An architecture for trusted PaaS cloud computing for personal data

Lorena González-Manzano, Gerd Brost and Matthias Aumueller

Abstract Cloud computing (CC) has gained much popularity. Large amounts of data, many of them personal, are consumed by CC services. Yet, *data security* and, derived from that, privacy are topics that are not satisfyingly covered. Especially usage control and data leakage prevention are open problems. We propose the development of a trusted Platform as a Service CC architecture that addresses selected *Data security* and privacy threats (*Data breaches*, *Insecure interfaces and APIs*, *Malicious insiders* of service providers and *Shared technology vulnerabilities*). Services that consume personal data and are hosted in the proposed architecture are guaranteed to handle these data according to users' requirements. Our proof of concept shows the feasibility of implementing the presented approach.

1 Introduction

CC providers offer services according to the well-known service models Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a service (IaaS) [17]. As the usage of Cloud Computing (CC) progresses, so does the distribution of personal data. As a result, data is managed in a decentralized way by services running on nodes with a particular set of features. This leads to security questions that have to be answered [35]. The possibili-

Lorena González-Manzano
University Carlos III of Madrid (Leganés, Spain), e-mail: lgmanzan@inf.uc3m.es

Gerd Brost
Fraunhofer Research Institution for Applied and Integrated Security, e-mail: gerd.brost@aisec.fraunhofer.de

Matthias Aumueller
Fraunhofer Research Institution for Applied and Integrated Security, e-mail: matthias.aumueller@aisec.fraunhofer.de

ties of having huge storage and computational capabilities are unquestionable benefits of CC. However, the persistent compliance of users' requirements or the liability of delivering data in a trusted service are some of the security problems which have to be solved.

Moreover, the increase of services based on cloud computing, like data storage services or upcoming services like e-government and health care, which use huge amounts of personal data, highlights the need of developing services that care about the security of personal data. In order to protect the user's privacy, data protection has to be in the focus of service- and application development [20]. Though some users are willing to disclose personal data in return for benefits [15], this is not true for all of them and laws and rights highlight the need of privacy preservation.

As an example, users who manage their contacts and appointments with a cloud based application, have to be sure about the trustworthiness of the node where the service is running and the trustworthiness of the service itself.

Likewise, using a health care application where medical history is managed, the trustworthiness of the node and the service that use these data must ensured. Note that according to the Trusted Computing Group "*an entity can be trusted if it always behaves in the expected manner for the intended purpose*" [1].

The scenario that this paper tries to protect against are service providers who offer malicious services, meaning services that do not work as intended by the users and, by that, threatening the security of the user's data. This could happen accidentially by a broken design or intentionally by a malicious service provider.

This can be solved by offering a PaaS service platform which limits functionality to a provided API and by designing this platform with special regard to mechanisms that provide data security. The contribution of this proposal is the development of a trusted PaaS CC architecture.

Users that are uploading personal data to services running in the proposed architecture, can be sure that their data is handled according to their requirements, stated in form of policies. A proof of concept shows the feasibility of implementing the proposal.

The document is structured as follows. The threat model is described in Section 2. Related work is disucssed in Section 3.Section 4 presents the overview and the design of the proposed architecture. In Section 5 a use case is depicted. Section 6 presents the developed proof of concept. In Section 7 we discuss our approach. Finally, conclusions and future work are outlined in Section 8.

2 Threat model

The Cloud Security Alliance defined the top cloud computing threats in [9]. These Threats can be classified under these nine categories:

a) *Data breaches* refers to the acquisition of data by entities which are not compliant with users' expectations.
b) *Data loss* corresponds to the loss of data intentionally, e.g. a malicious data administrator, or accidentally, e.g. an earthquake.
c) *Account or service traffic hijacking* refers to attack methods such as phishing or exploration of software vulnerabilities.
d) *Insecure interfaces and APIs* corresponds to the appropriate design of interfaces or APIs to prevent accidental and malicious attempts to circumvent policies.
e) *Denial of service* bases on using huge quantities of CC resources until leaving services unavailable.
f) *Malicious insiders* refers to the existance of users who are able to manage data in an unauthorized way. However, it must be distinguished between malicious insiders in cloud nodes and in service providers.
g) *Abuse of cloud services* bases on using CC for illicit purposes such as brute forcing decryption keys.
h) *Insufficient due diligence* refers to the ignorance or a lack of understanding of techniques, procedures, responsibilities, etc. related to CC.
i) *Shared technology vulnerabilities* corresponds to shared components that offer insufficient isolation on infrastructure (IaaS), platform (PaaS) and software (SaaS) level.

As stated in the introduction, the scenario that this paper tries to protect against are service providers who offer malicious services, meaning services that do not work as intended by the users and, by that, threatening the security of the user's data.

Regarding this, a), d), f) and i) is the addressed set of threats in this proposal, though highlighting that f) focuses on malicious insiders on the service provider side.

Consequently, attackers outside of the domain of the provider (external attacks, firewall hacking, etc.) and malicious insiders of cloud nodes (physical access to server rack etc.) are out of the scope of the proposal.

3 Related work

CC is a significant area of research where a lot of authors have contributed. In sum, a total of 15 proposals have been analysed, identifying mechanisms applied to address CC threats related to *Data security* (*Data breaches*, *Insecure*

interfaces and APIs, *Malicious insiders* and *Shared technology vulnerabilities*). Table 1 presents a summary of the analysis.

Table 1 Cloud Computing security state of art

Proposals	Data breaches	Insecure interfaces and APIs	Malicious insiders	Shared technology vulnerabilities	Service model	Threat model
[22]				Attribute based encryption to match user requirements concerning node settings	IaaS	Attacker is an agent who has access to the node's management interface.
[16]	Data is cryptographically bundled to access and usage policies. Secure execution environment which tracks data and enforces policies.	Secure execution environment which tracks data and enforces policies.	Secure execution environment which tracks data and enforces policies.	*Node attestation protocol required+ Track data + Access control enforcement	*IaaS/ PaaS	Users that act either: Intentionally, Accidentally, Unknowing.
[6]		Certify programs running		Certify programs running	PaaS	A service provider launches a malicious or faulty application service.
[21]				Node attestation protocol according to a set of configuration settings and also node attestation while migration	IaaS	Insiders that administer the software system pose a serios risk.
[30]	Execution monitor in kernel of service provider platform, client and service commitment protocol.	Service commitment protocol: certify services by vendors		Client and service commitment protocols	PaaS	**Attacker gains root priviledge over the server and trick the user into having requests processed by untrusted code.
[25]		Fine-grained attestation, attest critical pieces of code		Fine-grained attestation, attest critical pieces of code	PaaS	Prevent software attacks
[11]				VM attestation	IaaS	**Malicious devices using hardware.
[8]	Encrypting data in the storage service		Encrypting data in the storage service	VMM registration + nodes attestation through TPM and PCR values	IaaS	**Malicious cloud administrators.
[29]				Attest nodes integrity, regarding users' requirements, when VMs launch, migrate and return results.	IaaS	**Attestation platform is prone to be attacked.
[26]	Certificates to attest applications	Certificates to attest applications		Certificates to attest applications	PaaS	**Storage replay, modification and exhaustion attacks .
[23]				Attest integrity of VMs and hosts. Data remain encrypted until the trustworthiness of the nodes and VM are verified.	IaaS	**Attack the VMs where services run.
[33]	Whole virtual disk encryption		Whole virtual disk encryption	VM attestation, the hypervison guarantees integrity and confidentiality of software running in guest VMs + whole virtual disk encryption	IaaS	Local and remote adversaries in full control of a VM.
[28]				Attesting VMs over a single TPM	IaaS	**Man-in-the-middle attacks.
[4]				Node attestation and VM trustworthiness verification	IaaS	
[19]			Labels within file to define how they should propagate	Labels within file to define how they should propagate	SaaS	User who inadvertently uploads sensitive data to the cloud and then shares or accesses the data in a way that violates company policy.

* Briefly mentioned
** Not explicitly defined

A total of 5 proposals address *Data breaches*. In particular, [16] and [8] apply cryptography to protect data in the storage service, thereby data cannot be delivered to untrusted parties. Moreover, Maniatis et al. propose a tracker module to follow data and persistently control them [16]. However, this contribution presents a conceptual approach without specifying details on the execution and performance of encryption and data tracker procedures. Similarly, Zhang et al. proposes the encryption of the whole virtual disk [33]. By contrast, other works base on applications that are certified by their vendors [26, 30]. Then, an appropriate data management is ensured as long as users trust these vendors.

Concerning *Secure interfaces and APIs*, 5 out of 15 works address this threat. The main mechanism focuses on the use of certificates [6, 30]. Certifying applications or services leads to assuming the right execution of interfaces and the corresponding APIs. Otherwise, Shi et al. look for fine-grained attestation. They propose the attestation of critical pieces of code [25]. Then, if such critical pieces of code refer to interfaces or certain parts of APIs, this threat is addressed to some extent.

Malicious insiders have been addressed by 4 out of 15 proposals, two of them focus on malicious insiders of the SP [19, 16] and the other two on malicious insiders administering the cloud node [8, 33]. Cheng et al. apply cryptography to encrypt data in the storage service [8] and Zang et al. proposes the encryption of the whole disk of a VM [33]. It can be assumed that data is only partially protected with this approach against malicious insiders because data is not protected at runtime. Maniatis et al. proposes the tracking of data to be in persistent knowledge of their location [16]. Similarly, but at software level, Papagiannis et al. work focuses on filtering HTTP connections to manage data propagation [19]. This technique helps to determine if data is controlled by an attacker or not, though requiring the installation of a browser extension.

All proposals address *Shared technology vulnerabilities* to some extent and assorted mechanisms are applied. Specifically, apart from mechanisms that address the remaining set of threats, node attestation [21, 22, 29], virtual machine (VM) attestation [11, 23, 33, 28] and combined attestation [4, 8] are the main ones developed.

Lastly, a great variety of threat models are distinguished, even not being explicitely defined in all proposals (6 out of 15 do not explicitely describe a threat model [11, 8, 29, 26, 23, 28]). For instance, some threat models base on VM attacks [23, 33] and others on software attacks [25, 21, 6].

In the light of this analysis, a lot of publications have gone towards the development of trusted CC. Nonetheless, none of these proposals addresses all threats related to *Data security* (note that [16] is a conceptual approach where only a high level description is provided).

Several mechanisms concerning the prevention of data leakage have already been developed. First of all, some techniques focus on automatically identifying personal data, such as expression-pattern-matching algorithms to

identify strings like credit cards [5]. By contrast, other sets of mechanisms search data leakages knowing personal data beforehand and using them as input. The most common technique is taint tracking, where a variable is tainted when accepting user data and all instructions that make use of them are also tainted [34]. For instance, many approaches are related to data taint analysis of smartphone applications, Android applications in particular [10]. Besides, from a different perspective, [7] presents an algorithm that instruments untrusted C programs regarding places where policy violations may exist.

Users want to be in persistent control of their data, thereby requiring the enforcement of access control policies along the whole usage process. To address this issue the concept of sticky policies comes into play. It means that policies have to be always attached to the data they apply to [13].

Multiple mechanisms in different contexts have been proposed to address this matter. In general, two different groups can be distinguished. The first group corresponds to the application of Attribute or Identity Based encryption where policies are embedded in keys or ciphertexts which helps to control access to data [13, 18]. A second group of proposals focus on verifying access control policies at runtime, requiring the installation of a small piece of software, e.g. a plug-in, at the client-side [12, 3].

Attestation of an entity is a proof of specific properties of that entity [1]. It is usually related to digital signatures and to software and hardware integrity.

Assorted attestation protocols have been proposed to attest nodes, virtual machines (VM) or nodes and VMs simultaneously. Santos et al. apply Attribute-Based Encryption to encrypt data left on a node, as long as the node satisfies a set of attributes [22]. Another approach based on cryptography is proposed by Garfinkel et al. [11]. It attests the integrity of VMs and hosts, encrypting data satisfying user's requirements. Similarly, though avoiding the use of cryptography, other works base on attesting nodes integrity regarding user conditions [29] and on attesting nodes according to a set of configuration settings as well as on attesting nodes during migration [21]. Velten et al. present a node attestation procedure to attest an arbitrary amount of VMs using a single TPM [28].

4 An architecture for trusted PaaS cloud computing for personal data

This Section presents the overview (Section 4.1) and the design (Section 4.2) of the proposed trusted PaaS CC architecture.

4.1 System overview

Trusted PaaS for personal data bases on creating services implemented through a provided API. These services are inspected to avoid data leakage and are running on attested nodes that guarantee the enforcement of access control along the whole process of data usage. To address this issue the proposed architecture consists of the following objects and entities (see Figure 1):

- *User* uses services and uploads personal data to services. Users have a key pair (K_{pub_u} and K_{pv_u}) used to sign delivered personal data and access control policies.
- *Personal data store (PDS)* is a storage of personal data controlled by data owners. For instance, it can correspond to a physical device like a smartcard within a pen drive, a mobile phone [2] or a cloud storage service [14]. Specifically, the PDS contains personal data of the users, access control policies related to each used service and data and user's key pairs.
- *Services* are pieces of software created through the use of a provided API.
- *Service provider (SP)* creates services and sends them to the Leakage Inspector to guarantee their trustworthiness and appropriate enforcement of access control policies. Once a service is successfully analysed and signed, the SP uploads it to the cloud node to make it available to users.

- The *Leakage inspector (LI)* certifies that a particular service does not contain data leakages, that is, any kind of personal data leaves the service without the data owner's consent. Moreover, the LI instruments the services' code at runtime, so the SPA can guarantee a persistent enforcement of access control policies. This entity has a pair of keys ($K_{pub_{LI}}$ and $K_{pv_{LI}}$) to sign services after inspection and instrumentation. The LI is attested by the node when running a signed service.
- *Sticky-policy analyzer (SPA)* guarantees, in each service, the persistent enforcement of access control policies. The SPA is attested by the node before being called for the first time.
  Note that for the sake of simplicity just a single SPA and LI are mentioned but many of them are assumed to be managed.
- *Cloud node* is a particular PC/server within the cloud computing environment that offers the appropriate infrastructure to execute services. Besides, it includes the management of the provided API. Its trustworthiness has to be attested by users before delivering data to running services.

Fig. 1 Architecture

In general, SPs implement services using a provided API. Each service is uploaded to the LI that inspects, instruments and signs the code if the inspection has been successful. Afterwards, the SP uploads the service to the cloud node, making it available to users. Users who want to use a cloud service attest the node where the service is running to confirm it being in the expected, trustworthy configuration.

This guarantees the node is only running signed services, that it is connected to a trustworthy LI and SPA and that it delivers valid service signatures. Checking the services signature ensures it has been inspected by the LI. If the verifications are successful, the service is used and personal data is delivered when required. While the service is running, access control policies are verified by the SPA, so access control enforcement is performed along the whole service usage guaranteeing no data leaves the system if not allowed by an attached policy. The enforcement process requires the SPA to verify signed policies and signed data, retrieved from users' PDSs, to ensure that both type of elements belong to the same user.

Note that signatures refer to digital signatures where the public keys are included in certificates issued by certification authorities. Thus, a public key infrastructure is assumed to exist.

4.2 System design

This section presents the description of applied mechanisms. In particular, the provided API is briefly introduced (Section 4.2.1), applied access control policies are defined (Section 4.2.2), the functionality of the LI is described (Section 4.2.3), the execution of the SPA is defined (Section 4.2.4) and the applied attestation protocols are presented (Section 4.2.5).

4.2.1 Proposed API

Assuming that the development of a complete API is a matter of future work, this section presents the first steps towards the development of the final API. It consists of nine methods implemented in Java. This API must be used by SPs that want to take advantage of the proposed architecture.

- *Object receiveFromPDS(String PDSLocation, String dataType)* obtains a list that contains the requested object and the set of policies attached to the object. The data is retrieved from a PDS located in a particular place.
- *Object sendToEntity(Entity entity, Object data)* sends data to a particular entity.
- *Object receiveFromEntity(Entity entity)* retrieves data from a particular entity.

- *Boolean dataStoreDB(Object DB, Object table, Object data)* stores data in a particular database and a particular table.
- *Object dataRetrieveDB(Object DB, Object table, Object cond, Object request)* retrieves requested data from a particular database and a particular table given a set of conditions.
- *Object searchDB(Object DB, Object table, Object cond)* searches data in a particular database and a particular table given a set of conditions.
- *Boolean writeFile(Object data, File file)* writes data to a given file.
- *Object readFile(File file)* reads a given file.
- *Object retrieveEnvironmentalAtt(LinkedList<LinkedList<Data, Policy> > policies)* retrieves the necessary environmental attributes to evaluate established policies.

It should be noted that the analysis performed by the LI focuses on methods *writeFile* and *sendToEntity*. On the one hand, the LI inspects the use of *writeFile* because it causes data leakages when the written data is sensitive. On the other hand, it instruments the code according to the use of *sendToEntity* because this method may send personal data to entities without the consent of users.

4.2.2 Access control policies description

Access control policies (ρ) for CC services can be complex [32, 27, 31]. Considering that the search of fine-grained policies is a desirable feature and attribute management facilitates this issue [31], proposed access control policies are constructed under the Attribute Based Access Control Model. This model manages access control evaluation rules against data attributes, entities attributes (either a subject, a group or a general entity) and environmental attributes [32]. In this respect, proposed access control policies are composed of the type of data to which they apply (TD), entity conditions to satisfy (EntC), environmental conditions to satisfy (EnvC) and the granted right (GR). Applying BNF notation [24], proposed policies are depicted as follows:

- $\rho ::= \{TD,\ EntC,\ EnvC,\ GR\}$
  - $TD ::= DataType|DataType.att_i\{\wedge|\vee\ DataType|DataType.att_i\}^*$
    - $DataType ::= Medical|Contact|Hobbies|Multimedia|...$ refers to the type of data under which personal data are classified.
    - $DataType.att_i ::= Medical.medicalHistory|Medical.lastPrescription|Contact.name|...$ corresponds to an attribute of a particular data type.

    For example, $Medical \wedge Contact.name \wedge Contact.surname \wedge Contact.address$ expresses that the policy is linked to *medical data* and to users' name, surname and address which are considered *contact data*.
  - $EntC ::= EntityType|(EntityType.att_i\ FunctionalOpe\ [\neg]EntityType.att_iValue)\ \{\wedge|\vee EntityType|(EntityType.att_i\ FunctionalOpe\ [\neg]EntityType.att_iValue)\}^*$.

  - $EntityType ::= Receptionist|Doctor|Ministry|...$ refers to the type of the entity that may use personal data.
  - $EntityType.att_i ::= Receptionist.age|Doctor.yearsExperience|...$ corresponds to attributes of a particular entity type.

  For instance, $Doctor.yearsExperience > 10 \wedge Doctor.age > 40$ expresses that the subject who uses a particular data has to be a doctor older than 40 and with more than 10 years of experience.
- $EnvC ::= EnvAtt_i\ FunctionalOpe\ [\neg]EnvAtt_iValue\{\wedge|\vee\ Env.att_i\ FunctionalOpe\ [\neg]Env.att_iValue\}^*$ refers to applied environmental restrictions, that is attributes.
  For instance, $ExpirationDate > 30/05/2012 \wedge ExpirationTimeDate < 30/05/2013$ expresses that the policy is valid along a year, from 30/05/2012 to 30/05/2013. Note that $\emptyset$ implies that restrictions do not exist.
- $GR := Read|Write|Download|Print|...$ corresponds to the set of operations that can be performed over data.

Operators are distinguished between the following pair of types. Note that brackets ("()") can be applied to manage precedence in the order of attributes.

- $LogicOpe ::= \wedge|\vee|\neg$ where $\wedge$ means a logic conjunction, $\vee$ a logic disjunction and $\neg$ means negation. The first pair can be applied to two elements and the last one can be applied to particular attribute values.
- $FunctionalOpe ::= < \mid > \mid \leqslant \mid \geqslant \mid =$ refers to operators applied over two attributes. For example, given the attribute $YearB$, the birthday year of a particular user, $User1$, can be compared by building the expression $YearB(User1) < 2013$ to identify if $User1$ was born before 2013.

Access control policies are established by users, per each service, and located in Personal Datastores. Alleviating the burden of defining policies per each piece of data, policies are linked to data types. For instance, a policy can be simultaneously linked to *medical data* (medical history, previous prescriptions, etc.) and *contact data* (name, surname, phone, etc) instead of defining as many policies as pieces of personal data. However, a more detailed definition of access control policies, including the definition of existing data types, entities and attributes, as well as the management procedure, is a matter of future work. Analogously, the simplification of access control policies establishment is another open issue.

4.2.3 Data leakage inspection

Despite the variety of existing algorithms (see Section 3), this paper contributes defining a general mechanism, based on [7], to identify personal data leakages in CC services.

It should be noted that services are implemented using a provided API and operations that cause data leakages as well as operations that require access control enforcement are known in advance. Under these assumptions,

the proposed mechanism applies a Data Leakage Inspector (LI) to inspect, instrument and sign the service code after a successful inspection. In particular, once SPs send services to the LI, the process is split into:

A. Data leakage inspection:

1. All data received from a PDS are considered personal data.
2. Personal data are tracked along the whole service:
   a. If they are used within an allowed operation, the analysis continues.
   b. If they are used within a disallowed operation, the analysis aborts and the SP is notified.

B. Instrumentation regarding access control enforcement:

1. Places where access control enforcement has to be performed are marked.
2. A call to the SPA is inserted in each place previously noticed. In general, the SPA is called through the method *sendToSPA*:
   Boolean sendToSPA(LinkedList<Data> data, LinkedList<Object> envAtt, String right, Entity entity, LinkedList<LinkedList<Data,Policy> > dataPolicies) makes a call to the SPA to verify policies at runtime. Specifically, *data* refers to the request data which is signed by data owners, *envAtt* is the set of existing environmental conditions, *right* is the right to execute, *entity* is the entity which has the granted right over data and *dataPolicies* refers to signed data and access control policies to verify.
3. Once the inspection is successful and instrumentation is finished, the LI signs the service's code and hands it back to the SP.

4.2.4 Sticky policies analysis

For the persistent control of data, policies always have to follow the data to which they are applied to enable later enforcement. In this regard, this paper presents a general mechanism to address sticky policies. In contrast to existing approaches (see Section 3), the application of cryptography and the installation of software at client-side is avoided. The mechanism applies a Sticky Policy Analyzer (SPA) in charge of verifying access control policies at runtime. Given calls introduced in the services' code by the LI, the SPA is able to evaluate access control policies at runtime. The process consists of:

1. Signed data and policies are verified to ensure that both types of elements refer to the same user.
2. Policies are verified and enforced.

4.2.5 Node attestation

In order to be able to trust a certain service and therefore being willing to provide sensitive data to it, users must be assured that their sensitive information is processed in a way they are comfortable with. To achieve that level of assurance, the node as well as the limiting and enforcing security components (LI, SPA) are attested by different attestation protocols: the proposed node attestation protocol ensures that the node is free from malicious system files, faulty configuration files or backdoors and that it has not been booted into a system state that is considered insecure; the SPA attestation protocol certifies that the SPA is running, working correctly and that is has not been altered and therefore that the policies of the user are enforced and not bypassed. The LI attestation protocol ensures the correct signing and inspection process and also ensures the service provider that its source code that is revealed to the LI is inspected with regard to confidentiality. The node attestation protocol verifies the service container is in a trusted configuration and only runs signed code.

The problem of configuration management (e.g. how to verify the trustworthiness of a node after a software update) is not considered here and is a matter of future work, as well as the large trusted computing base (meaning a whole system running software componets is the trusted computing base).

Two scenarios would be possible: Attesting a node configuration which incorporates SPA and LI, or placing each component on a single node and attesting it then. Due to flexibility (e.g. having multiple SPAs and LIs) we chose the second approach and will illustrate it here:

Fig. 2 Attestation flow

1. When a service is instantiated on the cloud node and signed code is about to be run, the node will attest the PCR values of the LI bundled with the signed source code. By verifying the signature with the attestation identity key of the LI, it ensures it was in a trusted configuration when signing the code. Thus, the LI is indirectly attested.
2. Before the user uploads personal data to a node, she can verify the configuration by attesting the node. This makes sure only "trusted" nodes receive personal data.
3. Before calling a SPA for the first time, the node attests the configuration of the SPA to make sure it works in the expected manner (thus, not simply ignoring policies etc.).
   However, self-attestation does not guarantee freshness. So an attacker could attest the LI to get a trusted configuration, manipulate the LI so it performs arbitrary signing operations and thus obtain malicious service code bundled with a trusted PCR value. To avoid this, the node running

the service would need to re-attest the associated LI before running it and trigger a re-signing process if fingerprints do not match.

Fig. 3 Attestation protocol

We use a very simple attestation protocol close to [22] and use the same notation. To attest the node the user first retreives the public attestation identity key AIK^P_{node} and sends a nonce. The node answers with a quote that contains this nonce and its current PCR values that are stored inside the node's TPM. This quote is signed by the node's AIK to ensure the integrity of the response. After the receipt of the payload from the node the user can compare the nonce to check the freshness of the attestation and, if this matches the expected value, compare the received PCR values with a library of trusted node configurations. This protocol is used for attesting cloud node and SPA.

5 Use case: health care service

The proposed use case shows a realistic scenario, regarding a Health Care Service, where the presented trusted computing architecture is applied. The proposed service functionalities, implemented making use of the API presented in section 4.2.1, are described (section 5.1). After that, the definition of entities and attributes applied in access control policies associated with this particular service are briefly presented, as well as an example of applicable policies (section 5.2). Concluding the section, the process of sending the service's code to a LI is defined (section 5.3).

5.1 Service description

A Health Care Service, developed in Java, offers users the possibility of requesting medical appointments and taking part in online consultations.

Figure 4(a) depicts the method which allows users to request medical appointments. A retrieval of the user's name, surname and address from the user's PDS is performed. If the user is not found, medical history data is requested and her name, surname, address and medical history are stored in the DB, registering the user. After that, the type of the appointment (that is the type of the consulting room) and the desirable date and time for the appointment are sent to a receptionist (a subject in charge of managing ap-

pointments) who processes the request. When the receptionist responses, the appointment is returned to the user.

Figure 4(b) depicts the method that allows users to take part in online consultations. Again, it is verified if the user has previously used the service and if not, she is registered. Subsequently, the request type, that is the type of doctor to whom the request should be forwarded, is identified and the request is sent accordingly. Finally, when the right doctor responses, the answer is forwarded to the user. Note that, for simplicity reasons, this service can only make requests to general medicine doctors and to allergists.

5.2 Service access control policies

The proposed Health Care Service offers services related to medical issues and entities involved refer to doctors, doctor assistants, nurses and receptionists. Besides, entity attributes such as age or years of experience are also involved. Personal data, as the user's name and address, as well as the medical history data, which refers to *medical data* is the data that is consumed. Environmental attributes such as the expiration time of a policy can be also applied.

Under the mentioned set of entities, personal data and environmental attributes, manyfold policies can be defined. However, for the sake of clarity, the following set of policies has been created as an example. Note that environmental conditions are not applied.

- *Policy1-Medical-1(medical, doctor.yearsExperience>10, ∅, read).* It means that personal data categorized as *medical data*, can only be read by subjects who are doctors with more than 10 years of experience.
- *Policy2-Medical-1(contact.address, receptionist, ∅, read).* It means that the user's address, which is categorized as *contact data*, can only be read by subjects who are receptionists.
- *Policy3-Medical-1(contact.name AND contact.surname, receptionist OR doctor, ∅, read).* It means that the user's name and surname, which are categorized as *contact data*, can only be read by subjects who are receptionists or doctors.

5.3 Service inspection

Before uploading a service to a cloud node the Health Care Service, composed of the methods *requestMedicalAppointment* and *requestMedicalInfoOnline*, is sent to the LI. Recalling that *writeFile* may apply personal data, which refer to data retrieved from a PDS, and it may cause data leakages sending personal

data out of the service, its application is inspected. Thus, in case *writeFile* applies personal data, data leakages are found and the LI notifies the SP of all identified ones. Afterwards, the SP has to remove all data leakages and sends the code back again to the LI. Otherwise, the LI instruments the code. It tracks personal data and inserts calls before the use of *sendToEntity* in case personal data are used. These calls are introduced regarding *sendToEntity* because this method may deliver personal data to external entities. Figures 5(a) and 5(b) present the result of executing both processes, though considering that data leakages should be avoided before the code's instrumentation is performed. Finally, when the inspection and instrumentation finishes, the service's code is signed and sent to the SP.

6 Proof of concept

A proof of concept has been developed, using J2SE 1.7, to prove the feasibility of implementing the proposal. A pair of web services has been developed. One of them involves the basic implementation of methods *requestMedicalAppointment* and *requestMedicalInfoOnline* presented in the proposed use case (section 5). Basic means that the functionality of processing requests, either medical appointments or online consultations, is not currently implemented but the execution of inspected and instrumented code is proven to be successful.

On the other hand, the other implemented web service corresponds to the LI. It allows the inspection of java files pointing out statements that cause data leakages, and the instrumentation of java files to ensure an appropriate access control enforcement.

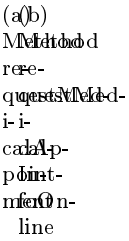

Fig. 4 Health Care Service's code

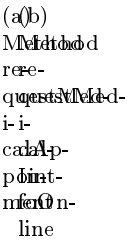

Fig. 5 Processed Health Care Service's code

7 Evaluation of our approach

The proposed approach focuses on the development of a trusted PaaS CC architecture which is supposed to address all noticed *Data security* threats (*Data breaches*, *Insecure interfaces and APIs* , *Malicious insiders* and *Shared technology vulnerabilities*). We will briefly assess if these threats have been covered.

Data breaches are addressed by the use of the LI. The service code is inspected to detect places where personal data is sent to external elements (e.g. written to a file), hindering data breaches.

Insecure interfaces and APIs is addressed by the development of an API used to implement running services, as well as the inspection of data leakages. An API is provided to facilitate the identification of possible places where users' privacy may be violated. Where malicious uses of the provided API can be performed, the LI analyses the code to detect disallowed statements that cause data leakages.

Malicious insiders is another addressed threat, though only malicious insiders in the service provider have been considered. The verification of users' policies at runtime avoids the disclosure of data against the consent of the user, thereby mitigating the existence of malicious insiders. Besides, the fact that users verify the LI signature attached to the used service helps to assure the lack of malicious alterations in the service's code. The enforcement of API usage also limits the possibility to run malicious code.

Last but not least, *Shared technology vulnerabilities* is mainly addressed through the development of an API to implement services. This leads to a limited set of functionalities and avoids shared technology vulnerabilities, since services cannot run arbitrary commands and do not have direct access to (virtualized) ressources.

Following this, we will briefly discuss strenghts and weaknesses of our approach:

On the one hand, the presented proposal has several weaknesses. It requires a public key infrastructure which brings complexity into the system. Also the range of services that could use our architecture is limited: Services like

social networks, in which a huge amount of personal data is processed and accessible to many entities, would require a lot of access control policies verifications causing a large number of calls to the SPA, thus reducing the service performance or even worse, provoking a denial of service in the SPA. Furthermore, the development of a complete API to allow the implementation of any type of service is difficult and successfully securing the code gets complex. Likewise, it should be noticed that existing services would have to be re-implemented if they want to be executed in the proposed trusted CC architecture. Attestation is currently feasible for a specific configuration, but is getting somehow difficult with several supported configurations, even made worse by having multiple components that need to be handled. This would lead to a trusted repository which would have to be managed as well. Defining independent components for SPA, Node and LI may lead to performance penalties (in regard to response times) and online/offline situations must be handled.

Nonetheless, the proposed architecture has multiple strengths. Limiting functionality to a provided API, a common mechanism in PaaS solutions like Google AppEngine, makes the detection of malicious code easier. The usage of components like the LI and SPA enable parallel or cascading designs to limit performance problems (in regard of traffic load) and enables segregation of duties. Concerning sticky policies, the proposed mechanism bases on instrumenting the code before being executed. Access control management is performed at the server side instead of applying more cumbersome mechanisms such as plug-ins or cryptographic procedures. Another advantage is that the development of an API simplifies services' code instrumentation and analysis because methods which may violate users privacy are known beforehand. Attestation ensures the trustworthiness of used components and guarantees policy enforcement and the integrity of the whole system. This even guarantees the SP that his code is not leaked to the outside world.

8 Conclusions and future work

CC services are used quite commonly and make use of large amounts of data, many of it personal. In this regard, *Data security* is of vital concern. Personal data has to be managed according to users requirements and free from privacy violations. To address the discussed threats, this paper presented a PaaS architecture concerning the development of attached mechanisms. In particular, it bases on inspecting and instrumenting services before uploading them to a cloud node to ensure an appropriate access control enforcement at runtime. Moreover, it also ensures the trustworthiness of the node the services are deployed to. Last but not least, a security analysis attests the coverage of all threats and a prototype shows the feasibility of the proposal implementation.

Concerning future work, the proof-of-concept prototype will be developed into a fully working showcase. This is accompanied by the specification by a full-fledged service API suitable for real world service types. The definition of allowed and disallowed calls that are evaluated when inspecting code will be formalized to work with a greater set of use cases. The integration with different platforms and/or protocols has to be studied. A detailed definition of access control policies, including management procedures and techniques to simplify their creation are also matters of future work. This is also true for configuration management of the software components and a more property based attestation workflow with a reduced trusted computing base.

References

1. Achemlal, M., Gharout, S., Gaber, C.: Trusted platform module as an enabler for security in cloud computing. In: Network and Information Systems Security (SAR-SSI), 2011 Conference on, pp. 1–6. IEEE (2011)
2. Allard, T., Anciaux, N., Bouganim, L., Guo, Y., al. et: Secure personal data servers: a vision paper. Proceedings of the VLDB Endowment **3**(1-2), 25–35 (2010)
3. Beato, F., Kohlweiss, M., Wouters, K.: Scramble! your social network data. In: Privacy Enhancing Technologies, pp. 211–225. Springer (2011)
4. Bertholon, B., Varrette, S., Bouvry, P.: Certicloud: a novel tpm-based approach to ensure cloud iaas security. In: Cloud Computing (CLOUD), 2011 IEEE International Conference on, pp. 121–130. IEEE (2011)
5. Brodie, B.C., Taylor, D.E., Cytron, R.K.: A scalable architecture for high-throughput regular-expression pattern matching. In: ACM SIGARCH Computer Architecture News, vol. 34, pp. 191–202. IEEE Computer Society (2006)
6. Brown, A., Chase, J.S.: Trusted platform-as-a-service: a foundation for trustworthy cloud-hosted applications. In: Proceedings of the 3rd ACM workshop on Cloud computing security workshop, pp. 15–20. ACM (2011)
7. Chang, W., Streiff, B., Lin, C.: Efficient and extensible security enforcement using dynamic data flow analysis. In: Proceedings of the 15th ACM conference on Computer and communications security, pp. 39–50. ACM (2008)
8. Cheng, G., Ohoussou, A.: Sealed storage for trusted cloud computing. In: Computer Design and Applications (ICCDA), 2010 International Conference on, vol. 5, pp. V5–335. IEEE (2010)
9. Cloud_Computer_Alliance: The notorious nine cloud computing top threats in 2013 (2013)
10. Fritz, C.: Flowdroid: A precise and scalable data flow analysis for android. Master's thesis, Technische universitat Darmstadt (2013)
11. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: A virtual machine-based platform for trusted computing. In: ACM SIGOPS Operating Systems Review, vol. 37, pp. 193–206. ACM (2003)
12. Ghorbel, M., Aghasaryan, A., Betgé-Brezetz, S., Dupont, M., Kamga, G., Piekarec, S.: Privacy data envelope: Concept and implementation. In: Privacy, Security and Trust (PST), 2011 Ninth Annual International Conference on, pp. 55–62. IEEE (2011)
13. González-Manzano, L., González-Tablas, A., de Fuentes, J., Ribagorda, A.: Security and Privacy Preserving in Social Networks, chap. User-Managed Access Control in Web Based Social Networks. Springer (2013)

14. Kirkham, T., Winfield, S., Ravet, S., Kellomaki, S.: A personal data store for an internet of subjects. In: Information Society (i-Society), 2011 International Conference on, pp. 92–97. IEEE (2011)
15. Li, H., Sarathy, R., Xu, H.: Understanding situational online information disclosure as a privacy calculus. Journal of Computer Information Systems **51**(1), 62 (2010)
16. Maniatis, P., Akhawe, D., Fall, K., Shi, E., McCamant, S., Song, D.: Do you know where your data are? secure data capsules for deployable data protection. In: Proc. 13th Usenix Conf. Hot Topics in Operating Systems (2011)
17. Mell, P., Grance, T.: The nist definition of cloud computing (draft). NIST special publication **800**(145), 7 (2011)
18. Mont, M.C., Pearson, S., Bramhall, P.: Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. In: Database and Expert Systems Applications, 2003. Proceedings. 14th International Workshop on, pp. 377–382. IEEE (2003)
19. Papagiannis, I., Pietzuch, P.: Cloudfilter: practical control of sensitive data propagation to the cloud. In: Proceedings of the 2012 ACM Workshop on Cloud computing security workshop, pp. 97–102. ACM (2012)
20. Pearson, S.: Taking account of privacy when designing cloud computing services. In: Software Engineering Challenges of Cloud Computing, 2009. CLOUD'09. ICSE Workshop on, pp. 44–52. IEEE (2009)
21. Santos, N., Gummadi, K.P., Rodrigues, R.: Towards trusted cloud computing. In: Proceedings of the 2009 conference on Hot topics in cloud computing, pp. 3–3 (2009)
22. Santos, N., Rodrigues, R., Gummadi, K.P., Saroiu, S.: Policy-sealed data: A new abstraction for building trusted cloud services. In: Usenix Security (2012)
23. Schiffman, J., Moyer, T., Vijayakumar, H., Jaeger, T., McDaniel, P.: Seeding clouds with trust anchors. In: Proceedings of the 2010 ACM workshop on Cloud computing security workshop, pp. 43–46. ACM (2010)
24. Scowen, R.S.: Extended bnf-a generic base standard. Tech. rep., Technical report, ISO/IEC 14977. http://www. cl. cam. ac. uk/mgk25/iso-14977. pdf (1998)
25. Shi, E., Perrig, A., Van Doorn, L.: Bind: A fine-grained attestation service for secure distributed systems. In: Security and Privacy, 2005 IEEE Symposium on, pp. 154–168. IEEE (2005)
26. Sirer, E.G., de Bruijn, W., Reynolds, P., Shieh, A., Walsh, K., Williams, D., Schneider, F.B.: Logical attestation: an authorization architecture for trustworthy computing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 249–264. ACM (2011)
27. Takabi, H., Joshi, J.B.: Semantic–based policy management for cloud computing environments. International Journal of Cloud Computing **1**(2), 119–144 (2012)
28. Velten, M., Stumpf, F.: Secure and privacy-aware multiplexing of hardware-protected tpm integrity measurements among virtual machines. In: Information Security and Cryptology–ICISC 2012, pp. 324–336. Springer (2013)
29. Xin, S., Zhao, Y., Li, Y.: Property-based remote attestation oriented to cloud computing. In: Computational Intelligence and Security (CIS), 2011 Seventh International Conference on, pp. 1028–1032. IEEE (2011)
30. Xu, G., Borcea, C., Iftode, L.: Satem: Trusted service code execution across transactions. In: Reliable Distributed Systems, 2006. SRDS'06. 25th IEEE Symposium on, pp. 321–336. IEEE (2006)
31. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: INFOCOM, 2010 Proceedings IEEE, pp. 1–9. IEEE (2010)
32. Yuan, E., Tong, J.: Attributed based access control (abac) for web services. In: Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. IEEE (2005)

33. Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 203–216. ACM (2011)
34. Zhu, D.Y., Jung, J., Song, D., Kohno, T., Wetherall, D.: Tainteraser: protecting sensitive data leaks using application-level taint tracking. ACM SIGOPS Operating Systems Review **45**(1), 142–154 (2011)
35. Zissis, D., Lekkas, D.: Addressing cloud computing security issues. Future Generation Computer Systems **28**(3), 583–592 (2012)