration of all the components of the PALANTIR project used in this work to evaluate the SC attestation capability. The first testbed is set up to evaluate the AE with one monitored node. Table 1 presents the testing environment used for the evaluation of the AE and AE agent in the infrastructure attestation use case.

Table 1

Resources and environment in-use for testing AE's infrastructure attestation capability

Attestation Engine	Attestation Engine Agent
• 4 CPUs, • 8GB of RAM, • 60 GB of HDD • OpenSUSE Leap 15.3 • HPE AE package for openSUSE_Leap-15.3	• bare-metal node • TPM 2.0 chip • HPE's iLO • OpenSUSE Leap 15.3 • IMA and SELINUX configured

Table 2

Resources and environment in use for testing AE's SCs attestation capability

Attestation Engine (VM)	SCHI node (bare metal)
• 2 CPUs • RAM: 8 GB of RAM • 106 GB SSD • Ubuntu Server 20.04.3 LTS • Custom Keylime framework based on v. 6.0.0 • Docker CE v. 20.10.12 • Docker Compose v. 1.29.2	• 40 vCPUs (20 cores, 2 threads/core) • 128 GB of RAM • 500 GB SSD • Ubuntu Server 20.04.3 LTS • Custom Linux based on v. 5.13.19 • TPM 2.0 chip • Custom Keylime framework v. 6.0.0 • Docker CE v. 20.10.12

The second testbed is configured to monitor SCs deployed on a single physical node, which is one of the worker nodes in the SO-managed Kubernetes cluster. The characteristics of the nodes used to test the AE's SCs attestation capability are listed in Table 2.

The AE has been configured acknowledging a real operational environment of having fixed period of attestation cycle. The attestation cycle for this evaluation is set to 10 minutes. To evaluate the infrastructure OS verification by the AE, a bash script has been designed to execute an attack on the monitored node on a random time (in the 2 to 8 minutes interval) between an attestation cycle. The host is forced to reboot each time the attack is detected to revert the attack. A minimum wait of 2 minutes is enforced before the next attack is executed to ensure that an initial attestation has been performed after

the reboot. When the attack is detected immediately in the next attestation cycle and the attestation result is logged to measure the mean time of detection for the compromised node. For the infrastructure node attestation, a script is executed to make changes in hardware and firmware. In order for these changes to take effect, a reboot of the system is required. Once the machine is up, an attestation is performed on the monitored node which detects any sign of compromise in the machine. Finally, in order to evaluate SC attestation, a malicious insider actor compromising a SC has been simulated; as soon as the malware injected in the SC is executed, the attack is detected at the next attestation cycle.

Evaluation: The performance evaluation of the AE focused on the time taken by the AE to detect a compromise. The test scenarios for AE have been designed according to the attestation capabilities of the AE, extensively covering the system stack of SCHI to detect integrity fault. An attack has been simulated to trigger each attestation capability and the mean time to detect compromise was evaluated on the average from about 100 to 300 attestations performed for each set of experiments, in order to get statistically meaningful results. As for the SC attestation capability, the values have been acquired as the number of SCs instantiated on the node increases, starting from 1 SC up to a deployment scenario of 32 SCs instances.

- **Hardware attestation:** A hardware-tampering attack has been simulated by changing the reference value of the hardware and rebooting the platform for the hardware change to take place. When the system boots up, an initial attestation is requested from the node which triggers a change in reference value and the attack is detected.
- **Firmware attestation:** An attack scenario is simulated by disabling the Secure boot configuration in a platform and rebooting the platform for the firmware change to take place. The AE detects the change in the UEFI configuration when the system boots up and performs initial attestation on the node.
- **OS Verification:** An attack scenario is depicted where a new binary file is added to the system by a malicious actor which will create a new IMA measurement. As the node is being attested periodically with an attestation cycle of 10 mins, the attack is detected by the AE immediately in the next attestation cycle.
- **Runtime Attestation with DIME:** An attack scenario is simulated by injecting a kernel module in the platform. Since HPE DIME continuously monitors and verifies portions of OS kernel using a scanning engine, the attack is detected instantly with an average detection time of 10 sec.
- **SC Attestation**: Two attack scenarios have been simulated: (i) a malicious modification to a legitimate executable present in the image of a container belonging to a SC, (ii) the injection of an unauthorised binary inside a container of a SC. Both types of attacks are immediately detected at the next attestation cycle and the overall time taken by the AE to assess the integrity of each SC running on the node has been logged to estimate the average time to detect a compromised SC. The experiment has been repeated as the load on the SCHI node increases, in order to evaluate whether the number of deployed SCs impacts AE performance.

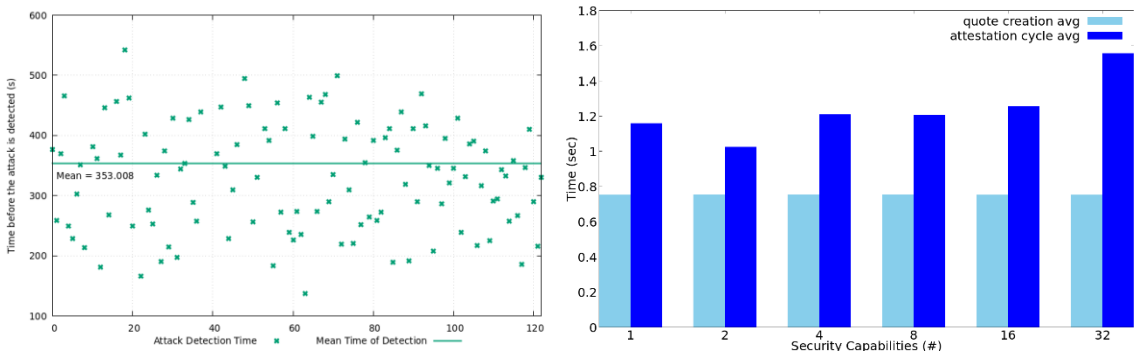


Figure 2 (left) and 3 (right): Mean time of detection of compromised node (left) and attestation latency as the number of SCs increases (right)

Figure 2 shows experiments results for OS verification capability. A total of 123 attestation cycle was performed to evaluate the mean time of detection for a compromised node by AE. A script was designed to execute an attack randomly between 2-8 minutes in each attestation cycle. Figure 2 shows that the mean time of detection is 353 seconds in an attestation cycle of 10 minutes. This time for detection is satisfactory as a threat report [24] from CrowdStrike reports that an average of 18 minutes is required for state sponsored attackers to infiltrate a company's network after gaining initial foothold.

Figure 3 shows the time required for the AE to complete the attestation process; the values reported in the histogram have been obtained on the average of 300 attestations performed for each group of SCs analysed. The different time contributions to the attestation cycle comprise the time for sending an attestation request over the network, the time *t* that the TPM takes to create a quote (i.e. the evidence provided by the TPM on the integrity status of the platform, reliable even when the platform has been compromised), the time needed for the Attestation Agent on the monitored node to read the IMA measurement log and send it over the network, the time the AE takes to authenticate the attestation report and evaluate the trustworthiness of the SCs. The results reported in the histogram show that the time required to complete a SC attestation cycle (dark blue bar) remains almost constant, at 1.2 seconds, up to 16 SCs deployed and reaches 1.6 seconds for 32 SCs deployed. This means that the attestation time grows very slowly as the number of SCs deployed on a node increases, making our AE highly scalable. Furthermore, a preponderant part of the overall attestation time is occupied by the TPM to create the quote (light blue bar); yet, this time cannot be optimised by the AE because it depends on the TPM's internal implementation of the asymmetric algorithm that creates the quote signature, as well as on the implementation of the Software Stack used by the Attestation Agent to send the *TPM2_quote* command to the TPM (tpm2-tss v. 3.2.0 in the second testbed). From the data analysis it follows that the SCs attestation process is highly scalable, given that the latency of the AE remains low and not significantly affected by the increase of the number of SCs.

4.2. Remediation decision-making process

Implementation: The **RS** implementation is designed with a Finite State Machine (FSM) framework build upon the Spring state machine library. An FSM is a computing model that mimics sequential logic through a set of inputs and the implementation of a finite number of states and transitions that describe and regulate execution flows. RS enacts recovery policies when attestation faults are detected to direct automated mitigations, as well as interfacing with a front-end for action needing human intervention. RS executes its predefined FSM as a recovery policy to respond with necessary actions. One of the RS recovery policies in PALANTIR is the re-instantiation of the failed SC by calling the SO endpoints to stop and to re-instantiate the failed SC. The execution of the RS recovery policy is initiated once the AE sends the information about the failed attestation of the SC. In this recovery policy, RS is also capable to notify the user about the executed actions.

With the implementation of this evaluation, we measured the average time of the RS component applying the recovery policy. The flow was as follows: first, RS received the JSON input where an instance of SC was marked as a failed. This triggered the flow where RS called the REST endpoint of the SO to stop the failed service. Once the service was stopped, a notification was sent from RS over a Kafka topic to inform the Portal about the performed action. After that, the RS called another endpoint from the SO to create a fresh instance of the failed SC. Once this was done, RS sent again the notification message to the Portal over the Kafka topic, which concludes the flow. We measured the total running time (from the beginning to end of the flow) and the average time for one request, as well as the efficiency of the message retrieval.

Testbed: To evaluate the RS we prepared a Python script to measure the RS execution times. We used the ORION testbed, the original environment used in the PALANTIR project. SFERA hosts an on-premises VM that is connected to the testbed over the Kafka cluster from ORION. Table 3 describes the environment of the VM, and frameworks/libraries used in the deployment and testing of the RS.

The VM that hosts the RS system is built with Spring Statemachine, and it has the Eclipse Papyrus tool installed. Eclipse Papyrus can also be installed and used to deploy policies outside the test environment, since it does not affect the RS VM if it is not running. To provide the real-time execution, the RS VM runs new instances on each incoming request based on the detected threats. Once the

execution of the RS policy has reached to the final state, the instance is stopped. The FBM contains the Kafka and REST API endpoints to communicate with other systems and components.

Table 3
Environment and framework/libraries used in testing environment

VM Environment	Framework/Libraries
• Xubuntu 20.04 LTS VM • 4 GB RAM • 1 CPU (4 Cores) • 32 GB SSD • 1 Gbps internet connection	• Spring Boot 2.3.0 • Spring Statemachine 3.0.1 • Eclipse Papyrus 2020-06 (4.16) • Apache Camel 3.9.0

Evaluation: For the evaluation we used multiple threads in the form of Multiple-Input and Multiple-Output (MIMO) method. With this evaluation we are testing the platform's capability to scale up and proceed with concurrent remediation action on several nodes simultaneously. Results are presented in Figure 4. We did not experience difficulties while using threads, and messages delivery was successful since RS was able to handle the speed of incoming requests. In case of difficulties, a solution could be to implement multiple RS instances and split the requests. A remediation request basically represents the execution of one remediation policy. The time needed for the RS to finish with one remediation request is 10,13 seconds. For the load test we used 10 threads, which would represent 10 users sending remediation requests at the same time (simultaneously). With those 10 threads we tested with 1, 2, 3, 4 and 5 remediation requests and observed the times. To constitute the dataset, we conducted 4 measurement iterations. Those are the results for this research project; however, when PALANTIR is used in industrial settings, we anticipate being able to handle higher loads by scaling up the component's instances in response to demand. Results show that the total time increases linearly from 50,91 seconds for 10 threads and 1 request to 265,35 seconds for 10 threads and 5 requests. The average time for one request, or the average time for one remediation execution to finish is under 60 seconds for each testing case.

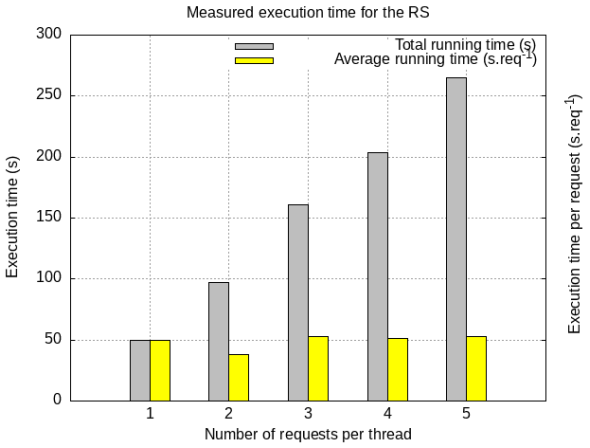


Figure 4: Evaluation times of the RS component

4.3. Security orchestration for the decision enforcement

Implementation: The **SO** oversees operations that are especially relevant to ZTA.

First, and to provide data that contributes to the assessment of the trustworthiness of every new registered node (typically, a running SC instance), the attestation and monitoring modules internal to SO extract and exposes data coming from the SCHI to other components in the architecture. Specifically, data is recovered from (i) the container runtime and (ii) the running SC instances themselves. The internal attestation module provides runtime details, which are passed to the AE during

any new SC onboarding and deployment – where the AE uses a subset (i.e. container ID, image ID and IP) to access the nodes and images subject to the integrity attestation. The internal monitoring module registers, persists and exposes generic (e.g. resource consumption) and/or custom metrics (subset of UNIX-like commands) requested on specific running SC instances.

Secondly, the SO interfaces indirectly (via OSM) or directly with the SCs and SCHI, respectively, using encrypted TLS channels with mutual authentication. These interfaces are named after the reference points laid out in the ETSI NFV architecture. The first one is named "Ve-Vnfm-em" and interconnects OSM with the VNF Manager (VNFM), and the VNFM to the Element Manager (EM), which is ultimately in charge of passing the configuration actions that are part of the decision enforcement. The VNFM is implemented by a third-party software (Canonical's Juju), where the Juju controller interacts with the Juju units (or applications being deployed) over TLS-encrypted websockets. The last one is the Or-Vi interface, where the SO accesses the Kubernetes cluster that is part of the SCHI in the canon, secured way, i.e., through the usage of a kubeconfig file with appropriate credentials (based on X509 certificates).

Testbed: The ORION testbed used in the PALANTIR project was used also here. This evaluation procedure considers the two nodes (VMs) allocated for OSM and SO as part of the control plane, where each of them deploys their specific features across interconnected containers. On the other hand, the Kubernetes cluster consists of one master and three worker nodes (two of them VMs and one a dedicated server with TPM2). Details are provided in Table 4.

Table 4
Resources and environment in use for the evaluation of SO and OSM

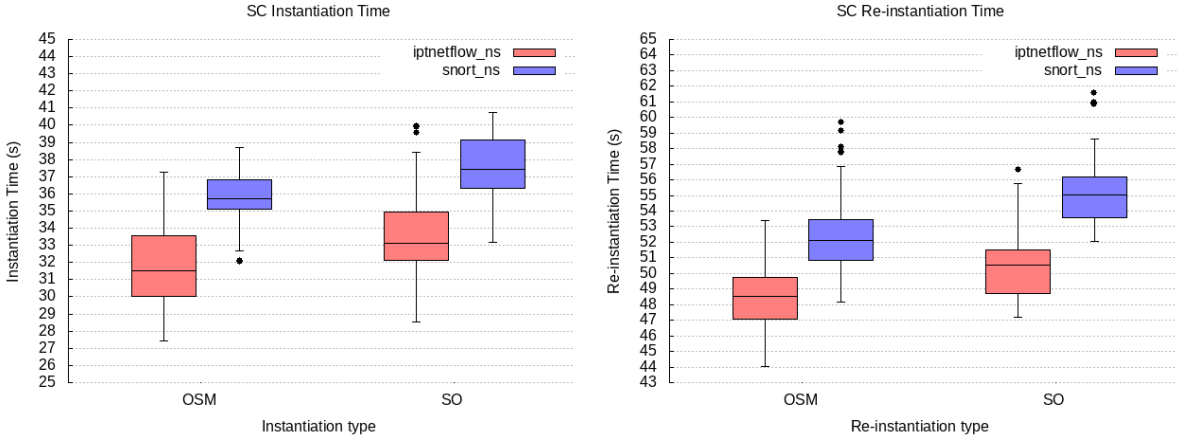
SO (VM)	Main frameworks and libraries	Kubernetes details
• Ubuntu 18.04.6 LTS • 2 cores CPU, 8 GB RAM, 20 GB SSD	• fastapi 0.73.0 • uvicorn 0.17.1 • flask 2.0.2 • kafka-python 2.0.2 • mongoengine 0.23.1	• Kubectl 1.23.4 • CRI: Docker 20.10.12 • CNI: Flannel 0.3.1 • OpenEBS 3.1.0
SO (Containers)	Kubernetes cluster (VMs / server)	
• Debian 11 • Python 3.8.12 • pip 22.0.4	• Ubuntu 20.04.4 LTS • Master: 4 cores CPU, 8 GB RAM, 40 GB SSD • Workers (x2): 12 cores CPU, 20 GB RAM, 100 GB SSD	• Ubuntu 20.04.3 LTS • Worker: 20 cores CPU, 128 GB RAM, 400 GB SSD • TPM 2.0
Connectivity		
• 1 Gbps Internet connection		

Typically, all Kubernetes nodes, except one (the dedicated server with TPM 2.0 to support the integrity measurement of the virtual containers running on the Docker container runtime) are tainted so to not perform scheduling. However, these tests target the performance evaluation for typical SO operations within the two control plane nodes. Since the attestation procedure does not affect the scheduling in the worker nodes, all nodes were used in the end, disregarding their role.

Evaluation: Given the intermediate position of the SO in the orchestration pipeline, this evaluation aims at identifying the extra time incurred by this component compared to the bare usage of OSM (the MANO leveraged by SO). The three possibilities that can be leveraged during the mitigation and decision enforcement procedures are taken as metrics and measured, i.e. (i) instantiation, (ii) re-instantiation and (iii) configuration times. The first and third metric cover the steps that can be (all together or separately) incurred during each mitigation process, where a new SC instance is instantiated in the adequate network segment to protect and/or where the configuration of a running SC is necessary. The second metric focuses on measuring the time required for the outcome of a failed integrity

assessment, where re-instantiation occurs when a compromised, running SC instance is terminated and, instead, a new one of the same type is re-instantiated.

Three Python scripts measure each of the three metrics for the SO, and another three scripts act as counterpart for OSM. Each script iterates the operation defined by the metric N (50) times over each type of SC. This is done sequentially and in batches of R (5-15) requests, reverting that same operation at the end of the measurement of that request and waiting for S (25-60) seconds before moving to the next request. The reverting and waiting process is enforced to minimise the risk of leftovers that can otherwise significantly increase the measured times, due to constraints on disk space and hitting timeouts. Time is measured right between submitting the action(s) and receiving the success confirmation that corresponds to each request. The resulting times per metric, comparing the two instantiation "types" (SO vs OSM) are depicted in Figures 5, 6 and 7. It is worth noting that each request to any of these components is (i) enacted sequentially (not concurrently with any other pending operation); (ii) independent from any other (disregarding order and relations); and (iii) idempotent (leaving the environment in the last clean state after every request). These tenets have facilitated measuring in the optimal conditions for the chosen testbed.



Figures 5 (left) and 6 (right): Distribution of the instantiation (left) and re-instantiation (right) times across SCs between SO and OSM

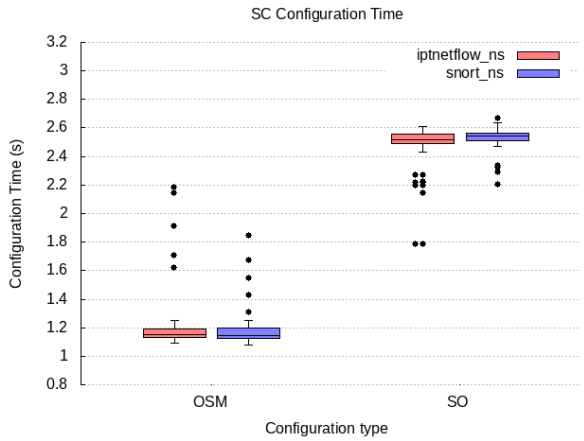
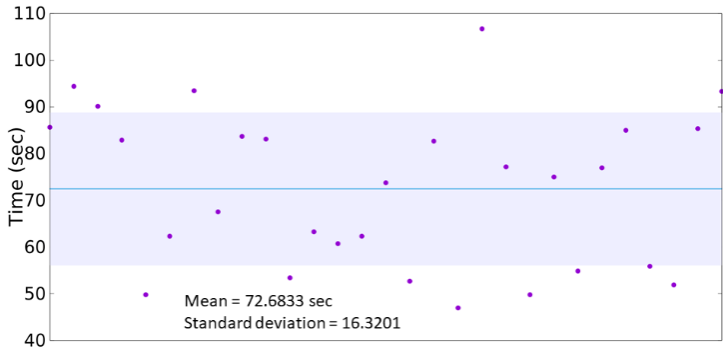


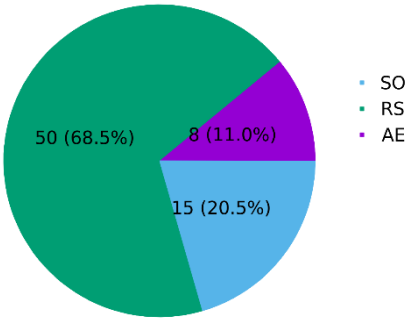
Figure 7: Distribution of the configuration times across SCs between SO and OSM

4.4. PALANTIR Zero-Trust Architecture evaluation

This subsection reports the results obtained in the evaluation of the overall reaction time of the PALANTIR framework to force a remediation action as soon as a compromise to a SC is detected. To estimate this time, a Bash script was created to simulate an attacker which compromises a SC by maliciously modifying one of the image's binaries and then executing it. Time was acquired from the

moment the attack is executed to the moment the compromised SC is terminated by the SO. Figure 8 shows the results of 30 measurements performed using *snort_ns* as SC and the average value obtained, equal to about 73 seconds, while Figure 9 shows how the time is distributed among the various components. During the experiments, the AE was configured to perform an attestation cycle every 2 seconds and to send notifications to the RS every 10 seconds; and the RS was subjected to a workload of 1 remediation request at a time. From the data it emerges that the ZTA architecture of the PALANTIR framework guarantees that, in less than 2 minutes, an attack aimed at compromising the integrity of the SC is detected by AE, processed by RS by sending a request to remove the SC untrusted and resolved by SO forcing its removal from SCHI.





Figures 8 (left) and 9 (right): average reaction time of PALANTIR ZTA (left) and distribution of average times (in seconds) between the involved components (right)

5. Discussion

Table 5
Comparison between ZTA tenets and PALANTIR framework implementation

#	ZT tenet	Is it met in PALANTIR?	Justification
1	All data sources and computing services are considered resources.	✓	All security capabilities deployed in the SCHI are resources of the PALANTIR ZTA.
2	All communication is secured regardless of network location.	✓	All communication between the PALANTIR components in the control plane and the SCs and SCHI, is secured.
3	Access to individual enterprise resources is granted on a per-session basis.	≈	Access request is granted on a per-session basis for most of individual PALANTIR resources.
4	Access to resources is determined by dynamic policy—including the observable state of client identity, application/service, and the requesting asset—and may include other behavioural and environmental attributes.	✓	Access to PALANTIR resources depends on dynamic policies since, when the security posture of a resource get compromised, it is immediately isolated and remediated by the actions enforced by the RS and the SO.
5	The enterprise monitors and measures the integrity and security posture of all owned and associated assets.	✓	The integrity and security posture of all PALANTIR resources is continuously monitored through the AE.
6	All resource authentication and authorisation are dynamic and strictly enforced before access is allowed.	≈	Access to most of PALANTIR resources is granted with dynamic policies for authentication and authorisation.
7	The enterprise collects as much information as possible about the current state of assets, network infrastructure and communications and uses it to improve its security posture.	✓	Monitoring data coming from the AE are used to improve the security posture of resources through remediation actions enforced by RS and SO to fix detected security breaches.

ZT security model suggests that no request for access to a resource of the infrastructure must be considered trusted a-priori and that the integrity of each infrastructure component must be constantly monitored and evaluated. While architecting PALANTIR, the authors thrive to comply to this paradigm, which replaced the traditional one based on the network perimeter, no longer applicable to modern virtualised distributed infrastructures. NIST defined a ZT security model through a set of seven basic tenets, conceived as ideal objectives that should be realised in a ZTA, fully or even partially depending on the strategy adopted in each context. We examined the PALANTIR architecture considering such tenets to evaluate which of them are met into the infrastructure and to what extent. Table 1 exposes how PALANTIR platform complies with these principles when managing security capabilities according to their security posture.

6. Conclusion

This paper proposed, prototyped, and evaluated the PALANTIR ZT architecture, justified it with an MSSP threat model, and analysed its compliance with the ZTA paradigm, highlighting how the basic ZTA tenets are satisfied in the PALANTIR framework. An extensive range of experiments was conducted in order to demonstrate the effectiveness of PALANTIR's ZTA solution and the feasibility of its adoption in real operational scenarios for the protection of SMEs and ME. Even though PALANTIR shows remarkable progress in terms of applying ZTA principles to the SecaaS model, further improvements can be investigated and applied. As future work, the ZTA security model should be considered on different perspectives, not only tied to the PALANTIR infrastructure itself, and extended by adopting some of the presented solutions directly on the customer infrastructure (e.g. remote attestation).

Acknowledgement

The work described in this article has received funding by the European Union Horizon 2020 research and innovation programme, supported under Grant Agreement no. 883335. Part of this work is also supported by the Spanish Government Grant ONOFRE-3 PID2020-112675RB-C43 funded by MCIN/ AEI /10.13039/501100011033. The content of this article does not reflect the official opinion of the European Union or any other institution. Responsibility for the information and views expressed therein lies entirely with the authors.

References

[1] E. Mantas et al., "Practical Autonomous Cyberhealth for resilient Micro, Small and Medium-sized Enterprises," in 2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom), Sep. 2021, pp. 500–505. doi: 10.1109/MeditCom49071.2021.9647609.

[2] Ravie Lakshmanan, "Government Agencies Warn of Increase in Cyberattacks Targeting MSPs," The Hacker News, May 11, 2022. https://thehackernews.com/2022/05/government-agencies-warned-of-increase.html.

[3] ETSI, "ETSI GS NFV-SEC 001: Network Functions Virtualisation (NFV); NFV Security; Problem Statement," ETSI, ETSI GS NFV-SEC 001 V1.1.1, Oct. 2014. [Online]. Available: https://www.etsi.org/deliver/etsi_gs/NFV-SEC/001_099/001/01.01.01_60/gs_NFV-SEC001v010101p.pdf.

[4] A. Darabseh, M. Al-Ayyoub, Y. Jararweh, E. Benkhelifa, M. Vouk, and A. Rindos, "SDSecurity: A Software Defined Security Experimental Framework," in 2015 IEEE International Conference on Communication Workshop (ICCW), Jun. 2015, pp. 1871–1876. doi: 10.1109/ICCW.2015.7247453.

[5] M. Compastié, R. Badonnel, O. Festor, and R. He, "From virtualisation security issues to cloud protection opportunities: An in-depth analysis of system virtualisation models," Computers & Security, vol. 97, p. 101905, Oct. 2020, doi: 10.1016/j.cose.2020.101905.

[6] A. Baumann, M. Peinado, and G. Hunt, "Shielding Applications from an Untrusted Cloud with Haven," ACM Trans. Comput. Syst., vol. 33, no. 3, p. 8:1-8:26, Aug. 2015, doi: 10.1145/2799647.

[7] "Trusted Platform Module Library Specification, Family "2.0"", Trusted Computing Group. https://trustedcomputinggroup.org/work-groups/trusted-platform-module/.

[8] "TPM 2.0 Keys for Device Identity and Attestation", Trusted Computing Group. https://trustedcomputinggroup.org/resource/tpm-2-0-keys-for-device-identity-and-attestation/.

[9] "TCG Trusted Attestation Protocol Information Model", Trusted Computing Group. https://trustedcomputinggroup.org/resource/tcg-tap-information-model/.

[10] T. Lodderstedt, D. Basin, and J. Doser, “SecureUML: A UML-Based Modeling Language for Model-Driven Security,” in ≪UML≫ 2002 — The Unified Modeling Language, Berlin, Heidelberg, 2002, pp. 426–441. doi: 10.1007/3-540-45800-X_33.

[11] D. Basin, J. Doser, and T. Lodderstedt, “Model driven security for process-oriented systems,” in Proceedings of the eighth ACM symposium on Access control models and technologies, New York, NY, USA, Jun. 2003, pp. 100–109. doi: 10.1145/775412.775425.

[12] J. Cappos, J. Samuel, S. Baker, and J. H. Hartman, “A Look in the Mirror: Attacks on Package Managers,” in Proceedings of the 15th ACM Conference on Computer and Communications Security, New York, NY, USA, 2008, pp. 565–574. doi: 10.1145/1455770.1455841.

[13] M. Compastié, R. Badonnel, O. Festor, and R. He, “A TOSCA-Oriented Software-Defined Security Approach for Unikernel-Based Protected Clouds,” in 2019 IEEE Conference on Network Softwarization (NetSoft), Jun. 2019, pp. 151–159. doi: 10.1109/NETSOFT.2019.8806623.

[14] European Union Agency for Cybersecurity., ENISA threat landscape for supply chain attacks. LU: Publications Office, 2021. Accessed: May 05, 2022. [Online]. Available: https://data.europa.eu/doi/10.2824/168593

[15] C. Buck, C. Olenberger, A. Schweizer, F. Völter, and T. Eymann, “Never trust, always verify: A multivocal literature review on current knowledge and research gaps of zero-trust,” Computers & Security, vol. 110, p. 102436, Nov. 2021, doi: 10.1016/j.cose.2021.102436.

[16] J. H. Saltzer and M. D. Schroeder, “The protection of information in computer systems,” Proceedings of the IEEE, vol. 63, no. 9, pp. 1278–1308, Sep. 1975, doi: 10.1109/PROC.1975.9939.

[17] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, “Zero Trust Architecture,” National Institute of Standards and Technology, Aug. 2020. doi: 10.6028/NIST.SP.800-207.

[18] B. Chen et al., “A Security Awareness and Protection System for 5G Smart Healthcare Based on Zero-Trust Architecture,” IEEE Internet of Things Journal, vol. 8, no. 13, pp. 10248–10263, Jul. 2021, doi: 10.1109/JIOT.2020.3041042.

[19] S. A. Siddiqui, A. Mahmood, Q. Z. Sheng, H. Suzuki, and W. Ni, "A Survey of Trust Management in the Internet of Vehicles", Electronics 10, no. 18: 2223, Sep. 2021, doi: 10.3390/electronics10182223.

[20] R. Vanickis, P. Jacob, S. Dehghanzadeh, and B. Lee, “Access Control Policy Enforcement for Zero-Trust-Networking,” in 2018 29th Irish Signals and Systems Conference (ISSC), Jun. 2018, pp. 1–6. doi: 10.1109/ISSC.2018.8585365.

[21] S. Mehraj and M. T. Banday, “Establishing a Zero Trust Strategy in Cloud Computing Environment,” in 2020 International Conference on Computer Communication and Informatics (ICCCI), Jan. 2020, pp. 1–6. doi: 10.1109/ICCCI48352.2020.9104214.

[22] “Security Protocol & Data Model (SPDM) Specification”, DMTF. https://www.dmtf.org/standards/SPDM

[23] “HPE Distributed Intrusion Monitoring Engine”. https://community.hpe.com/t5/Advancing-Life-Work/Quis-custodiet-ipsos-custodes-HPE-next-generation-intrusion/ba-p/7042089

[24]” CrowdStrike report: Russian hackers are the fastest ", https://www.tellerreport.com/tech/--crowdstrike-report--russian-hackers-are-the-fastest-.S1NSS8tBV.html