Designing a Secure IoT System Architecture from a Virtual Premise for a Collaborative AI Lab

Vida Ahmadi Mehri
Blekinge Institute of Technology
vida.ahmadi.mehri@bth.se

Dragos Ilie
Blekinge Institute of Technology
dragos.ilie@bth.se

Kurt Tutschku
Blekinge Institute of Technology
kurt.tutschku@bth.se

***Abstract*—IoT systems are increasingly composed out of flexible, programmable, virtualised, and arbitrarily chained IoT elements and services using portable code. Moreover, they might be sliced, i.e. allowing multiple logical IoT systems (network + application) to run on top of a shared physical network and compute infrastructure. However, implementing and designing particularly security mechanisms for such IoT systems is challenging since a) promising technologies are still maturing, and b) the relationships among the many requirements, technologies and components are difficult to model a-priori.**

The aim of the paper is to define design cues for the security architecture and mechanisms of future, virtualised, arbitrarily chained, and eventually sliced IoT systems. Hereby, our focus is laid on the authorisation and authentication of user and host, as well as on code integrity in these virtualised systems. The design cues are derived from the design and implementation of a secure virtual environment for distributed and collaborative AI system engineering using so called AI pipelines. The pipelines apply chained virtual elements and services and facilitate the slicing of the system. The virtual environment is denoted for short as the *virtual premise (VP)*. The use-case of the *VP* for AI design provides insight into the complex interactions in the architecture, leading us to believe that the *VP* concept can be generalised to the IoT systems mentioned above. In addition, the use-case permits to derive, implement, and test solutions. This paper describes the flexible architecture of the *VP* and the design and implementation of access and execution control in virtual and containerised environments.

I. Introduction

The Internet of Things (IoT) is evolving. Originally, it aimed at wirelessly interconnecting cyber-physical devices (sensors, actuators, wearables, vehicles, home appliances) with control applications [1]. However, complex enhancements of the IoT have been recently developed. IoT is increasingly viewed as a service- and Cloud-based system [2]. Such systems leverage, for example, the intermittent nature of wireless connectivity or provide for stateless information access using RESTful services. The concepts of Fog computing [3] and Multi-access Edge Computing (MEC) in public cellular networks[4] also complement the original IoT idea. Here, edge devices are located close to sensors and perform substantial amounts of computation, storage, communication (*e. g.*, high bandwidth or multi-homing for reliability). Furthermore, slicing concepts for IoT systems have been proposed [5]. They typically focus only on slicing techniques for the network segment and use SDN (Software Defined Networking) concepts to control the data forwarding.

Our hypothesis is that IoT systems accelerate even stronger towards flexible, virtualised, programmable, sliceable, and arbitrarily chained IoT elements and services using portable code. We substantiate this hypothesis by highlighting three technology trends: a) small but powerful single-board computers became popular, such as Rasberry PI [6] with 14 million units sold by 2017 [7]; b) various hypervisors and container environments became available for this class of devices [8], and c) component-based application development [9] and orchestration of containers [10] have gained traction in application and service provisioning.

While it becomes clear that future IoT systems pick up concepts and mechanisms from virtualisation, slicing and code portability, limited research has been conducted on the impact of those technologies on IoT security architectures and mechanisms. A recent analysis [11] shows that network security, data encryption, API security and authentication and PKI infrastructures are among the most required technologies.

Our approach is to focus on authentication and authorisation mechanisms in IoT systems that allow for virtualisation, slicing and code portability. However, many virtualisation and slicing technologies are still evolving. For example, Docker container security is increasingly addressed [12], [13], however not yet solved. This status impacts the engineering of ICT systems in general and of IoT systems in particular.

To overcome this dilemma, we applied a prototype-based approach for identifying the design cues to for security architectures and mechanisms in virtualised IoT systems. We start our investigation by analysing and comparing today's IoT architectures and draw parallels from the use-case of AI system engineering using a virtualised AI lab, denoted as secure *VP*. The lab permits the usage of a marketplace for AI artefacts and pipelines for component-based development. The use case is of particular challenge since it includes code mobility, multi-party collaboration including security, DRM, and privacy requirements for the aforementioned parts and edge devices. The *VP* aims at simplifying and orchestrating the security and security policy management. That means, the *VP* reliefs the system designer from addressing the security mechanisms and management, including security policy enforcement (*e. g.*, for DRM and data privacy). The system designer can focus on the AI application.

The paper is organised as follows: Sec. II describes our design concept, analyses and compares the various IoT architectures. Sec. III describes a component-based AI development

Workshop on Decentralized IoT Systems and Security (DISS) 2019
24 February 2019, San Diego, CA, USA
ISBN 1-891562-56-8
https://dx.doi.org/10.14722/diss.2019.23006
www.ndss-symposium.org

concept, its requirements and the suggested architecture for the *VP*. Sec. IV focuses on the authentication and authorisation of users, hosts, and code in virtual labs using code mobility. Sec. V discusses the lesson leaned and generalisation of the mechanism, incl. their vulnerabilities. Sec. VI summarises our finding and provides a brief outlook.

II. Towards Future Secure, distributed and flexible IoT system Architecture

A. Design Concept

Engineering security is a multidisciplinary endeavour [14]. Designing ICT architectures, and IoT systems, in particular, is typically a rigorous and disciplined system engineering task [15]. However, it is also acknowledged in [15] that there is an increasing disagreement between theory and practice in system design. This gap is mainly accounted to the huge push for fast system integration, *e. g.,* by market needs. We additionally attribute it to the lack of means for modelling and validating techniques for large-scale systems using pre-maturing technologies. To overcome this dilemma, [15] establishes the vision of autonomous IoT systems that can cope with the complexity through tighter integration of computing elements and compensating the reduced human intervention by enabling self-adaptation. While we believe that self-management mechanisms are of future importance, fully autonomous security control is hard to establish currently. Therefore, we focus on adaptivity in security, *c.f.,* Sec. II-B for a definition, as a first approach.

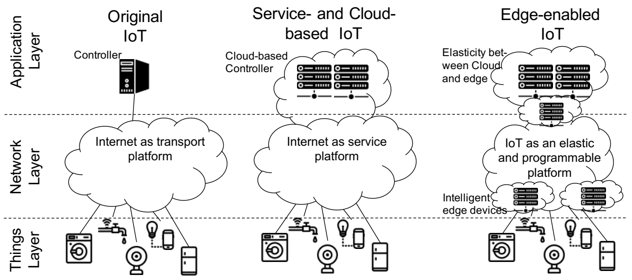
Requirements versus Architecture: Our scientific approach is not to design a fully flexible IoT architecture from scratch. We derive our design cues for security architecture from the interaction of security requirements and system architecture. In order to have a valid body of requirements and possible solutions, we consider the challenging use-case of collaborative AI system design.

From a distance, requirements specify what a system must be capable and the architecture describes how a system will be organised and behave such that it will fulfil these requirements. Hence, requirement engineering precede typically architecture design. However, when using immature technologies, architectural design choices easily influence these techniques. Therefore, we start our approach to security design by considering the IoT architectures first, *c.f.,* Sec. II-B and then define requirements for security mechanisms, in particular, authentication, described in Sec. III.

B. Comparison of IoT Architectures

Fig. 1 outlines the three major IoT architectures introduced in Sec. I. In general, IoT architectures consist of three different layers, namely: a) the *Things Layer*: sensors and actuators, b) the *Network Layer*: forwarding data or service routing (including access points and gateways), and c) the *Application Layer*: implementing data processing and control.

Table I shows a feature-based comparison of the architectures displayed in Fig. 1. The following features were used in the comparison: compute and storage location, service chaining, slicing, enabling collaboration, flexibility and adaptivity towards application workflows, scalability, and finally, security and trust.



Application Layer
Network Layer
Things Layer
Original IoT
Controller
Internet as transport platform
Service- and Cloud-based IoT
Cloud-based Controller
Internet as service platform
Edge-enabled IoT
Elasticity between Cloud and edge
IoT as an elastic and programmable platform
Intelligent edge devices

Fig. 1: Types of IoT architectures

The original IoT concept demonstrated impressively the capabilities of the applications and the scalability of the Internet as a transport platform. Service- and Cloud-based IoT enabled a larger range of applications by augmenting the original IoT with the elasticity and flexibility of Cloud and service computing. Edge-enabled IoT permits fine-grained resource elasticity between Cloud and edge and also the slicing of the network segment. All in all, over the years the IoT concept became more flexible in terms of resource usage. However, Table I also reveals that these IoT concepts still lack in terms of flexibility of security. Flexible security, of course, aims at high security but also at specialising and adapting the security to the application needs, *e. g.,* specific data privacy is enforced at participating Cloud and edge nodes. In addition to security, trust is needed for collaboration. Hence, mutual trust must be adaptively established between those entities who would like to cooperate on data or application.

III. Pipeline- and Component-based Development of AI Systems

The concept of a pipeline- and component-based system development was recently successfully applied for engineering data-driven AI systems [17]. Its main objective is to reduce design costs and time by re-using knowledge manifested in so-called *AI artefacts*. These artefacts are objects, code and functions used in AI development. Multiple artifacts can be chained to form an *AI engineering pipeline*. Artefacts and pipelines enable AI specialists to focus on AI functions and to exchange the results with another specialists. This approach may improve the quality of the AI solution by use of specialisation. Google has recently published a concept for a Cloud-based pipeline [18] and the H2020 project Bonseyes suggests a distributed one [19].

A. AI Pipeline

An excerpt of a Bonseyes AI training pipeline for Keyword Spotting (KWS) is depicted in Fig. 2, *c.f.,* [20]. The KWS applies a Deep Neural Network (DNN) model for identifying keywords in spoken sentences. This pipeline includes four steps: Data Sourcing, Data Preparation, Training, and Target Benchmarking. The steps are needed to obtain and pre-process data and to train and benchmark the DNN model. Each step is embedded in an artefact and instantiated as a Docker container. The output of an artefact is handed over as input to the succeeding one. Hence, the pipeline forms an *AI service function chain*. In Bonseyes, the pipeline is distributed, *i. e.,* Docker containers are executed on arbitrary hosts (including

TABLE I: Comparison of the different types of IoT architectures

Feature	Original IoT	Service- and Cloud-based IoT	Edge-enabled IoT	Virtual Premise (VP)
Compute location	at controller	in Cloud	elastically at Cloud / edge	elastically and programable at Cloud and edge
Data location	at controller	in Cloud and service proxies	elastically at Cloud / edge	elastically at Cloud / edge
Service chaining	not intented	in Cloud	in Cloud and edge	per pipeline in Cloud and edge
Slicing	not intented	in network (when using SDN)	in Cloud and network (when using SDN)	application overlay per pipeline
Collaboration	single application	in Cloud	in Cloud (not across edge)	per pipeline
Flex. / adaptive to app. workflow	static by design	in Cloud	elastic and programable	through programmability and orchestration
Scalability	by transport and processing capacity	by Cloud and service elasticity	by Cloud and edge elasticity	per pipeline through Cloud / edge elasticity
Security and trust	static by secured transmission, nodes and PKI	static by secured transmission, nodes, service API and PKI	predefined by architecture [16]	**flexible per pipeline through programmability, policies and orchestration**

datacenter or local developer workstations). The outcome of the training is an AI model that can be converted into code, which can even be deployed on IoT devices with adequate computational capabilities.

B. AI Marketplace

A core element of the Bonseyes project is the *AI Marketplace (MP)*, *c.f.*, Fig. 3., which enables the exchange of AI artefacts and the component-based AI design using artefacts. Developers can develop artefacts and upload them into the MP as containers. The containers are stored by the MP in the *Bonseyes' Private Docker repository (BPDR)*. In turn, AI specialists can search the MP for specific artefacts, download them from the BPDR, and place them into pipelines. In this way, the marketplace enables *collaborative development*, *i. e.*, multiparty cooperation in AI design.

C. Component-based Development Challenges

Bonseyes use of MP for AI artefact exchange imposes these specific *high-level challenges on security related functions*:

1) Protection of DRM (digital rights management): code and data are of value and thus it is reasonable that their owners want to protect and manage their digital rights.
2) Protection of privacy and data ownership: individuals may be willing to share private data for the purpose of AI training but may not be willing to waive data privacy and data ownership.
3) Compliance in collaboration: digital rights, privacy and data ownership need to be obeyed across various physical locations and legal entities when collaborating.
4) Fair enumeration: individuals, developers or companies need to be reimbursed when their digital intellectual property or data is used.

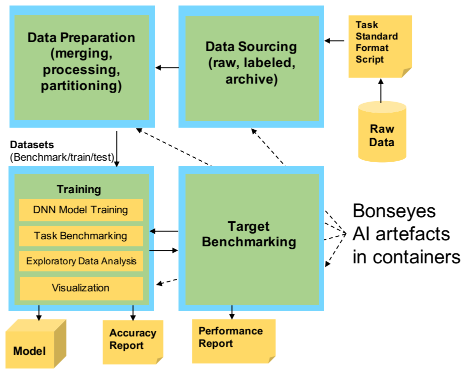

Data Preparation (merging, processing, partitioning)
Data Sourcing (raw, labeled, archive)
Task Standard Format Script
Raw Data
Datasets (Benchmark/train/test)
Training
DNN Model Training
Task Benchmarking
Exploratory Data Analysis
Visualization
Target Benchmarking
Bonseyes AI artefacts in containers
Model
Accuracy Report
Performance Report

Fig. 2: An AI data collection and training pipeline [20]

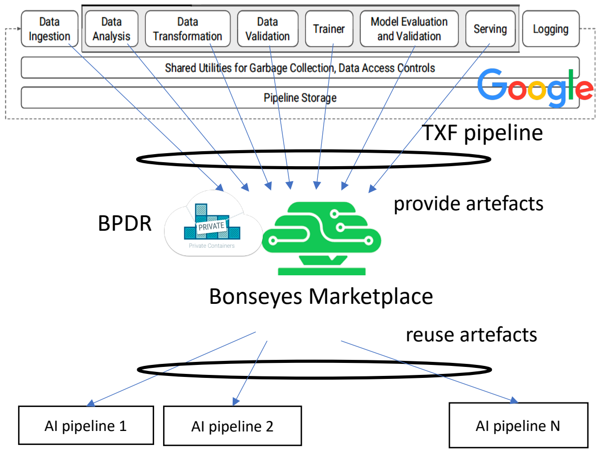


Fig. 3: Bonseyes AI marketplace, using a pipeline [18]

These high-level security challenges need to be translated into *technical security solutions*: An AI marketplace architecture is required, which enables authorised access, integrity and compliance verification for artefacts in a distributed environment, *i. e.*, across various physical locations. Moreover, the use of virtualisation technology for artefacts requires a) the compliance verification of the virtual environment at a host, as well as b) the validation of the artefact integrity, *i. e.*, that the Docker image carrying the artefact was not changed. Hence, authentication, authorisation, and policy enforcement are the core elements of our solution.

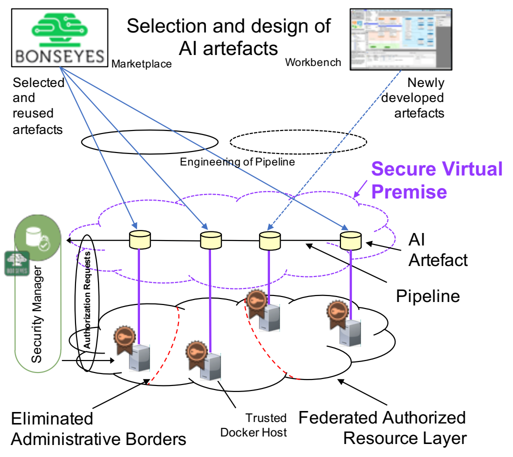

BONSEYES
Marketplace
Selection and design of AI artefacts
Workbench
Selected and reused artefacts
Newly developed artefacts
Engineering of Pipeline
Secure Virtual Premise
AI Artefact
Pipeline
Security Manager
Authorization Requests
Eliminated Administrative Borders
Trusted Docker Host
Federated Authorized Resource Layer

Fig. 4: Architecture of the Virtual Premise

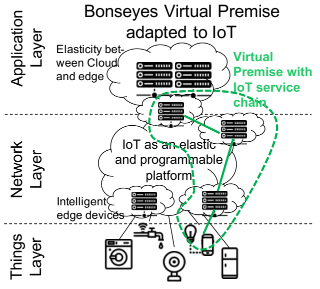

Application Layer
Network Layer
Things Layer
Bonseyes Virtual Premise adapted to IoT
Elasticity between Cloud and edge
Virtual Premise with IoT service chain
IoT as an elastic and programmable platform
Intelligent edge devices

Fig. 5: Virtual Premise as an IoT Architecture

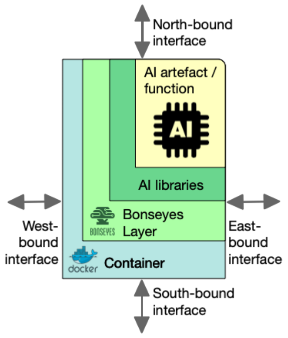


Fig. 6: A Bonseyes' Docker container comprising an AI artefact.

D. The Virtual Premise (VP)

Our solution for solving the technical challenges is to enhance the MP by a *secure VP*, *c.f.*, Fig. 4. The core functions of the VP are authentication, authorisation and policy enforcement. VP places the AI pipeline in the centre and builds a virtual boarder around it. The pipeline is instantiate on federated authorised resources. The pipeline elements/artefacts can be downloaded from the MP (or created and registered in MP using a developer's workbench). The VP uses network slicing and virtualisation techniques to provide isolation of pipelines, prevents unauthorised pipeline communication with the outside and unifies the security and access policies per pipeline. It allows workflows between authorised artefacts through secure programmable and polymorphic interfaces. The VP has similarities to an application overlay and builds a *virtual AI laboratory* per pipeline. The inside of the VP is a trusted area where artefacts comply with each other on DRM, privacy and collaboration policies, while the outside is untrusted. The artefacts, hosts and users of a pipeline must be authenticated by the *Security Manager (SM)* in order to join the VP. The policies of the VP are enforced locally on the Docker hosts participating in the VP. The split between security policy decision entities and enforcement points is discussed in *c.f.*, Sec. IV.

In terms of IoT architectures, the VP can be considered as an enhancement of the Edge-enabled IoT concept which focuses on slicing with respect to adaptive and flexible security policies, *c.f.*, Fig. 5. The IoT features of the VP are summarised and compared with other IoT architecture in the last column of Tab. I. As can be observed, the VP enables flexible security and trust per AI pipeline or IoT service chain.

E. Bonseyes AI Artefact Architecture

The second major component of the VP concept is the architecture of AI artefact, which is shown in Fig. 6. The artefact is implemented as a Docker container which is enhanced by a specific *Bonseyes Layer (BL)*. The BL serves four main purposes: a) to control communication with the artefact; b) to enable secure APIs to access AI functions and content; c) to facilitate artefact authentication with entities from the Bonseyes eco-system; and d) to verify the functional integrity of the underlying host platform, for maintaining the confidentiality and integrity of data processing and data exchange (*e. g.*, enforcing access control and DRM services). The BL is stsrted as soon as the container is executed, thus ensuring that all communication into the container is intercepted and secured. The secure APIs for the workflow between artefacts are referred to as the *east-bound interface* and the *west-bound interface*, respectively. The API for the execution of the AI function and towards the workbench is denoted as the *north-bound interface*, while the one towards the host (including virtualisation environment and filesystem access) is denoted *south-bound interface*. The AI function together with associated executable code is stored in component called *AI payload* and the required libraries for this function can be implemented as a layer in the Docker image.

IV. Authentication and Access Management

The MP offers an interface through which artefacts can be uploaded into or downloaded from the BPDR. In addition, the MP supports payment services for commercial artefacts. AI artefact usage can be conditioned by licensing terms, privacy requirements or local legislation. Furthermore, hosts available for artefact execution may belong to various administrative domains, each free to enforce different policies in terms of which artefacts and users are allowed to execute on their systems as well as how resource allocation should be performed. Therefore, access to hosts and artefacts is coordinated through the SM to enable compliance with this type of constraints through a distributed license management system.

In addition, the SM is the fundamental component that enables mutual authentication and functional integrity checks for the components belonging to a VP. This involves both the AI artefacts (*i. e.*, the containers and their contents) as well as the supporting infrastructure (*i. e.*, the server hosts executing the containers). The main concern in this case is that hosts or containers are tampered with in order to circumvent the license management system and bypass the usage constraints associated with an artefact (*e. g.*, using the artefact without a valid license). However, as past experience has shown, being able to counter *any* attack on the license management system, including those where malicious code is injected into the system, is a very difficult issue. Therefore, we make a set of

TABLE II: Security entities in Bonseyes

Actor	Roles
Bonseyes layer (BL)	Located inside the Docker image. Manages the authenticity of the artefact.
Bonseyes module (BM)	Located on the Docker host. Manages the authenticity of the host.
Bonseyes' Private Docker repository (BPDR)	Remote-accessible storage for Docker images containing artefacts.
Marketplace (MP)	Virtual meeting place for Bonseyes users. Enables artefact, host and user registration as well as artefact exchange with the aid of the SM.
Security Manager (SM)	Manages identities and crypto material and facilitates authentication and authorization. Contains a certification authority (CA). Trust anchor for the entire system.
User	AI developer creating and executing pipelines.
Virtual Premise (VP)	A set of Bonseyes registered hosts (each having a valid BM). Used for executing one or more pipelines.

reasonable simplifications, shown below, to relax the difficulty of the problem.

A. Assumptions for the Security Model

We assume that the SM is not compromised and that administrative domains are not malicious, do not collude with attackers and do not instrument the hosts to bypass the license system. Also, we assume that Docker images uploaded to BPDR are searched for malicious software before becoming available for download. Furthermore, we assume that the SM is equipped with a Certificate Authority (CA) that enables it to issue certificates for other entities in the MP. Essentially, the SM is the trust anchor for the system. Finally, the SM is equipped with a HTTPS server capable of certificate-based authentication.

B. Authentication and Authorization Protocol

Given the aforementioned assumptions, we have defined a protocol that verifies if a host is allowed to execute an artefact, and similarly if the artefact is allowed to execute on the given host. The actors involved in the protocol and their roles are listed in Table II. The protocol consists of a set of discrete phases listed below and shown in Fig. 7:

1) Artefact, host, and user registration
2) Host authentication
3) Image integrity check
4) Host (BM) authorisation for execution of artefact
5) Image execution (BL start)
6) Artefact (BL) authorisation for execution on host

In general, phases 1 and 2 can be performed at any time. However, given a specific artefact-host-user triplet, the other phases cannot execute correctly before phase 1 and 2 are completed successfully. Artefact registration entails that

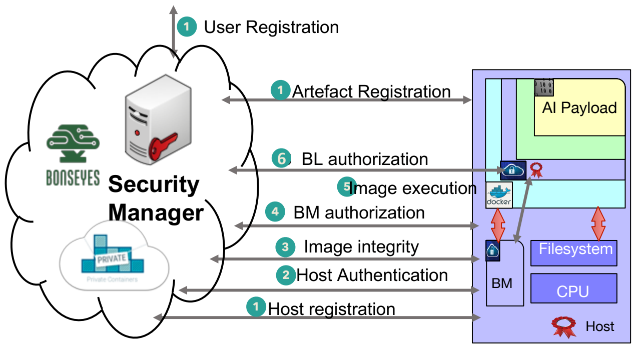


Fig. 7: Mutual authentication and authorization

an image is created and uploaded to the BPDR. Policies concerning the artefact execution can also be configured during this step. Then, the SM adds the BL layer to the image. It also inserts an artefact identifier (a human readable string) which is then signed with the SM private key yielding the *artefact ID*. This ID is shared by all instances of the same image and is used when the artefact states its type in phase 6. Finally, the SM's root certificate and any intermediate certificates required for signature verification are added to the trust database in the image. A message digest (*e. g.*, SHA-256) is computed over the final image yielding the *image ID*. The artefact ID and the image ID are stored as a pair in the SM database.

For hosts undergoing the registration process, the first step is to compute their *hardware ID* (*e. g.*, by reading specific entries in the SMBios). The hardware ID is signed with the host owner's private key, thus tying the identity of the owner to the host. The signed hardware ID is denoted as the *host ID*. If the host platform contains a trusted platform module (TPM), the hardware ID can be replaced with the Endorsement Key (EK) certificate for the TPM. The owner (typically the system administrator or the AI developer) is responsible for verifying that hosts are free from malware before taken in use. As part of the registration, the owner can also specify if there are restrictions in the use of the host (*e. g.*, if it is reserved for a specific developer, or restrained from executing artefacts from a specific vendor).

The hardware ID, the host ID and any usage policies are stored in the SM database. The SM generates a software component called Bonseyes Module (BM), which is installed on the host. The BM plays the same role for the host as the BL does for the artefact. Inside the BM there is an asymmetric key pair and the corresponding certificate, both generated by the SM specifically for this hardware ID. The SM's root certificate and any intermediate certificates are also present inside the BM.

Developers and host owners need also to register and obtain user IDs and user certificates (*e. g.*, in PKCS12/PFX) format). This is a one time process, and the ID is carried over to any pipeline or VP that the developer is working on. The certificates enable seamless client authentication (*e. g.*, from a web browser).

The hosts are configured to start the BM after the operating systems has booted up. This is where phase 2 begins. The BM will open a HTTPS connection to the SM and use the host ID (and EK certificate[1], if available) to authenticate itself with the SM. The SM will refuse to authenticate hosts that are missing from registration database or for whom the authentication fails for any other reason. If the authentication succeeds, the SM marks the host as available resource. Thanks to the cryptographic material installed in the previous steps, both client and server authentication can be performed when the HTTPS connection is established.

Phase 3 begins when a developer instructs the SM to assign the pipeline to a VP for execution. Each artefact can be pinned to a specific host in the VP or the SM can automatically assign them to hosts based on a predefined policy. The SM coordinates with the BMs in copying the artefacts to the hosts and in checking the integrity of the received Docker images using strong message digests (*e. g.*, SHA-256).

During phase 4, the BMs request from the SM policies surrounding the execution of the artefact under the controller of the developer. The requests are transported over HTTPS and each request contains the message digest of Docker image, the host ID, the user ID and a nonce. The message digest allows the SM to determine the identity of the artefact being executed, the host ID provides the identity of the host, and user ID denotes the developer executing the artefact. The purpose of the nonce is to help SM in constructing identifiers that can be used to differentiate between multiple instances of an artefact running on the same host. The requests are transported over HTTPS.

If the request is accepted, the SM constructs an entry consisting of five items: image digest, user ID, host ID, nonce and host IP address. This combination, *runtime ID*, uniquely identifies the artefact instance about to be started. The reason for storing the IP address is to enable SM to open connections towards the host, if necessary. The SM looks up an existing *host policy specification*, based on the contents of entry. If one is found, it is associated with the entry and stored in an internal database. The policy specifies the constraints of the host in running the artefact under the control of user ID. If the policy allows artefact execution, the SM creates an asymmetric key pair and a certificate for the artefact instance. The runtime ID, the associated policy specification, and the key pair and certificate (if created) are together signed with the SM's private key. The signature and the signed items define the *host license*, which is sent to the host BM over the HTTPS connection established previously[2].

If the received license specifies that the host is allowed to execute the artefact, phase 5 can begin and the BL is started. Otherwise, an error message is sent to the developer and the artefact execution is aborted. When the BL is started it expects to find a copy of host license from the BM. This enables phase 6 to begin. The BL uses the SM public key to verify the signature from the license and determine if it is being executed on a genuine host. If that is the case, the BL constructs a request for SM consisting of the license from BM, its artefact ID, and a nonce. The BL is not aware of its own image digest. Therefore, the inclusion of artefact ID in the request lets SM detect if the license presented was generated for a different type of artefact. The request is sent to the SM over HTTPS. The SM uses the artefact ID to lookup the image digest stored during phase 1 and compares it with the one stored in the license. If there is no match, an error is returned to the artefact which stops execution. Otherwise, the SM looks up an existing *artefact policy specification*. If none is found, a default one is used. The looked up artefact policy is associated with the runtime ID maintained by SM. The SM prepares then a reply consisting of the runtime ID and policy specification. These items are signed by the SM and the resulting *artefact license* is transported over the established HTTPS connection to the BL. The BL learns through the license if it is allowed to execute the artefact code, or if it is supposed to cease execution. Also, the BL is expected to forward the license to the BM before a timeout occurs. In case of a timeout, the BM will stop BL's execution. Otherwise, the protocol completes successfully having created mutual trust and the pipeline can begin execution.

[1]In this case, formal remote attestation is possible.

[2]The BM will be able to observe the private key transmitted to the BL, but according to the assumptions in Sec. IV-A, the hosts are not instrumented to exploit this situation in an undesirable way.

V. Generalisations from the VP as Design Cues for IoT Systems with Flexible Security

The specifications of the high-level security and trust requirements and implementation of mechanisms for collaborative AI design using pipelines and artefacts gives detailed insights into the security architectures and mechanisms for IoT system using similar concepts. We will generalise our findings:

1) *Collaborative and distributed workflows in IoT service chains require security and compliance verification and enforcement along the workflow,* i. e., *in an "horizontal way" (authentication for cooperation).* The policies, *e. g.*, for DRM or privacy, need to be enforced on the involved IoT element, *e. g.*, edge devices, along the east/west-bound interfaces of the artefacts.
2) *Code mobility in IoT,* i. e., *use of portable container, requires security and compliance checking on the IoT host,* i. e., *in a "vertical way" (authentication for resources)*.
3) *The use of a virtualisation concepts and of containers in IoT need the verification of integrity of containers and hosts.* Therefore, the container and the host need to implement a module/layer to verify each others integrity.
4) *Docker's container technology is a great tool for component-based system design for AI system and probably also for IoT service chains.*
5) *Docker's mechanisms to ensure that code in the container can not be by-passed are still limited.*
6) *A centralised Security Manager is typically a single-point-of-failure/attack and needs to be distributed in future.*

An initial analysis of the security threads against the VP is given in [21]. It lists 13 simple threats by unexperienced malicious users, ranging from by passing license constraints to artefact execution on wrong VPs, and techniques to protect against them. While these techniques are not yet fully implemented in the Bonseyes MP and VP demonstrator, their availability suggests that VP-enabled architectures can be secured against them. More difficult to solve are advanced threats from skilled malicious users. A major approach is to raise the overall trust level by using TPMs.

VI. Conclusion

The security architecture for the AI marketplace and the authentication protocol described in Sec. IV are the main contributions of this paper. In addition, the investigation of the security, DRM, and trust requirements for pipeline-based collaborative AI system design and the specification of the VP enabled the investigation of similar concepts for future IoT architectures. These IoT systems will apply service chains where flexible and adaptive security per service is needed.

The suggested solution builds a *VP* which forms a virtual boarder around the pipeline or service chain. This chain facilitates also the slicing of the full infrastructure, *i. e.,* of the application infrastructure and of the network. The use of containers in such IoT systems requires the authentication of both, containers and execution environments.

Although only Docker containers were considered in this work, our solution is easily extensible towards other types of virtualized environments such as KVM, LXC, or VMware.

Container and virtualisation technologies are still rapidly evolving in their capabilities. Hence, their future capabilities might impact the features of the VP, *e. g.,* the VP might need to enforce that a container starts always the Bonseyes Layer either by Docker mechanisms or by other verification mechanisms. Another future feature for the VP might be the automatic policy compliance matching, enforcement and eventually policy orchestration, *e. g.,* using distributed ledgers [22]. Another future enhancement is the use of distributed TPMs [23].

Acknowledgments

The authors would like to thank the Bonseyes team for extensive discussions, in particular our gratitude goes to Tim Llewellyn, Samuel Fricker, Lorenzo Keller and Yuliyan Maksimov. This project has received funding from the European Unions Horizon 2020 research and innovation programme under grant agreement No 732204 (Bonseyes). This work is supported by the Swiss State Secretariat for Education Research and Innovation (SERI) under contract number 16.0159. The opinions expressed and arguments employed herein do not necessarily reflect the official views of these funding bodies.

References

[1] F. Mattern and C. Floerkemeier. From the Internet of Computers to the Internet of Things. In *From active data management to event-based systems and more*. Springer, 2010.

[2] ITU-T. Overview of the internet of things. ITU-T Recommendation ITU-T Y.4000/Y.2060 (06/2012), 6 2012.

[3] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli. Fog Computing and its Role in the Internet of Things. In *Proceedings of the First MCC workshop on Mobile cloud computing*, 2012.

[4] T. Taleb et al. On multi-access edge computing: A survey of the emerging 5G network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[5] V. Sciancalepore, F. Cirillo, and X. Costa-Perez. Slice as a Service (SlaaS) Optimal IoT Slice Resources Orchestration. In *Proc. of GLOBECOM 2017*, 2017.

[6] Raspberry Pi Foundation. Raspberry Pi. www.raspberrypi.org.

[7] L. Tung. Raspberry Pi: 14 million sold, 10 million made in the UK. ZDnet Information available at:https://www.zdnet.com/article/14-million-raspberry-pis-sold-10-million-made-in-the-uk/, 2017.

[8] P. Bellavista and A. Zanni. Feasibility of Fog Computing Deployment Based on Docker Containerization over RaspberryPi. In *Proc. of the 18th Int. Conf. on Distributed Computing and Networking*, 2017.

[9] P. Mohagheghi and R. Conradi. An empirical investigation of software reuse benefits in a large telecom product. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 17(3):13, 2008.

[10] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes. Borg, Omega, and Kubernetes. *Queue*, 14(1):10:70–10:93, January 2016.

[11] G. Press. TechRadar: 6 Hot Internet of Things (IoT) Security Technologies. Forrester Research Information available at: https://www.forbes.com/sites/gilpress/2017/03/20/6-hot-internet-of-things-iot-security-technologies, 2017.

[12] H. Gantikow, C. Reich, M. Knahl, and N. Clarke. Providing security in container-based hpc runtime environments. In *International Conference on High Performance Computing*, pages 685–695. Springer, 2016.

[13] R. Yasrab. Mitigating docker security issues. *arXiv preprint arXiv:1804.05039*, 2018.

[14] C. C. Wood. Why information security is now multi-disciplinary, multi-departmental, and multi-organizational in nature. *Computer Fraud & Security*, 2004(1):16–17, 2004.

[15] J. Sifakis. System Design in the Era of IoT? Meeting the Autonomy Challenge. In *Proc. of the 1st Int. Workshop on Methods and Tools for Rigorous System Design (MeTRiD 2018)*, 2018.

[16] R. Roman, J. Lopez, and M. Mambo. Mobile edge computing, fog et al.: A survey and analysis of security threats and challenges. *Future Generation Computer Systems*, 78:680–698, 2018.

[17] S. Yegulalp. Data in, intelligence out: Machine learning pipelines demystified. Information available at: www.InfoWorld.com, August 2018.

[18] D. Baylor et al. TFX: A tensorflow-based production-scale machine learning platform. In *Proc. of 23rd ACM SIGKDD*. ACM, 2017.

[19] T. Llewellynn and other. Bonseyes: platform for open development of systems of artificial intelligence. In *Proc. of the Computing Frontiers Conference*. ACM, 2017.

[20] M. de Prado, J. Su, R. Dahyot, R. Saeed, L. Keller, and N. Vallez. AI Pipeline – Bringing AI to You. Submitted; available on request from: https://www.bonseyes.com/, 2018.

[21] V. A. Mehri, D. Ilie, and K. Tutschku. Privacy and DRM requirements for collaborative development of ai application. In *Proceedings of ARES*. ACM, Hamburg, Germany, August 2018.

[22] Tim Swanson. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. Report available at: http://www.ofnumbers.com/wp-content/uploads/2015/04/Permissioned-distributed-ledgers.pdf, 2015.

[23] P. Wagner et al. Distributed usage control enforcement through trusted platform modules and sgx enclaves. In *Proc. of 23rd ACM on Symp. on Access Control Models and Tech.* ACM, 2018.