Towards a Secure Proxy-based Architecture for Collaborative AI Engineering

Roman-Valentyn Tkachuk
Blekinge Institute of Technology
Karlskrona, Sweden
roman-valentyn.tkachuk@bth.se

Dragos Ilie, Ph.D.
Blekinge Institute of Technology
Karlskrona, Sweden
dragos.ilie@bth.se

Kurt Tutschku, Ph.D.
Blekinge Institute of Technology
Karlskrona, Sweden
kurt.tutschku@bth.se

***Abstract*—In this paper, we investigate how to design a security architecture of a Platform-as-a-Service (PaaS) solution, denoted as Secure Virtual Premise (SVP), for collaborative and distributed AI engineering using AI artifacts and Machine Learning (ML) pipelines. Artifacts are re-usable software objects which are a) tradeable in marketplaces, b) implemented by containers, c) offer AI functions as microservices, and, d) can form service chains, denoted as AI pipelines. Collaborative engineering is facilitated by the trading and (re-)using artifacts and, thus, accelerating the AI application design.**

The security architecture of the SVP is built around the security needs of collaborative AI engineering and uses a proxy concept for microservices. The proxy shields the AI artifact and pipelines from outside adversaries as well as from misbehaving users, thus building trust among the collaborating parties. We identify the security needs of collaborative AI engineering, derive the security challenges, outline the SVP's architecture, and describe its security capabilities and its implementation, which is currently in use with several AI developer communities. Furthermore, we evaluate the SVP's Technology Readiness Level (TRL) with regard to collaborative AI engineering and data security.

***Index Terms*—Security Architecture; Trusted and Collaborative AI engineering; Proxy-based Architecture;**

I. Introduction

Access to machine learning (ML) algorithms implemented in freely available and easy-to-use software development kits has lowered the bar for incorporating artificial intelligence (AI) into general purpose applications. However, this simplicity is deceiving. Building robust and efficient AI features requires a deep understanding on how AI algorithms and data, *i. e.,* AI artifacts, interact. Moreover, ML requires large training data sets which typically exist in large enterprises that can afford to develop ML concepts. Small companies, however, might collect data or define models but must rely on collaborations with distributed stakeholders for developing AI solutions. Hence, trusted collaborations in AI engineering are needed for empowering ML beyond large companies [1].

The authors of [1], [2] developed a *collaborative form of AI engineering* within a H2020 project. This approach uses agile methods, *e. g.,* continuous integration, to accelerate collaborative AI development. Furthermore, the project has implemented an AI marketplace (MP), which is an open platform for trading AI artifacts. However, it requires a *Secure Virtual Premise (SVP)* which is a variate of a PaaS for distributed, secure and trusted AI engineering using artifacts and which is presented here. The SVP connects distributed computation and storage resources (both physical and virtual) into a large virtual resource space for AI training and enforces a perimeter around this space. Hence, this SVP can be called a federation of distributed resources.

The MP supports the controlled exchange of AI artifacts, *i. e.,* of algorithms and data, among third parties. An application developer can obtain licensed access (*e. g.,* by paying fees) to these artifacts and use them locally, *i. e.,* in the local parts of the SVP. As a result, the developers can focus on application design while having access to AI artifacts and AI data, thus accelerating the system development.

In this context, the concerns arise that malicious users may try to bypass the constraints imposed by the license, or even share artifacts with unlicensed users. This would threaten not only the intellectual property rights (IPRs) of artifact owners, but also may create significant data privacy issues.

The main contributions of this paper are the identification of the security challenges and requirements for collaborative AI engineering using the SVP, the definition of threat models, and the specification of a proxy-based PaaS security architecture that is based on a threat-analysis. The *proxies* are implemented close to AI artifacts, run *on-demand* (*i. e.,* dynamically when artifacts are instantiated), and shield the artifacts and pipelines from adversaries. To our knowledge, the SVP is the first PaaS that meets the needs of distributed and collaborative AI engineering and which enables users to have certain administration rights within distributed computational resources of SVP.

The paper uses the following *methodology* for defining the SVP's security architecture. First, it specifies the assets to be protected and identifies potential attack paths. Then, it uses a STRIDE approach [3] to define a threat profile against the assets. Finally, it translates the profile into requirements and specifies the architecture for the SVP.

The paper is structured as follows. Sec. II outlines the considered approach for *collaborative AI engineering*. Section III discusses the threats and security requirements of this concept. Sec. IV introduces the SVP's secure design and discusses how it meets the requirements of the threat profile. Sec. V describes implementation and operation of the SVP. Sec. VI describes evaluation of the current SVP implementation. Sec. VII describes related work on securing digital marketplaces, AI engineering, and proxy-based security concepts. Finally, Sec. VIII sums up the security capabilities of the SVP and provides an outlook to future research.

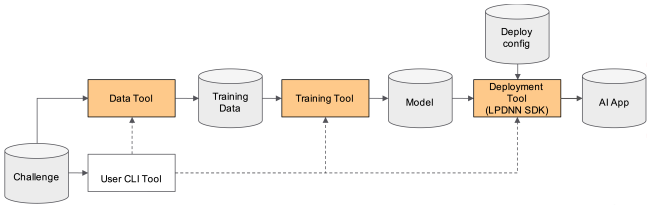

Fig. 1: AI Training Pipeline, after [2]

II. Collaborative AI Engineering

The considered systematic engineering of data-driven AI solutions [2] is facilitated by the use of AI artifacts and ML pipelines, called *AI pipelines*. These pipelines are used to structure and eventually automate ML workflows. They consist of several modular steps to generate an AI application or to benchmark its accuracy. The solution of [2] assumes that each pipeline step can be provided by an *AI Tool* or *AI artifact*, which is implemented in a Docker container [4] and is similar to a microservice [5]. Fig. 1 shows the structure of an AI training pipeline comprising three AI tools: Data (Extraction) Tool, Training Tool, and Deployment Tool. The data is exchanged between artifacts through file objects, *e. g.,* volumes. Fig. 1 shows AI tools in containers, which are represented by rectangles, and data objects as cylinders.

Another key element of the AI engineering in [1] is the *Marketplace (MP)* for AI artifacts. The MP enables stakeholders to meet, offer, find and exchange artifacts. Here, the users agree on terms and conditions for collaborations and artifact usage, *i. e.,* on human- and machine-readable licenses.

While the MP is open, the Secure Virtual Premise (SVP) needs to be a closed and protected space where the AI computing tasks take place [2]. The SVP binds and federates eventually distributed compute and storage resources (due to the potential geographic distribution of stakeholders). A SVP is defined for a specific AI application design task and multiple SVPs for different applications may exist. Furthermore, a *SVP provider* may operate a SVP in a commercial way.

Only eligible users should access a specific SVP. Resources and users on the inside (*i. e.,* on *premise*) are typically trustworthy and compliant, and everything on the outside is untrusted and potentially malicious.

III. Security Challenges and Requirements

From a security perspective, the AI artifacts are the assets at risk and the SVP is the security scheme that protects them. A successful attack on this scheme will grant adversary access to the asset, such that the asset can be exfiltrated or configured to engage in potentially malicious behavior. To have a complete security definition, a threat model is also needed to outline the capabilities of the adversaries. This is done next.

A. Inside and Outside Adversaries

We choose a binary model consisting of adversaries either on the *inside* or on the *outside* of the SVP. Inside adversaries are legitimate SVP users, formerly trustworthy, but now turned rogue (*e. g.,* avoiding license fees). Outside adversaries have no legitimate access to the SVP and thus have less ”power” to attack the security guarantee.

Protection against outside adversaries is typically enforced by access control (AC) mechanisms located at the boundary of the SVP. Thus, the security level is directly related to their robustness. Inside adversaries that have successfully passed the AC scheme can access the SVP resources, *e. g.,* AI artifact, that they are authorized for. Hence, they have the opportunity to tamper with resources, *e. g.,* attackers could instrument the execution environment (EE) to reverse engineer (RE) and bypass the license checking.

In general, handling inside adversaries is quite difficult. The authors of [6], [7] approached the problem by classifying adversaries into *regular users* and *malicious users*. The main difference between them is that regular users are not able to tamper with the software. By excluding malicious users from the threat model authors were able to design a scheme based on public-key cryptography to ensure compliance from both outside and inside adversaries.

B. Tampering with the System Software

Software tampering (ST) is an umbrella term describing activities that aim at introducing unwanted modifications to binaries or to processes running on a host. ST provides possibilities for multiple attacks, including RE. Software-based defenses against ST rely primarily on code obfuscation [8]. Hardware-based defenses [9] are more robust to software attacks, but are dependent on hardware from a specific CPU manufacturer and usually incompatible with each other.

The threat posed by ST through RE to the SVP exists because skilled *inside* adversaries (*i. e.,* legitimate users turned rogue) have direct access (physical or remote) to the host running the artifact. By removing direct access, the RE possibilities are severely curtailed. This requires a slight modification to the threat model defined in [6], [7]. In the modified model, the users and hosts are no longer trustworthy and no longer inside the SVP. They can still access resources from the SVP, but only through well-defined interfaces as shown by the blue rectangles inside the User host box in Fig. 2. An interface is a stub proxying requests towards AI objects in the SVP and receiving responses from the SVP, much similar to the stub concept used by remote procedure calls (RPC). Thus, the user host interacts through interfaces and the actual artifact is executing remotely on hosts belonging to the SVP, which acts as an infrastructure towards the users.

We assume that SVP providers are not malicious nor do they collude with adversaries. We think this is a reasonable assumption because otherwise the SVP providers would put their own business model at risk and lose their reputation.

C. Threat Model

Fig. 2 shows the functional SVP architecture without security mechanisms. An AI user communicates with the SVP and the AI components through two different Application Programming Interfaces (APIs). These are shown as blue rectangles inside the left box denoted User host. The user end of

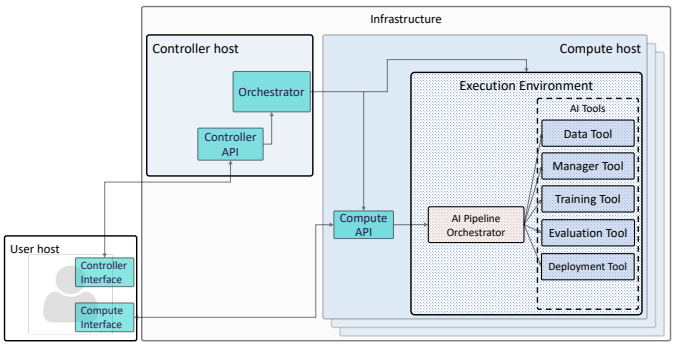

Fig. 2: SVP architecture without security mechanisms.

the APIs is connected to the server end on the Controller host (CtrlH) and Compute host (CpH). The Controller Interface enables the user to instruct the *orchestrator* to provision the EE inside the CpHs. The Compute Interface is used to download AI artifacts from a MP repository (not shown in the figure) to the EE on the CpH, to assemble the AI pipeline, and to manage the pipeline and its components during their life cycle. The AI artifacts inside the EE are the main assets at risk.

The concern is that these assets can be exfiltrated, configured to disobey the license terms, or modified to operate maliciously. An attacker operating the User host can follow two paths to reach the main asset.

The first attack path begins at the Compute interface and ends at the Compute API (CpAPI) component on the CpH. The Compute Interface can execute CpAPI calls that may result in potentially harmful actions towards elements inside of the EE. These types of API calls must be allowed only for privileged users. Thus, a potential threat is that attackers can escalate their privileges by spoofing the identity of a privileged user. Although the API does not provide means to exfiltrate AI artifacts, the attacker may be able to launch API calls that modify ownership and access rights to the pipeline and its components and to exfiltrate pipeline results. Even when the attackers can only impersonate non-privileged users, they can engage in damaging activities such as stopping, removing or reconfiguring pipelines in order to tamper with the output of the pipeline (*e. g.*, degrading output quality) or to overload the infrastructure through Denial-of-Service (DoS) attacks.

The second attack path starts at the Controller interface and continues, first through the Controller API (CtrlAPI), and then through the Orchestrator on the CtrlH. An attacker who is able to infiltrate the system over this path obtains backdoor access to the EE and to the CpAPI. The Orchestrator can execute operations that interfere with AI artifacts and pipelines. Only privileged users must be able to execute these operations. However, the Orchestrator is not directly exposed to the User host and it can only be reached through the CtrlAPI. Thus, an attacker must mount a successful privilege escalation attack against the CtrlAPI in order to control the EE through the Orchestrator. For example, an attacker from a User host would need to spoof the identity of users with privileged access. If successful, the adversary will obtain full access to the CpH OS and to its EE. Implicitly, the attacker will have access to the AI components inside and will be able to interfere with CpAPI calls. Tampering with CtrlAPI calls presents similar risks as described in the previous paragraph.

Even if attackers do not control the User host, they may able to interfere with the data flows between the User host, CtrlHs and CpHs. Unless the network traffic is protected, attacks on confidentiality, integrity, and authenticity become possible. For example, attackers would be able to extract credentials that enable them to impersonate legitimate users. DoS attacks from outside the SVP towards the hosts in Fig. 2 are also a potential threat. However, the SVP mechanisms described here do not aim to address this type of threat, as DoS attack mitigation typically requires the involvement of network operators [10].

TABLE I: Identified threats

Threat	Element	Description
T01	CpAPI	Illegitimate user accesses CpAPI.
T02	CpAPI	Spoofing the identity of privileged users to execute undesirable or dangerous CpAPI calls towards the EE.
T03	User-CpAPI traffic	Interception and possibly tampering of data exchanged between User host and CpAPI. May enable T-01 and T-02.
T04	CtrlAPI	Illegitimate user accesses CtrlAPI.
T05	CtrlAPI	Spoofing the identity of privileged users to the CtrlAPI. Enables access to EEs via the Orchestrator, and exfiltration or damage to AI objects.
T06	User-CtrlAPI traffic	Interception and likely tampering of data exchanged between User host and CtrlAPI. May enable T-01 and T-02.
T07	CtrlAPI	Malicious privileged user secretly exfiltrates artifacts, and/or conducts operations with the intent of creating harm to the pipeline infrastructure.
T08	CpAPI	Malicious privileged user secretly conducts operations with the intent of creating harm to the pipeline and its output.

TABLE II: Threat profile

Threat	S	T	R	I	D	E
T01	X			X		X
T02	X			X	X	X
T03		X		X	X	
T04	X			X		X
T05	X			X	X	X
T06		X		X	X	
T07		X	X		X	
T08		X	X		X	

Using the above identified threats, *c. f.*, Table I, we use the STRIDE approach [3] to produce the SVP's threat profile, *c. f.*, Table II. Each of the letters in STRIDE denotes a specific threat category against a desirable security property: spoofing *vs.* authentication, tampering *vs.* integrity, repudiation *vs.* non-repudiation, information disclosure *vs.* confidentiality, DoS *vs.* availability, and elevation of privilege *vs.* authorization. As a result, we identified the requirements shown in Table III.

TABLE III: Security requirements.

Req.	Description
R01	User's identity and authentication must be based on X.509 digital certificates. Mitigates spoofing (T01, T02, T04, T05).
R02	User host to CtrlH / CpH communication must use HTTPS (SSL/ TLS) with certificate-based mutual authentication. Mitigates tampering (T03, T06) and information disclosure (T03, T06).
R03	CpAPI must use role-based access control (RBAC) to control which users can execute privileged API calls. Mitigates information disclosure (T01, T02), DoS (T02) and elevation of privilege (T01, T02).
R04	CtrlAPI must use RBAC to control which users can execute privileged API calls. Mitigates information disclosure (T04, T05), DoS (T05) and elevation of privilege (T04, T05).
R05	CpH must use secure logging that cannot be tampered with from the CpAPI to log all actions. Mitigates repudiation (T07). Secure logging does not mitigate tampering and DoS (T07) but provides information about unwanted activities, which allows the system administrator to take action against the malicious users.
R06	CtrlH must use secure logging that cannot be tampered with from the CtrlAPI to log all API calls. Mitigates repudiation (T08). Secure logging does not mitigate tampering and DoS (T08) but provides information about unwanted activities, which allows the system administrator to take action against the malicious users.

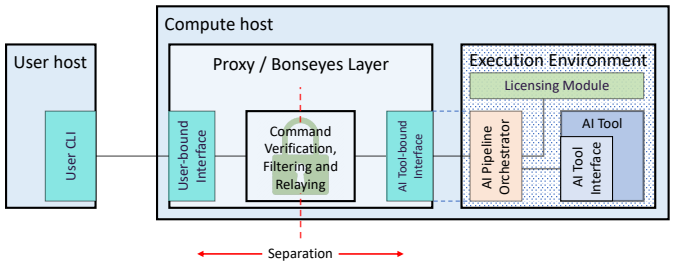

Fig. 3: Security and Separation by BL Reverse Proxy.

IV. Proposed Security Architecture

Next, we will present the SVP components used to address and implement the security requirements derived in Sec. III.

A. A Proxy Concept for Artifact Security

Proxies provide security by applying the security concepts of *separation* and *verification* [11]. They implement services such as authentication, encryption of data communication, network isolation, or access control. [12] proposes proxies for addressing security, trust and privacy for collaboration in multicloud environments. However, they were not located close to the offered services. Lately, so-called *sidecar proxies* [13], [14] have been suggested. They are attached to parent services and provide them with supporting features.

The SVP architecture employs a reverse proxy which is denoted as *Proxy / Bonseyes Layer (BL)*. This proxy shields the artifact from misuse, is implemented next to the artifact like a sidecar proxy, and is started on-demand when the artifact is instantiated, *c. f.*, Fig. 3. The Proxy / BL is executed on the CpH and it interfaces to the User host by the *User-bound interface*. A user can connect to an artifact (here *AI tool*) only through this interface. Furthermore, the BL interfaces to the AI tool via the *AI Tool-bound interface*. The Proxy / BL relays only verified and filtered commands between these interfaces. These interfaces manage the control of the artifact and the pipeline. The AI Tool-bound interface has the additional advantage that the proxy can be adapted to arbitrary containerized tools, *i. e.*, the available AI assets don't need to be adapted. A *trust boundary* between the user and AI artifact is implemented close to the artifact by permitting connections only to the Proxy / BL while having the Proxy / BL verify the eligibility of received commands. We believe that it is a compelling feature of this proxy concept that it provides clear separation without requiring any explicit support from the artifact. This reduces the efforts of developer since they can focus on AI functions only and don't need to build security mechanisms into the artifact.

Fig. 3 depicts the *AI Pipeline Orchestrator*. It coordinates activities among the artifacts by exposing an interface to the AI tools. Also, it performs the syntax translation from a user command to the function calls of the AI tool. Unlike the Proxy / BL, the orchestrator usually does not perform any security checks, unless it has been delegated to do so.

B. Elements of the SVP Architecture

The requirements listed in Table III indicate the need to support the concept of a digital identity. Hence, the SVP architecture follows the NIST guidelines for digital identities [15] and relies here on three specific entities: Credential Service Provider (CSP), Verifier, and Relaying Party (RP). The CSP enables users (*applicants*) to enroll in digital identity services. If successful, the applicant becomes a *subscriber*, who will eventually attempt to claim access to a secured service. The Verifier will interact with the user to ascertain the *claimant* is a valid subscriber (*i. e.*, authentication is performed). The Verifier may also contact the CSP to obtain additional claimant attributes that are required by the RBAC in use. The result of the authentication is passed on to the RP, which can complete the RBAC and determine if the type of access required by the claimant is allowed (*i. e.*, authorization is performed).

The applicant who wishes to utilize the services of the SVP must undergo first a process called *identity proofing*, where the user's real-world identity is connected to a digital identity. The CSP contains a sub-function called Registration Authority (RA), which is the equivalent of a front desk where the physical applicant proves their real-world identity (*e. g.*, by showing a passport). Upon successful identification, the RA invokes another sub-function of the CSP, the Certification Authority (CA), to create a X.509 digital user certificate. The certificate is installed in the applicant's web browser (on User host in Fig. 4) and becomes the applicant's digital identify. Furthermore, it enables the Verifier and RP to conduct certificate-based authentication and authorization of users. **This addresses requirement R01**, *c. f.*, Table III.

For simplicity, we have kept the CA and RA inside the CSP, although in some scenarios they can be separate entities managed by different organizations. In Fig. 4, they are shown as the box denoted "Credential Service Provider" inside the CtrlH. Currently, the architecture contains two RPs: the Controller RP located within the Controller API and the Proxy RP

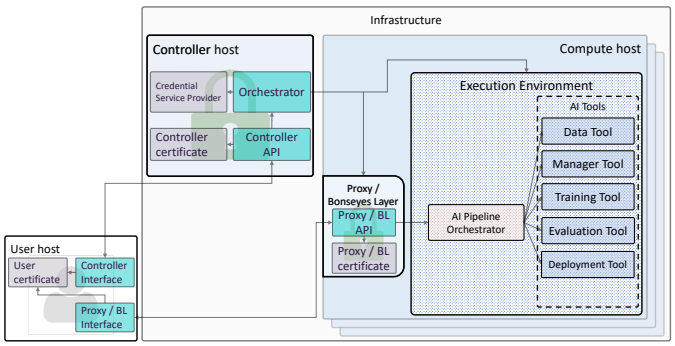

Fig. 4: Generic architecture of SVP.

TABLE IV: Location of digital identity elements

Entity	Location
Applicant/Subscriber/Claimant	User host
CSP (with RA and CA)	CtrlH
Controller RP (incl. Verifier)	CtrlAPI on CtrlH
Proxy RP (incl. Verifier)	Proxy / BL API on CpH

located within the Proxy / BL API. Again, for simplicity, the functionality of the Verifier is merged in each of the RPs.

The CA is also used to produce digital server certificates for the elements listed in Table IV. These are installed in web servers running at the location shown in the same table and serve a dual-purpose. First, they enable authentication of the elements towards the User host. Secondly, they are fundamental in enabling encrypted communication over HTTPS. **This addresses requirement R02.**

The Controller RP interacts with the CSP to implement a simple RBAC scheme that differentiates between *regular* and *privileged* users. The CSP provides the subscriber attributes required by the Controller RP to enforce the scheme. **This addresses requirement R04.** All requests and replies that are processed by the CtrlAPI (with associated RP) are logged at the CtrlH. Logging features cannot be controlled from the CtrlAPI and thus logs can't be tampered with this way. **This addresses requirement R06**.

The Proxy / BL works as an interceptor between the User host and the EE on the CpH. Through the Proxy / BL, the User host can interact with the AI pipeline and various AI tools. Incoming Proxy API calls from the User host are processed by the Proxy / BL into low-level sets of API calls, each targeting specific components within the EE. This is required because although the Proxy BL exposes a uniform API towards the User host, the APIs of the components within the EE may vary substantially. Similar to the Controller RP, the RBAC scheme for the Proxy RP must also be able to differentiate between regular an privileged users. However, it must be able to support attributes that describe the type of low-level API calls that are allowed towards a component. In addition, all API calls (both high- and low-level) are recorded similarly as explained for the Controller RP. **This addresses requirements R03 and R05.**

V. SVP Implementation

The structure of the SVP implementation is shown in Fig. 5. It uses currently available technologies to implement the security mechanisms and requirements, cf. Table III.

A. Controller Host (CtrlH)

At the CtrlH, the *CSP* is implemented with the *OpenSSL* as a bash script which manages certificates in an automated manner. *Controller certificate* is created initially and *User* and *Proxy / BL certificates* are created on demand.

The *CtrlAPI* is implemented as a *Python-Flask* web application which runs under *uWSGI* web server. uWSGI server is started with an X.509 CtrlH certificate generated by the CSP.

As part of the system design, the architecture of the *Orchestrator* can follow one of the following patterns: *a) Client-Server*, *b) Server-Only*, as was described in [16]. In a Server-Only architecture, the Orchestrator software is installed only on the CtrlH, whereas in a Client-Server architecture there is also agent software installed on CpHs. We chose the *Client-Server* architecture because this allows the agent to individually check the integrity of downloaded AI artifacts. Orchestrator consists of two components – *Security Manager* on Server-side and *Bonseyes Module (BM)* on Client-side. They are implemented as Python-Flask web applications running under the uWSGI web server, with a high degree of integration between them. Namely, each BM belongs to a specific Security Manager and their communication is encrypted using HTTPS.

B. Compute Host (CpH)

The provisioning of the CpH is performed by the the BM, which requests the *Proxy / BL certificate* from the Security Manager and downloads and starts all *Proxy / BL* and *EE* related components from a repository (not shown in Fig. 5).

The *Proxy / BL* is implemented in the form of a Docker container. It intercepts the communication between the User host and the EE and allows only a strictly defined set of *Proxy / BL Web API* calls to be executed. The Proxy / BL Web API is implemented as a Python-Flask web application which runs under the uWSGI web server. The server is started with X.509 Proxy / BL certificate generated by CSP. In order to communicate with the proxy's Web API, a user executes commands over *User Command Line Interface (User CLI)*. The User CLI sends both privileged and regular commands, which are filtered by the proxy, based on the user's role.

The EE resides inside of the CpH and contains all the needed data and tools to conduct AI pipeline execution. The *Import Volume* is used to store information within the EE, which can later be used during AI pipeline execution. It is implemented as a folder in the CpH's file system which is attached as a volume to Proxy / BL docker container. AI artifact users can upload data into Import Volume, but cannot execute or download it - this data is designed to be used only by *Bonseyes Command Line Interface (Bonseyes CLI)*. The Bonseyes CLI is the implementation of the AI Pipeline Orchestrator shown in Fig. 4. Its purpose is to control the respective AI Tools and provide them mediated access to

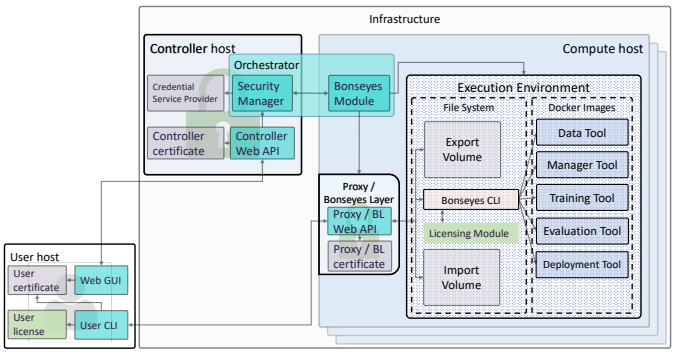

Fig. 5: Implemented architecture of the SVP.

data residing inside Import Volume. The Bonseyes CLI is implemented as a Python module that can be invoked by executing User CLI commands and is attached to Proxy / BL docker container as a volume. AI Tools are implemented in Docker containers which are started by the Bonseyes CLI and receive input from it. After the AI Tool finished execution, the Bonseyes CLI gathers the output and places it in *Export Volume*, which is used to transfer information out from the EE, thus from the SVP. The Export Volume is implemented in the same manner as an the Import Volume.

The *Licensing Module* is another essential security mechanism that resides within the EE. It is implemented with Python and ensures compliant usage of the artifact according to the *User License*. The user needs to upload the User license to the SVP and its validity is checked by Bonseyes CLI before the execution of any command from the user.

VI. SVP Evaluation

An system-wide evaluation of the SVP's security level is difficult since it combines a variety of complex security mechanisms. This makes it prohibitive to rely on symbolical verification (*i. e.*, formal model) [17] or computational approaches (*e. g.*, reduction techniques) [18] to assess the security of the system. Hence, we decided to apply here a more differentiated approach which builds on practical use-cases to verify its usability and security.

The SVP implementation was tested by several real-world AI engineering cases. These use cases permit judging the SVP's maturity and thus evaluating its *Technology Readiness Level (TRL)* [19]. The use cases were carried out by stakeholders and users of the collaborative AI engineering process of [1], [2] and in their labs and computing infrastructures. The stakeholders comprised two companies which were interested in AI application design, an SME (small- and medium-sized company) and a large automotive OEM manufacturer, as well as two international universities developing AI algorithms. The use-cases comprised the tasks of "AI training" and "AI model benchmarking" using AI pipelines, *c.f.*, Fig. 1. The various actors in the use cases were assigned one of the following roles: *legitimate SVP provider, legitimate SVP user, rogue SVP user*, and *outside adversary*.

Throughout these use cases, a legitimate SVP user and SVP provider were always able to execute the AI pipelines and to exchange the artifacts (Remark: this confirms the AI engineering capabilities of the SVP). However, a rogue SVP user and an outside adversary were experiencing the enforcement of security mechanisms. In detail, an outside adversary was given the task to penetrate into the EE and interfere with the process of pipeline execution. When an adversary attempted a connection, it was blocked by the X.509 digital certificate authentication. Moreover, since only network port 443 was open, there was no other point of interaction offered by the system. In parallel, the embedded logging system recorded every unsuccessful connection attempt for further analysis.

Another use case introduced a rogue SVP user on the inside of SVP's perimeter, having two malicious goals: 1) obtain the AI data and tools which are stored in the execution environment, and 2) execute an AI benchmarking tool with an expired license. By design, the Proxy / BL enforces for regular users a store-only mode for the data inside of the Import Volume. This prevents data exfiltration from the Import Volume. Only privileged users or programs can obtain direct access to data inside the Import Volume. In completing the first goal, the rogue user must authenticate himself using his digital certificate. Using the identity from the certificate, the RBAC mechanism determined that regular user privileges apply, and thus blocked read access to the assets.

In order to complete the second goal, the rogue user modified the license, which is encoded in plain-text JSON format, to extend the license expiration date. The modified license was uploaded as part of the pipeline construction process to the SVP along with the original license signature. This signature was computed over the sha256 hash of the license content, using a private key known only to the MP. During license verification, the SVP first verified the license signature using MP's public key. Since the license content was changed, the signature verification failed. Thus, the execution of the benchmarking tool was blocked.

The use cases confirmed that the AI and security requirements were appropriately addressed and that the security capabilities of the SVP provide sufficient trust for enabling the collaborative AI engineering process of [1], [2]. Moreover, users of this AI engineering process concluded in [20] that the SVP implementation provided by us achieves at least TRL 4 ("Technology validated in lab") and eventually also TRL 5 (TRL 4 + "... relevant environment (industrially relevant environment in the case of key enabling technologies)"). While the TRL shows the maturity of the approach, we suggest to carry out a more threat-focused verification in the future, *e. g.*, a full-fledged penetration test of a live SVP.

VII. Related Work

Protecting collaborative and distributed software engineering environments involves a large number of technologies. Next, we outline important related works this and outline the context for the research presented here.

Securing Docker containers has recently gained significant attention [21], [22]. Our approach complements the Docker-

specific efforts by not relying on a specific containerization technique and adding another layer of protection.

Software marketplaces for Virtual Network Functions (VNFs) and AI are going back as early as 2014 [23] but rarely support edge clouds or multi-stakeholder collaborations. Trusted and secure collaborative software marketplace substrates and platforms using Blockchains have been suggested lately [24], [25]. Ericsson's Nubo platform [24], however, doesn't support licenses and hasn't yet an option for distributed computation. The OceanProtocol [25] is the closest known larger-scale concept to the MP [1] and the SVP concepts. However, it also doesn't enforce licenses and requires the Docker containers to be adapted by a Blochchain layer.

Related work on the network scope and dynamics of proxies were discussed in Section IV-A in order to highlight the features of our mechanism. Recent proxy solutions for micro services and service chains are discussed in [26], [27]. Hereby, [26] is close to the proposed *Proxy / BL* concept. However, it is tailored to offline devices and the proposed architecture is based on the usage of *secure cryptoprocessors*, such as the trusted platform module (TPM). Thus, their proposal is not applicable on platforms lacking this type of devices. Distributed business processes employing proxies are detailed in [27]. However, their proposed proxy-based controller is designed for protecting document flows only. Other uses, such as chained services as AI pipelines are, not considered.

VIII. Summary and Outlook

In this work we designed a PaaS solution for securing containerized AI artifacts and enabling trust in collaborative AI engineering. The design is based on a rigours threat and security analysis process: *threat modeling* $\rightarrow$ *security requirements definition* $\rightarrow$ *security services development*. The idea of securing AI artifacts and pipelines is implemented by the use of the *Proxy / Bonseyes Layer (BL)*, which implements the security concepts of *separation* and *verification*. The BL is placed on-demand, close to each artifact, enforces licenses and prevents direct access from users. The evaluation of the SVP has demonstrated that it has matured security and can be applied in real-world collaborative AI engineering using software marketplaces and AI pipeline concepts.

We think that our solution allows the SVP to be stretched into devices on edge clouds. In this scenario, edge devices acts as CpHs that are managed from the SVP CtrlH. However, in order for the SVP to remain trustworthy, the edge operator must enjoy the same level of trust as the SVP provider.

As future work, we suggest investigating how to integrate and extend the MP and SVP concepts for the deployment of VNFs or Service Function Chains (SFC) [28] trustfully on edge devices of collaborating parties or service providers.

Acknowledgment

The authors would like to thank Lorenzo Keller, Samuel Fricker and Yuliyan Maksimov for support. Furthermore, this work has received partial funding from the European Unions Horizon 2020 research and innovation program 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] T. Llewellyn *et al.*, "BONSEYES: platform for open development of systems of artificial intelligence," in *Proc. ACM Int. Conf. on Computing Frontiers*, Siena, Italy, May 2017.

[2] M. De Prado *et al.*, "Bonseyes AI Pipeline—Bringing AI to You: End-to-End Integration of Data, Algorithms, and Deployment Tools," *ACM Trans. Internet Things*, vol. 1, no. 4, Aug. 2020.

[3] A. Shostack, *Threat Modeling: Designing for Security*. Wiley, 2014.

[4] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, Mar. 2014.

[5] J. Thönes, "Microservices," *IEEE Software*, vol. 32, no. 1, 2015.

[6] V. A. Mehri, D. Ilie, and K. Tutschku, "Privacy and DRM requirements for collaborative development of AI applications," in *Proc. ARES*, Hamburg, Germany, Aug. 2018.

[7] ——, "Designing a secure IoT system architecture from a virtual premise for a collaborative AI lab," in *Proc. DISS*, San Diego, USA, Feb. 2019.

[8] C. Collberg *et al.*, "A taxonomy of obfuscating transformations," Dept. of Comp. Science, Uni. Auckland, NZ, Tech. Rep., Jul. 1997.

[9] B. Parno *et al.*, "Bootstrapping trust in commodity computers," in *Proc. IEEE SSP*, Oakland, CA, USA, Jul. 2010.

[10] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the DoS and DDoS problems," *ACM Computing Surveys*, vol. 39, no. 1, Apr. 2007.

[11] M. Schumacher *et al.*, *Security Patterns: Integrating security and systems engineering*. Wiley, 2013.

[12] M. Singhal *et al.*, "Collaboration in multicloud computing environments: Framework and security issues," *Computer*, vol. 46, no. 2, 2013.

[13] Istio Authors, "Security," 2019. [Online]. Available: https://istio.io/docs/concepts/security/

[14] Microsoft, "Sidecar pattern," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/sidecar

[15] P. A. Grassi and J. L. Fenton, *Digital Identity Guidelines*, Jun. 2017, NIST Special Publication 800-63-3.

[16] R.-V. Tkachuk *et al.*, "Orchestrating future service chains in the next generation of clouds," in *Proc. SNCNW*, Luleå, Sweden, Jun. 2019.

[17] B. Blanchet, "Modeling and verifying security protocols with the applied pi calculus and proverif," *Found. Trends Priv. Secur.*, vol. 1, no. 1–2, Oct. 2016.

[18] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. CRC Press, 2015.

[19] M. Héder, "From NASA to EU: the evolution of the TRL scale in public sector innovation." *Innovation Journal*, vol. 22, no. 2, 2017.

[20] S. Fricker (Edt.), "Validation and Open Developer Community Report - Del. D2.5," H2020 Bonseyes – AI Marketplace, Tech. Rep., Jan. 2020.

[21] X. Gao *et al.*, "Containerleaks: Emerging security threats of information leakages in container clouds," in *2017 47th IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*. IEEE, 2017.

[22] Docker Inc., "Content trust in docker." [Online]. Available: https://docs.docker.com/engine/security/trust/content_trust/

[23] G. Xilouris *et al.*, "T-nova: A marketplace for virtualized network functions," in *Proc. of EuCNC 2014*. IEEE, 2014.

[24] J. Kempf *et al.*, "The Nubo virtual services marketplace," *arXiv preprint arXiv:1909.04934*, 2019.

[25] Ocean Protocol Foundation, "Ocean protocol: A decentralized substrate for ai data and services," Mar. 2019. [Online]. Available: https://oceanprotocol.com/

[26] D. Migdal *et al.*, "Offline trusted device and proxy architecture based on a new TLS switching technique," in *2017 International Workshop on Secure Internet of Things*, 2017.

[27] N. Maroua *et al.*, "A new formal proxy-based approach for secure distributed business process on the cloud," in *IEEE AINA*, 2018.

[28] C. Zhang *et al.*, "L4-l7 service function chaining solution architecture," *Open Networking Foundation, ONF TS-027*, 2015.