5-2019

EPICS: A Framework for Enforcing Security Policies in Composite Web Services

Rohit Ranchal
IBM Watson Health

Bharat Bhargava
Purdue University

Pelin Angin
Purdue University

Lotfi ben Othmane
Iowa State University, othmanel@iastate.edu

EPICS: A Framework for Enforcing Security Policies in Composite Web Services

Abstract
With advances in cloud computing and the emergence of service marketplaces, the popularity of composite services marks a paradigm shift from single-domain monolithic systems to cross-domain distributed services, which raises important privacy and security concerns. Access control becomes a challenge in such systems because authentication, authorization and data disclosure may take place across endpoints that are not known to clients. The clients lack options for specifying policies to control the sharing of their data and have to rely on service providers which provide limited selection of security and privacy preferences. This lack of awareness and loss of control over data sharing increases threats to a client's data and diminishes trust in these systems.

Keywords
Cloud computing, composite web services, active bundles, security, privacy, access control

Disciplines
Information Security | Systems and Communications

EPICS: A Framework for Enforcing Security Policies in Composite Web Services

Rohit Ranchal, Bharat Bhargava, Pelin Angin, Lotfi ben Othmane

Abstract—With advances in cloud computing and the emergence of service marketplaces, the popularity of composite services marks a paradigm shift from single-domain monolithic systems to cross-domain distributed services, which raises important privacy and security concerns. Authorized data disclosure and access control become a challenge in such systems because authentication, authorization and data disclosure may take place across endpoints that are not known to clients. The clients lack options for specifying policies to control the sharing of their data and rely on service providers which provide limited security and privacy preferences. This loss of control and lack of awareness increases threats to client's data and diminishes trust in these systems.
We propose EPICS, an efficient and effective solution for enforcing security policies in composite Web services that protects data privacy throughout the service interaction lifecycle. The solution ensures that the data are distributed along with the client policies that dictate data access and an execution monitor that controls data disclosure. It empowers data owners with control of data disclosure decisions outside their trust domains and reduces the risk of unauthorized access. The paper presents the design, implementation, and evaluation of the EPICS framework.

Index Terms—Cloud computing, composite web services, active bundles, security, privacy, access control.

1 Introduction

Composite Web services are an extension of the traditional Service Oriented Architecture (SOA) model, which has evolved with the proliferation of cloud-hosted solutions, mobile apps, and Application Programming Interface (API)-centric services. Composite Web services integrate disparate, distributed, and self-sufficient services through a loose coupling to achieve a larger system with more sophisticated functionality [1]. This model enables the dynamic composition of service orchestrations: self-contained, loosely-coupled, and dynamically composed solutions to address the business requirements. It allows the services to be independently developed and transparently deployed while maintaining stable API [2]. This allows concurrent use of services from different ownership domains across various composite solutions without changes to existing services.

Services computing is the backbone of many different types of information systems such as online retail sites like Amazon and eBay, enterprise business-to-business systems [3], pervasive healthcare systems [4], etc. Notwithstanding the many advantages of services computing, ensuring proper enforcement of access control policies and preventing unwanted data leakage in composite Web services is a challenge due to:

- Inability of the client on the selection of services in an orchestration
- Vulnerabilities caused by improper implementation of access control in Web services
- Insufficient options for the client to specify their access control policies
- Improper communication of the client's access control policies by the services in an orchestration

Existing access control mechanisms for Web services limit clients to high-level policies, generally specified as a list of security and privacy preferences. These preferences and their options are selected by the service providers and do not allow clients with fine-grained control over the disclosure of their data, such as associating different policies to specific data items or changing access control behavior based on the operational context. Even when the client is able to specify fine-grain access control policies, existing services infrastructures do not guarantee enforcement and propagation of policies by the recipient services, which may simply ignore the policies.

Despite the importance of proper access control in the online world of growing security concerns, and an everyday increasing number of regulations for access control to sensitive data, Web services and applications still do not meet the expected standards to mitigate data leakage problems. According to the 2013 OWASP ranking of Web application security risks [5], four out of the top ten risks are related to incorrectly implemented access control checks. Information leakage is the second most prevalent vulnerability in Web applications, based on the 2014 Website Security Statistics Report [6].

1.1 How do Data Leak in Service Interactions?

While Web services have been widely used by enterprises since their inception, and the rise of cloud computing in the past decade has given increased traction to online storage and processing of huge amounts of data in many different domains, data leakage attacks are still strong deterrents for wider adoption of cloud and Web services. Examples of recent massive data leaks include the Target Data Breach [7] and Anthem Data Breach [8], where attackers were able to get access to sensitive user information such as credit cards, mailing addresses, email addresses, phone numbers, date of births, medical IDs, social security numbers and employment information. Recent research has demonstrated that many popular Web applications have semantic bugs in their access control implementation, resulting in unauthorized disclosure of data [9].

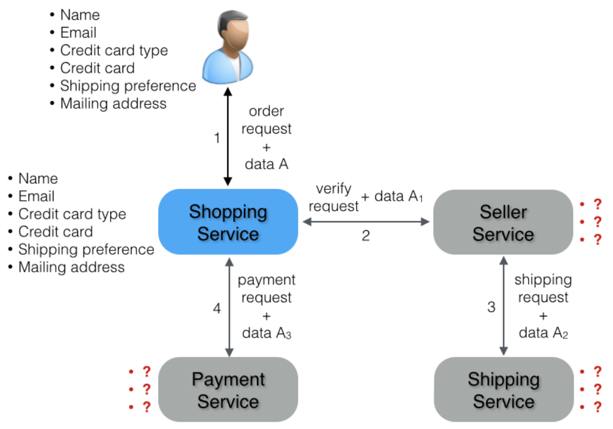


Fig. 1: Composite Web service for online shopping.

Consider a composite Web service for online shopping as shown in Figure 1. The user initially registers with the shopping service and must disclose sensitive information including name, email, credit card, mailing address, billing address, phone number etc., while creating an account. Next, the user sends an order request to the shopping service, which communicates with the seller service to verify the order (item availability in the specified quantities, sizes, colors). The seller service then shares the information with the shipping service to verify shipping eligibility. On verification from the seller service, the shopping service applies the tax, shipping charges and calculates the total amount due for the order. User information is sent to the respective payment service for verification. On payment approval, the order is completed and the client and the seller service are informed. In fact, the shopping service does not need all the data of the user in order to provide its service. For instance, the service can charge the user's credit card for an order without knowing the credit card details. This kind of scenario is prone to data leakage mainly in following ways:

- Primary services often justify the collection of all client data by arguing that the information they store is encrypted, but if the service (shopping here) is compromised user information could be leaked to malicious parties.
- The user data is shared with the seller service, payment service and shipping service but these interactions are not visible to or approved by the user, i.e. the data shared with each of these services may not include only the data they require to provide service, but additional data items as well. Normally, the shipping service only needs the address of the user, the payment service only needs payment credentials, and the shopping service only needs to authenticate the user.
- If the service storing the data of the client does not implement proper access control mechanisms, the private data of one user could be disclosed to other users.

1.2 Threat Model

In order to explain the threat model, we first define the adversary. An adversary is a service that receives data and the associated policies using an existing policy communication standard such as WS-Policy [10], but ignores the enforcement of policies. The services are generally trusted because a client would not want to use untrusted services to share its data with. However, these services can be interested in knowing more information than they are authorized to receive, but they would not generally attack client's protected data in order to discover this information. So they are only capable of making queries and receiving data if they are authorized. For instance, services store client information to provide value-added services, analyze client data to fine-tune their marketing strategies, etc.

1.3 Contributions & Paper Organization

In this paper, we propose EPICS, a framework for enforcing security policies in composite Web services. The framework is based on active data entities that are bundled with access control policies and a mechanism for ensuring proper policy enforcement in external service domains. The following are examples of policies that can be enforced with the proposed framework in the online shopping scenario:

- *Privacy policy* stating that the "address" information of the user should not be shared with "advertising" services.
- *Confidentiality policy* stating that the "credit card" information of the user should not be shared with services below a certain rating.
- *Operational policy* stating that the "credit card" information of the user should be disclosed only if the charges are less than the available credit.

The main contributions of the paper are as follows:

- We present the design and implementation of a framework for dynamic specification and enforcement of client's access control policies in composite Web service interactions. The presented framework is compatible with existing Web services infrastructures.
- We present an access control mechanism that limits data shared with services in a composition to the minimum essential data they need to accomplish their task, based on client-specified policies. The mechanism is independent of third party software or services, obviating the need to rely on proper enforcement/communication of policies in external service domains.
- We demonstrate through experiments with a real-world scenario that the framework is able to achieve satisfactory performance to meet real-time constraints of Web services.

The rest of the paper is organized as follows: Section 2 provides a brief background on policy enforcement approaches and Web service composition patterns. Section 3 describes the proposed EPICS framework and its operation on a sample Web services scenario. Section 4 presents the implementation details of the framework. Section 5 provides a security and performance evaluation of the framework. Section 6 discusses related work and Section 7 concludes the paper.

2 Preliminaries

2.1 Enforceable Security Policies

As Schneider states [11], the applicability of any security policy in practice is dependent on the enforceability of that

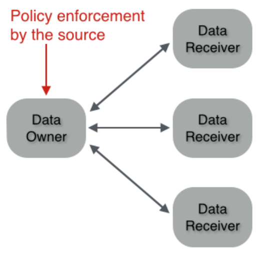


Fig. 2: Policy enforcement at owner.

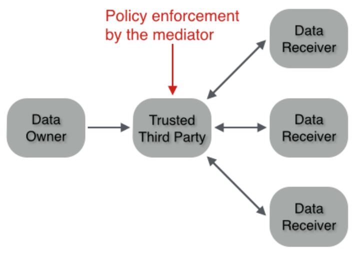


Fig. 3: Policy enforcement at mediator.

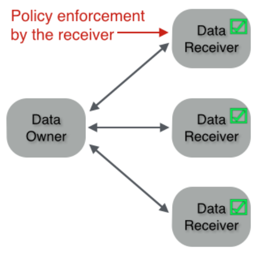


Fig. 4: Policy enforcement at receiver.

security policy. A common way to enforce security policies is to utilize an Execution Monitor (EM) (enforcement mechanism), where the EM *runs in parallel* with the target system, observes its behavior, and terminates the system if an action leads to a violation of the policies [11]. Basin et al. extended the role of Schneider's EM by proposing to distinguish between controllable actions, i.e. actions that could be stopped, and observable actions, i.e. actions that EM cannot prevent [12]. Enforceable security policies in distributed systems can be modeled using the concept of *security automata* as proposed by Schneider [11], described by:

- A finite set of states Q
- A finite set of initial states $Q_0 \in Q$
- A finite set of input symbols I
- A transition function δ: $(Q \times I) \rightarrow 2^Q$

Here Q represents the set of valid states of the system in question, i.e. the states in which the set of security policies (predicates) defined on the system hold. The set of inputs consist of the events that are observable by the EM and have an effect on the state transitions. When modeling data access policy enforcement, the inputs can include events such as authentication success or failure, as well as the results of checking conditions such as whether a data item or policy has expired, the integrity of policies has been preserved, etc. Access control policies that are representable using security automata can be enforced by feeding the automata into a simulator running in parallel with the service under monitoring, and denying access when an input is rejected by the defined automata. At each step of execution, the simulator receives the input before the system and allows execution of the step if the automaton is able to make a transition on the input symbol given its current state. If no transition is allowed, access is denied.

Figure 5 shows a sample security automaton for a policy requiring authentication of a data receiver before authorization takes place, and denying access to data after three authentication failures or a single authorization failure. Here, the automaton states indicate whether authentication/authorization has already taken place, or authentication has failed one or more times. The input fed into the automaton consists of the authentication/authorization request results, and no other event in the system is monitored.

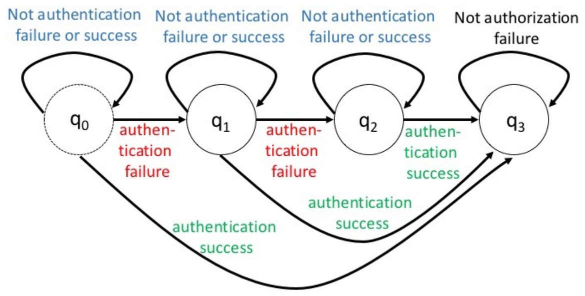

Not authentication failure or success
Not authentication failure or success
Not authentication failure or success
Not authorization failure
q0
q1
q2
q3
authen-tication failure
authen-tication failure
authen-tication success
authentication success
authentication success

Fig. 5: Security automaton example.

2.2 Policy Enforcement Mechanisms

In case of policy-protected data disclosure in a distributed environment, such as Web service interactions, a policy enforcement mechanism needs to be present at the owner of the data, at the receiver of the data, or at a mediator. Below we provide descriptions of these three cases.

2.2.1 *Policy enforcement at owner*

Figure 2 shows a typical scenario of policy enforcement at the data owner. Data owner has an application on their host, which is responsible for the enforcement of their policies. This is the traditional and most commonly used approach for data sharing in the client-server paradigm. The main issue with such solutions is that they require the data owner to be visible and accessible to all hosts and continuously available to serve requests.

2.2.2 *Policy enforcement at mediator*

Figure 3 shows a typical scenario of policy enforcement at a mediator. Such approaches use a Trusted Third Party (TTP) as a broker between data owners and data recipients. The owner sends data and policies to the TTP, which is responsible for enforcing policies and allowing authorized data access requests. For instance, a Publish/Subscribe model is based on this approach. There are various problems with this approach, including trust issues with the TTP, and loss of control over information published to the TTP. A TTP is a single point of failure and can leak information in case of hacking attacks or insider abuse. Furthermore, a TTP can aggregate private information and release it to interested parties for profit or under subpoenas, court orders, search warrants, etc.

2.2.3 *Policy enforcement at recipient*

Figure 4 shows a typical scenario of policy enforcement at the recipient. In such approaches, there is a trusted hardware or software component on the receiving host, which is responsible for policy enforcement. The security policies associated with the data are either predefined or are defined during data dissemination. The policies are enforced by the

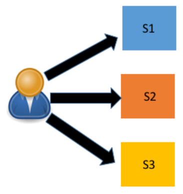


Fig. 6: Composite Web service.

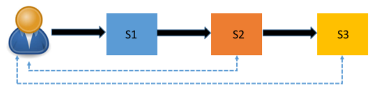


Fig. 7: Chain of Web services.

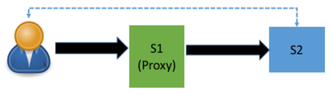


Fig. 8: Proxy Web service pattern.

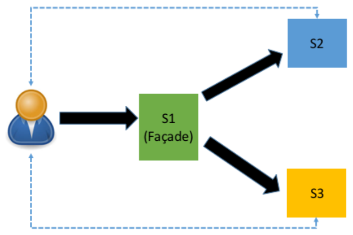


Fig. 9: Facade pattern.

trusted component at the host after the data is received. The main problem with this approach is that it requires advance knowledge of the recipient hosts and the presence of a trusted component on the hosts.

EPICS, as will be described in section 3, is based on this last method of policy enforcement, with an enforcement mechanism integrated into data itself, obviating the need to trust the data receiver or rely on a trusted component at the receiver.

2.3 Web Service Composition Patterns

Web services can be composed using one of the many SOA composition patterns to allow reusability and high cohesion. Some of the most widely used composition patterns include the *proxy*, *chain* and *facade* patterns. Figure 6 shows a composite Web service consisting of three service components. This is the most basic service composition style, where the client individually invokes all services and provides its data to them separately in every invocation, composing the end result itself. In terms of data access control, this type of composition style is how EPICS operates when enforcing access control policies.

Figure 7 shows a *chain* service composition pattern, where the client invokes the first service $S1$ in the chain, which invokes $S2$, which further invokes $S3$ to complete the request. In such a chain of service requests, each service in the chain requests the data item it needs to accomplish its task from the client, as marked with the dashed arrows. In a service chain consisting of n services, this results in $n-1$ extra interactions between the client and the services for data sharing, incurring delays in response.

Figure 8 shows a *proxy* service composition pattern. A proxy adds an additional level of indirection between the client and the invoked service. The invoked service in the composition requests the data item from the client resulting in the extra interaction.

Figure 9 shows an example of the *facade* service composition pattern with three services. A facade acts as a gateway to a set of services. As before, each service in the composition requests the data item from the client resulting in $n-1$ extra interactions for a composition consisting of n services.

3 EPICS Design

EPICS is a distributed data dissemination framework for enforcing data access and security policies in Web services possibly consisting of multiple components beyond the client's interaction zone. Data access in EPICS is based on an application of the *Principle of Least Privilege* [13] to the services domain, which limits data shared with each service in a service composition to what they actually need to accomplish their task and nothing more. The main building blocks of EPICS are *active bundles*, which are data packets augmented with self-protection capability, and a policy enforcement mechanism integrated into active bundles to ensure compliance with client-specified access control policies at every hop in service interactions. Below we describe the main components of EPICS and the operation of the framework to prevent data leakage in Web service interactions.

3.1 Active Bundles

Traditionally, solutions for protecting data against unauthorized access have considered data as passive entities that are unable to protect themselves. Such solutions require another active and trusted entity–a trusted processor, a trusted memory module, a trusted application or a trusted third party–to enforce data access authorization policies. We challenge this assumption and propose a framework based on the Active Bundle (AB) construct that transforms passive data into an active entity. An AB [14], [15], [16] is a self-protecting data encapsulation mechanism composed of:

- *Sensitive Data* is the information that needs to be protected. It can include any digital content such as documents, images, audio, etc. The content can have several data items, each with different protection requirements and applicable policies to ascertain their secure distribution and usage.
- *Metadata* includes policies that control the AB behavior and govern data dissemination. Policies can be operational constraints such as expiration time, active time, maximum number of requests, life duration etc., or data constraints that address privacy and access control.
- *Engine* is a specific-purpose program used to operate the AB, protect its content and enforce policies to guarantee access control of sensitive data in the bundle, e.g. disclosing to a service only the portion of data that it requires to accomplish its task.

In this work, we extend active bundles with the following capabilities to utilize them for policy-based data dissemination in composite Web service interactions:

- *Authentication* to verify the identity of the service requesting access to data. It can be based on standard authentication mechanisms, such as passwords, certificates, biometrics, etc.
- *Authorization* to validate the data access request of the service. It allows or denies access based on the evaluation of the applicable policies.

- *Integrity Verification* to dynamically verify the correctness of the AB's execution. If any modification is detected in the AB's data, policies, or code, the data access request is denied.
- *Data Disclosure* to decrypt the requested data and disclose to the service only if the service is authenticated and its request is authorized. It identifies the encrypted data item and determines the appropriate key to decrypt it.

3.2 Policy Enforcement in EPICS

3.2.1 Access control model

In EPICS, the data owner (client) needs to share data consisting of multiple items with a set of services, where each service is only authorized to access specific data items. The owner can directly interact only with a subset of those services and relies on them to disseminate data to other services. Note that in order to ensure the correct delivery of appropriate data to each service, it is necessary that each service shares the entire data even though the services are only authorized for a certain subset of the data. The owner may not know all the services in advance, but can use access control policies to authorize services dynamically. Therefore any service that satisfies the conditions defined in the policies for a data item is authorized to receive the data item.

Let D be a set of data items d_i owned by U that need to be shared with authorized services involved in a Web service invocation. Each data item d_i is a key-value tuple of the form $< k_i, v_i >$. We assume that each host S_i is already aware of the set of item keys k_{S_i} of items d_{S_i}, which it may be authorized to access. For instance, the email data item is organized as $< email, abc@xyz.com >$, where $email$ is the key element already known to the interested services, and $abc@xyz.com$ is the value element that should be disseminated only to the authorized services. P is the set of access control policies defined on D. An item d_i in data D may have several applicable policies. AP_i is the subset of P, which represents the policies applicable to item d_i. Note that AP_i can be $\emptyset$. U encrypts each data item value v_i separately and gets the corresponding ciphertext c_i. C represents the set of ciphertexts corresponding to all data item values. U and each service S_i shares the encrypted data C with other services instead of sharing the actual data D.

Common access control models that could be utilized for this data dissemination problem follow.

Role-based access control (RBAC) is based on assignment of specific roles to entities and proof of possession of a specific role entitles the receiver access to data items their role allows [17]. Although RBAC has been fairly popular in many domains including Web services, it has disadvantages for the data dissemination problem here, as it would require assignment of roles to all services that might be invoked in a composition, exhibiting the problem of pre-knowledge of possible services. It also does not allow specification of more fine-grained policies specific to the individual services and the operation context.

Attribute-based access control (ABAC) defines access control based on values of specific attributes of the data receiver [18]. The policies in ABAC define rules by combining the attributes of subjects (entities requesting access to resources), resources (objects to which access is requested), actions (type of access, e.g. read or write), and environment (context of the request, e.g. time of the access request). Under this model, an entity is granted access only if it satisfies the attributes defined in the policies for the requested resource. The use of attributes allows to express more complex access control policies and enables fine-grained policy-based authorization. Access control in EPICS is modeled based on ABAC due to its flexibility and richness of expression when specifying policies.

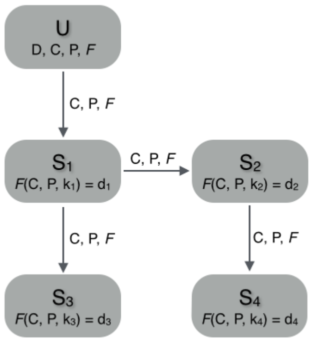


Fig. 10: Distributed data dissemination in EPICS.

The access control mechanism we propose for EPICS is based on a set of functions F, such that, given ciphertext set C, access control policies P and an item key k_i as input, it outputs the corresponding item value v_i if the service requesting access to the value v_i is authorized to get it based on access control policies AP_i associated with d_i. F is constructed by U and sent to each service S_i with which it can interact along with C and P (as shown in Figure 10). When a service S_i receives F, C and P, it forwards them to the set of services S_{S_i} involved in its service orchestration. Access control policies can include attribute-based policies such as the service type being a payment service or the service location being in the US, as well as operational policies based on the execution context, service interaction parameters etc. In the proposed framework, F is integrated into the AB engine.

F includes functions for:

- Verification of a receiving service (authentication, authorization and checking of operational constraints such as data expiration time)
- Integrity checking of the functions in F
- Derivation of the encryption/decryption key for the different data items
- Decryption of the ciphertexts for the different data items using the derived keys

For the encryption of the data items in an AB, EPICS uses symmetric-key encryption with secret keys generated from the execution flow of the AB's engine (as described in section) instead of an approach like Ciphertext-Policy Attribute-based Encryption (CP-ABE) [19], where a service's private key is associated with a set of attributes and a TTP is needed to manage attributes and issue keys. The proposed method offers the following advantages over existing approaches:

- There is no need for distribution of decryption keys to services in advance, which is infeasible especially in case of cloud-based services unknown to the client.
- Attribute revocation, which happens to be very costly in the case of an approach like CP-ABE, is not an issue due to the dynamic generation of keys, which provides both forward and backward security.
- Dynamic integrity checks on AB policies, data and code can be integrated into the key generation process, preventing disclosure of data in case of tampering. EPICS makes

it possible to integrate contextual parameters into data disclosure decisions as opposed to CP-ABE.

- Policy definitions in CP-ABE are limited to boolean and threshold operations over the defined set of attributes, so the type of access control is restricted compared to EPICS.

3.2.2 Policy enforcement mechanism

The policy enforcement mechanism in EPICS is based on the idea of execution monitoring [11] of the data access requests (described in section 2.1). The EM receives requests from services for data access and permits them to access the data only if the applicable policies allow it. It uses the history of execution steps and their outputs, such as a step for authentication of service and a step for authorization of data access request based on the evaluation of applicable policies, to decide whether the action is authorized, in which case execution is allowed, or the action is unauthorized (i.e. violates the applicable policies), in which case execution is terminated. The execution steps such as authentication and authorization are controllable actions and, therefore, their execution can be terminated by the EM [12].

The access policies that we consider in this system are enforceable by execution monitoring as they satisfy the enforceability conditions [12]: (a) inspecting the execution steps of the service interaction is sufficient to determine whether it is policy-compliant; and (b) the execution steps of previous interactions are independent and do not affect the policy compliance of the current interaction. In relation to standard access control architectures such as eXtensible Access Control Markup Language (XACML) [20], the EM acts as both a policy decision point (PDP) that is used to evaluate data access requests against applicable policies, and a policy enforcement point (PEP) that is used to intercept access requests and enforce policies by approving or denying them based on the evaluation results of the PDP.

In a composite service environment, the client request (including the client's data) is disseminated to external service domains, which are hidden and possibly unknown to the client. This adds a restriction on the placement of the EM because the EM needs to be available in every domain, where data is disseminated and access is requested, to ensure the enforcement of policies. The existing policy enforcement mechanisms place EM either at the source, the destination, or a mediator as discussed in 2.2. Since the component services in the composition are dynamically decided and hidden from the client, the placement of the EM at the source or a mediator is not possible. A pre-defined placement of EM at all possible destinations (component services) is not feasible, especially in a cross-domain multi-provider environment with real-time constraints.

In this work, we propose a mobile EM that can be dynamically placed on the component services that participate in an interaction to control data disclosure at each endpoint and ensure end-to-end policy enforcement. The solution utilizes active bundles, which include the sensitive data, the related policies as metadata, and the EM as the engine. In this way, the EM travels along with the data and the policies as part of the AB.

Figure 11 demonstrates the operation of the EM using a security automaton in the case of an access request for the value of the data item *credit card* in the online shopping scenario. In this simple case, there are two policies associated with *credit card*—a payment policy and a location policy related to the payment processing service. The first action taken by the engine is to check the integrity of the service authentication code and proceed with running authentication only if the code is intact. This is followed by checking the integrity of the two policies and evaluating the policies only if their integrity has been preserved. Finally, data access is authorized only if the authorization does not fail based on the supplied access control policies. Notice that this automaton allows multiple authentication failures, but no authorization/integrity check failure and can easily be modified to terminate execution in case of any authentication failure. The ability to specify such operational constraints provides increased control to the client over handling their data under different contexts. Typically a digest of the policies (part of the AB metadata) calculated using a secure hash function are used for the integrity checks.

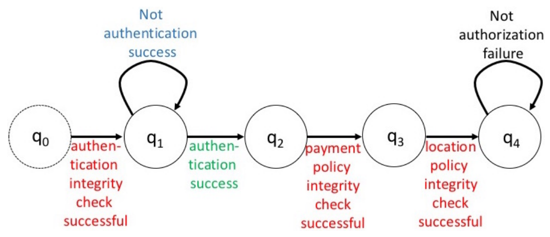

Not authentication success
Not authorization failure
q0
q1
q2
q3
q4
authentication integrity check successful
authentication success
payment policy integrity check successful
location policy integrity check successful

Fig. 11: EPICS policy enforcement security automaton example.

3.2.3 Key derivation

The symmetric keys for AB data items are derived based on unique information generated from the execution control flow path of the AB. The main control flow steps include authentication and authorization. These steps generate unique information during interaction with a service only if the service is authentic and its request is authorized based on service attribute values (as proven by a certificate authority). A Key Derivation Function (KDF)[1] is used to derive the key based on the information. In order to ensure proper entropy in the key, the information is transformed into a secret (hash of the information) first and the secret is used to derive the key (using methods such as SecretKeyFactory, PBEKeySpec and SecretKeySpec provided by javax.crypto[2]). During AB creation, each data item is encrypted using a unique key derived by running the AB engine with valid attribute values that would result in providing access to the data item according to the associated policies. The distinct information for each key comes from the authentication token (such as a password or PKI issued token), authorization policies applicable to the associated data items, and the mobile code executed by the AB, as described in section 3.3. During interaction with an authenticated-authorized service, the AB execution control flow steps generate the same information, which is used to derive the appropriate key and decrypt the data item.

1. A KDF is a deterministic algorithm to derive cryptographically strong secret keys from some secret value [21].

2. https://docs.oracle.com/javase/7/docs/api/javax/crypto/package-summary.html

Let us consider a toy example of key generation for the case of the security automaton in section 3.2.2. We extend the automaton with outputs generated at each state transition to demonstrate the process of key derivation as seen in figure 12. Let us assume for simplicity that the output at each state transition i, denoted by H_i is the hash of the function code run before the transition takes place (e.g. authentication code), and H_4 is an XOR of the required attribute values specified in the two policies to grant access. Let us further assume that the key aggregation function generating the secret key for encryption of the credit card information from the outputs of the automaton is XOR. Given a collision-resistant hash function, the key generated during a complete run of the AB will only match the key generated during AB generation if (a) the integrity of all code during the run has been preserved and (b) the attribute values of the service requesting access match those specified in the policies.

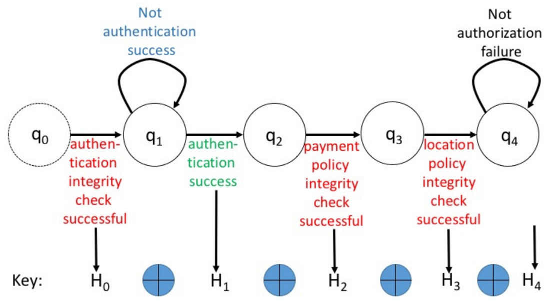

Not authentication success
Not authorization failure
q0
q1
q2
q3
q4
authen-tication integrity check successful
authen-tication success
payment policy integrity check successful
location policy integrity check successful
Key:
H0
H1
H2
H3
H4

Fig. 12: Security automaton extended with outputs for key derivation.

3.3 Tamper Resistance

The correctness of access control policy enforcement in EPICS depends on the correct execution of AB control flow steps. Since the service domain in which the AB is running may not be trusted, we need to check that the integrity of all elements of the AB including its engine is preserved throughout its execution. The approach we propose for tamper-resistant execution of ABs is based on augmenting the AB engine with integrity checkpoints distributed throughout the engine code using aspect-oriented programming (AOP) [22] as proposed by Angin et al. [23] to ensure timely detection of tamper. Tamper resistance checks the integrity of the AB execution to ensure that it has no difference from the original code. Each time an AB step is executed, the tamper resistance module calculates its digest and the digest of the resources used by it (e.g. authorization module and the policies that it evaluates) through a secure one-way hash function (such as SHA-2). These digests are combined with the information used for key derivation. Therefore, if there is any modification to any step or its resources, the digests change, making the derived key invalid.

In a Java-based implementation of the proposed tamper resistance model, integrity checkpoints can be inserted around every method call using the *AspectJ*[3] AOP framework. The code snippet in listing 1 shows how to implement tamper resistance using AspectJ for the methods of an policy enforcement engine class file *Authentication*. As clearly demonstrated by the code snippet, the whole process of adding integrity checking functionality is completely transparent to the actual engine code.

```
public aspect Guard {
  pointcut methodCalls():
    execution(* Authentication.*(..));

  Object around(): methodCalls() {
    ClassPool clPool=ClassPool.getDefault();

    byte[] initHash =
    (byte[])codeHash.get("Authentication");
    byte[] hash;
    try {
      CtClass cls =
        clPool.get("Authentication");
      byte[] bytecode = cls.toBytecode();
      MessageDigest messageDigest =
        MessageDigest.getInstance("SHA-256");
      messageDigest.update(bytecode);
      hash = messageDigest.digest();
      return proceed();
    }
    finally {
      if (java.util.Arrays.equals
                (hash,initHash)) {
      System.out.println("true␣hash␣value");
    }
    else {
      reportTamper();
    }
  }
}
```

Listing 1: AOP example using AspectJ

3.4 EPICS Operation

Figure 13 depicts the operation of EPICS for policy-based dissemination of data to Web services. A client (data owner) uses the framework as follows to interact with the service providers. He provides his data, applicable access policies, and the target service to the AB Generator application, which uses the information to generate an *AB* and sends the AB to the target service. The composite service forwards the AB to its component services to transmit the client's data, policies, and the EM. The services interact with the AB to send requests for data access. All data access requests are intercepted by the AB's engine (EM), which is responsible for enforcing the policies and allowing access to the data.

Figure 14 shows the UML activity diagram of AB steps in response to a request. The main steps follow.

Service Authentication is based on the use of signed digital certificates. The service sends a signed request along with its X.509 certificate signed by a trusted Certificate Authority (CA). AB's engine verifies that the request is signed by the service, the service certificate is valid, has not expired, and is signed by a trusted CA certificate. A positive verification authenticates the service.

Request Authorization uses the client policies that are applicable to the requested data item to evaluate the respective attributes of the interacting service (e.g. trust level), the data item request, and the environment conditions (e.g.

3. https://eclipse.org/aspectj

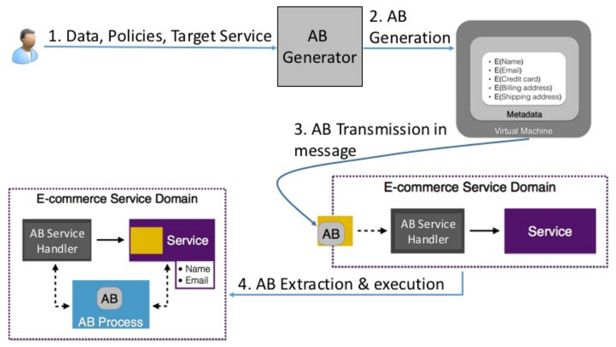
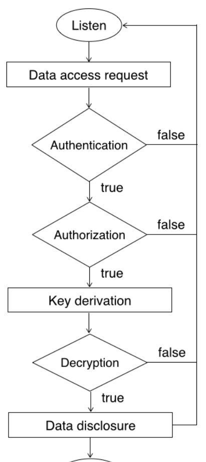


Fig. 13: EPICS operation.

emergency context, attack context). A positive evaluation of the applicable policies authorizes the service.

Data Disclosure identifies the data item that needs to be decrypted. The symmetric key that was used to encrypt the data item during AB creation is dynamically derived using the unique information generated during the AB's execution along with the digests of the AB modules (described in section 3.2.3). AB uses the key to decrypt the data item and returns it to the service.



3.5 Scenario Revisited

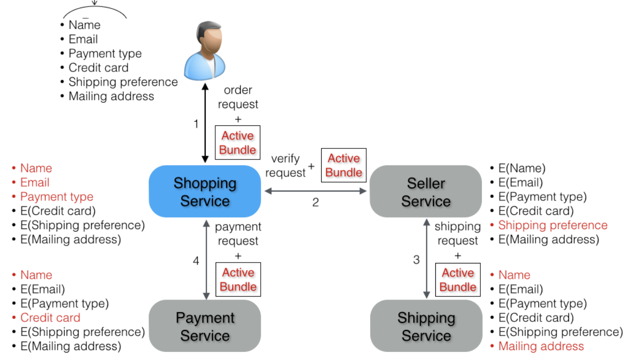


Fig. 15: EPICS in action for online shopping.

Figure 15 shows the online shopping scenario introduced in section 1.1 updated based on the proposed solution. We assume that the client and the shopping service have a pre-existing trust relationship, for instance, established during client's registration with the shopping service. During registration, the client, instead of sending all information, sends an AB to the service. The shopping service stores the AB and associates it with the client's account. When the client logs in, the shopping service starts the AB and gets only the information from the AB which it requires to provide service. When the client sends an order request, the shopping service sends AB along with the requests to the seller and the payment services. Similarly the seller service sends AB along with the request to the shipping service. Each service interacts with AB and gets only the information which it needs to provide the service. After completing the request, the seller, shipping and payment services discard the information and the AB. The shopping service stops the AB when the client logs out and discards any information received from AB. This also reduces the services' liability to the client, because if the service is compromised, the client information is not leaked and the attacker only gets access to the client's AB.

4 Implementation

We have implemented a prototype of the EPICS framework with software components for Active Bundle, AB Generator and AB Handler. In the subsections below we provide the implementation details of each component with the proposed security extensions.

4.1 Active Bundle

The AB consists of data, metadata, and an engine, as discussed in section 3.1. The three components have been implemented as follows:

Data: Each data item is encrypted and stored as a key-datum pair using JavaScript Object Notation (JSON)[4] format, e.g., { ab.email : E(abc@xyz.com) }. This organization allows services to query AB for specific data items using the *key* attribute. The symmetric key for encryption is based on the AB execution control flow steps as described in section 3.2.3.

Metadata: They include the access control and operational policies specified by the data owner. The policies are specified using the Attribute-based Access Control (ABAC) model and implemented using eXtensible Access Control Markup Language (XACML)[5] 3.0. An example policy, "access to credit card information is allowed only to certified visa payment services with 5-star ratings", expressed using XACML elements, is shown in Table 1.

TABLE 1: Policy: Payment.

ALLOW	
Resource	Credit card
Subject	Visa payment services
Action	Read
Environment	Rating >4

Engine: This has been implemented as a set of Java classes that execute the AB, handle access requests and perform policy evaluation and enforcement. The policy evaluation is implemented using an open source XACML PDP[6]) named WSO2 Balana[7]. AB provides a set of API methods for

4. A lightweight data-exchange format. http://json.org/.
5. XACML is an open OASIS standard for expressing access control polices using XML [20].
6. PDP is the module that evaluates access requests against authorization policies and issues access decisions [20].
7. WSO2 Balana. https://github.com/wso2/balana

interaction with services. The API is implemented using the Apache Thrift framework.[8]

4.2 AB Generator

The AB Generator has been implemented as a Node.js[9] Web application that creates ABs. Data owners use the application interface to specify the data (as key-value pairs), applicable access control policies (as XACML/JSON elements), the operational policies, the AB template[10] to use, and the Web address of the target service for AB. The application parses the input and uses it to derive the symmetric keys for data encryption. It encrypts the data and adds the ciphertext and the policies to the specified AB template. Next, it generates the AB as an executable Java Archive (JAR[11]) file. It serializes the AB file using Base64 encoding to preserve the data format, and appends the output to the HTTP body of the message, which is sent to the specified target service.

4.3 AB Service Handler

The AB Service Handler is a Web service middleware extension, implemented as a Node.js module that intercepts incoming requests before being serviced by the Web service APIs. The handler is invoked for each service request and performs the following steps:

1) Checks and proceeds only if the incoming request includes an AB, otherwise it passes the request to the next method.
2) Extracts the AB from the request.
3) Decodes the AB and stores it on the file system.
4) Generates a port number and starts AB on that port.
5) Starts execution, at which point it is ready for interaction with the service.
6) Passes the AB process information to the next middleware method which is eventually passed to the service.

5 Evaluation

In this section we provide a security discussion of the proposed framework, its translation into practice and performance evaluation of the prototype system implemented in the context of the online shopping scenario introduced in section 1.1.

5.1 Security discussion

5.1.1 How resilient is EPICS against data leakage?

EPICS provides three levels of protection for data disclosure. The first level of protection is based on the authentication. Each service needs to authenticate with the AB in order to access the data. The second level of protection is based on the authorization. Each data access request is authorized based on the evaluation of the applicable policies. The third level of protection relies on the integrity-based key derivation. The data decryption is possible only if the AB is unmodified. The decryption discloses only the portion of the data for which the service is authorized.

Attackers copying active bundles cannot decrypt the data without proper authentication and authorization even if they analyze the AB execution flow offline. The recipient needs to have exact knowledge of AB flow control and which modules have been marked for integration into key generation before receiving the AB to derive the correct key. The integrated modules and their code can be randomized for each AB dissemination to prevent replaying. Additionally, the key does not only depend on the module hashes, but also authentication tokens for services, without which proper key derivation is not possible. Authentication and authorization combined with tamper resistance prevent an attacker from bypassing access control.

The framework provides protection against the following threats that are possible in the current Web services model:

- *Unauthorized data disclosure*: This threat occurs when an unauthorized service accesses the client's data or an authorized service accesses the data items for which it is not authorized. In EPICS, the use of policy evaluation and enforcement for each data access request ensures that the appropriate data is disclosed only to an authorized service.
- *Ignored policy transmission*: This threat occurs when a service ignores to transmit the policies during data dissemination. In EPICS, the policies are always transmitted along with the data by means of AB.
- *Disregarded policy evaluation*: This threat occurs when a service disregards the evaluation of the policies associated with the data. In EPICS, the use of EM (as part of AB) ensures that any data access request is intercepted and evaluated against the relevant policies.
- *Circumvented policy enforcement*: This threat occurs when a service accesses the data either by disregarding the policy evaluation or by ignoring the decision of the policy evaluation. In EPICS, the use of EM (as part of AB) ensures that the decision to allow or deny access to data is based on the results of the policy evaluation.
- *Unauthorized data dissemination*: This threat occurs when a service disseminates the data to an external domain and the data is accessed by unauthorized services. In EPICS, the use of AB to transmit data and policies ensures that the services in any domain are unable to access the data if they are not authorized.

5.1.2 What factors does EPICS resilience depend on?

While EPICS provides protection against various data leakage threats as discussed above, the level of resilience against attacks depends on the level of protection it has against malicious receivers. A malicious service that receives an AB can try to attack the AB to gain unauthorized access to the data or compromise the AB to disseminate incorrect data or attack other services. Therefore, an AB needs to be

8. Thrift is a cross-language Remote Procedure Call framework that defines an interface language and a binary communication protocol to develop services that work seamlessly across different implementations. https://thrift.apache.org/

9. Node.js. https://nodejs.org/

10. An AB template defines the program skeleton of AB and is used to generate ABs with the same structure. It includes the implementation of the invariant parts (engine) and placeholders for customized parts (data and policies). ABGen can use multiple AB templates, for instance, based on the different CA certificates.

11. JAR is a standard way of packaging Java classes, metadata and resources into one file. It helps to distribute Java code.

protected against services that are malicious or have been compromised. This can be achieved as follows:

- *Secure communication*: An AB should be transferred using secure communication, e.g. HTTPS. This prevents man-in-the-middle attacks during AB transfer. When a service interacts with an AB, the AB employs authentication to identify the service and verify the authenticity of the service to ensure that it is interacting with a legitimate service, which prevents masquerade attacks on the AB. The AB API provides the `getSecureValue()` method, which a service can use to receive the data encrypted with its public key. The data can be decrypted only by using the secret key of the service. This prevents the man-in-the-middle attacks on the interaction between an AB and a service.
- *Tamper resistance*: The AB uses code integrity checks to provide protection against tampering attacks. It uses the digest values of its modules to dynamically derive the secret keys for data decryption. The correct keys are generated only if the modules are unmodified, which ensures correct execution. This prevents attackers from gaining access to AB's data (e.g. using tamper attacks that modify the policies of the AB).
- *Cloud-based execution*: A trusted cloud platform can be used to execute the AB (e.g. IBM Bluemix[12]). Services can interact with the AB executing in the cloud and get access to the data only if they are authorized. Note that the cloud platform in this case is used only for code execution; it does not broker data requests or perform policy enforcement and, therefore, does not get access to data. The execution of the AB in the cloud prevents tampering and hijack attacks.

5.2 Translation into practice

The services need to implement the AB Service Handler for supporting and handling requests that use AB. To interact with AB, the services need to use an AB Client that provides the protocol for communicating with AB. The AB Service Handler and AB Client can be provided as platform specific libraries that the services can include in their application servers. No changes are required at the service API layer as the AB is included in the HTTP body and transferred by means of REST messages.

Specification of access policies in EPICS are easily translated into practice using XACML. The policies define the applicable resource (requested data item), the allowed subject (requesting service), the permitted action (read access), and the acceptable environment conditions. Figure 16 shows a sample policy, which blocks the access to user's address information (resource) if the requesting service (access subject) is not a shipping service. In practice, use-case specific policy templates, with attributes pre-populated by default values and selectable through a drop-down list of options, can be provided as part of the AB Generator application on the user side, which the users can modify to specify their preferences.

12. https://console.ng.bluemix.net

```
<Policy xmlns="urn:oasis:names:tc:xacml:3.0:core:schema:wd-17" PolicyId="sample2"
    RuleCombiningAlgId="urn:oasis:names:tc:xacml:3.0:rule-combining-algorithm:deny-overrides"
    Version="1.0">
   <Target>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue DataType="http://www.w3.org/2001/
                   XMLSchema#string">ab.user.shipping.address</AttributeValue>
               <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-
                   id" Category="urn:oasis:names:tc:xacml:3.0:attribute-category:resource"
                   DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
         </AllOf>
      </AnyOf>
      <AnyOf>
         <AllOf>
            <Match MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
               <AttributeValue DataType="http://www.w3.org/2001/XMLSchema#string">Shipping</
                   AttributeValue>
               <AttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-
                   id" Category="urn:oasis:names:tc:xacml:1.0:subject-category:access-subject"
                   DataType="http://www.w3.org/2001/XMLSchema#string" MustBePresent="true"/>
            </Match>
         </AllOf>
      </AnyOf>
   </Target>
   <Rule Effect="Permit" RuleId="permit-rule"/>
</Policy>
```

Fig. 16: EPICS address policy example.

5.3 Performance experiments

We conducted a series of experiments to measure the overhead of using active bundles for interaction with services. We describe the experimental setup and present the results in the following sections.

5.3.1 *Experimental setup*

The experiments were conducted using Amazon EC2 cloud[13] to bechmark AB's performance. The variables used in the experiments are as follows:

1) AB types: These include 4 different implementations of AB — ABc, ABct, ABx and ABxt.
   - In ABc and ABct versions, the AB policy elements (such as subject, object, resource etc) are specified in JSON and the PDP is implemented as conditional statements in Java code that evaluate the policies. The ABct version includes tamper resistance in addition.
   - In ABx and ABxt versions, the AB policies are specified in XML according to XACML specifications and WSO2 Balana library is used as PDP for policy evaluation. The ABxt version includes tamper resistance in addition.
2) AB policies: The applicable policies for a data item were varied exponentially (2^x) from 1 to 16.
3) AB execution environment: Two different service execution environments (Amazon EC2 instance types) were used — EC2 Large and EC2 XLarge.

The AB is created with 6 data items, as shown in figure 15, and at least 1 policy applicable to each data item. An interaction between AB and a service involves — service sending request to AB for a data item, AB authenticating the service, AB authorizing the service request for the data item using applicable policies, AB decrypting the data item, and AB sending the response back to the service. It also involves tamper resistance when ABct or ABxt are used.

5.3.2 *Results*

The results are reported based on the mean of data collected over 5 runs, where each run includes 100 interactions.

Figure 17 shows the graph for increase in the size of different AB types as the policies of AB grow exponentially. The results show that the growth in AB size is linear with the

13. We used the EC2's C3 instances for experiments as they are suitable for Web-servers. http://aws.amazon.com/ec2/instance-types/

increase in policies for each type of AB. Tamper resistance adds a slight overhead to ABct and ABxt size. The use of XACML-based policies incur some additional overhead.

The first experiment was conducted to measure the interaction time between AB and a service for a data item request under different experiment settings. Figure 18 shows the graph for interaction time (in ms) between AB and service on EC2 Large for different AB types as the policies of AB grow exponentially. The results show that the growth in interaction time with the increase in policies is constant for ABc and ABct and is linear for ABx and ABxt. The difference in interaction time and growth is due to the difference in the implementation of policies. In case of ABc and ABct, the policies are specified as conditional statements in Java code and do not require any external library for evaluation, whereas, in case of ABx and ABxt, the policies are specified in XML according to XACML specifications and use WSO2 Balana for evaluation. Evaluation of conditional statements is highly optimized, whereas the evaluation of XACML policies involve the traversal of XML policy and request trees, which takes longer. Figure 19 shows the graph for interaction time (in ms) on EC2 XLarge. The graph confirms the earlier trends.

Figures 20a and 20b show the overhead of using tamper tesistance in AB with and without XACML respectively on EC2 Large. The overhead is linear in ABxt (with XACML) and constant in ABxt (without XACML). The difference in overhead comes from the tamper resistance's digest calculation of XACML policies, which takes longer. XACML policies are XML files in addition to the code, whereas in case of ABct, the policies are conditional statements which are already part of the code.

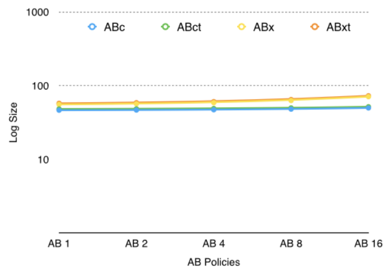


Fig. 17: AB size vs. number of policies.

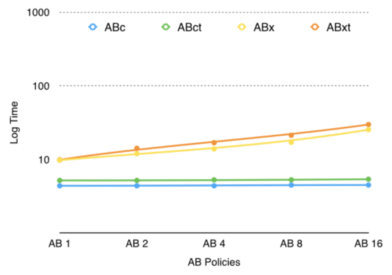


Fig. 18: AB-service interaction time on EC2 Large.

We also conducted experiments to measure the complete interaction time of the online-shopping scenario. These experiments measure the round-trip time taken by the client's order request. It includes the network time to transfer the

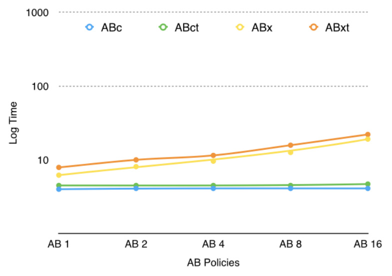


Fig. 19: AB-service interaction time on EC2 XLarge.

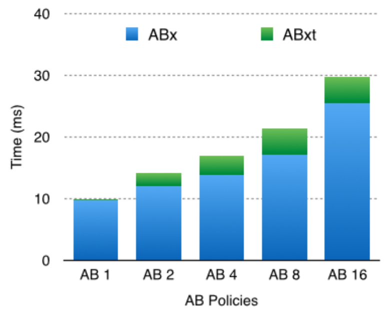


Fig. 20a: Tamper resistance overhead with XACML.

AB to the services in the composition, interaction time of the AB with each service, and the response time of each service. Figure 21 shows the round trip interactions in the scenario with EPICS.

Figure 22 shows the results obtained for average round-trip scenario interaction time on EC2 Large. The graph follows the same trends as the interaction time graphs shown in Figures 18 and 19. The results show that the interaction time of the scenario is under 1.7 secs on EC2 Large even with 16 XACML policies. The use of JSON policies reduces it to under 1 sec. These scenario times easily meet the real-time Web service constraints, showing that EPICS is promising for adoption in real-world composite Web services.

5.3.3 Comparison with baseline

We also conducted experiments to compare the performances of a baseline case, policy enforcement at data owner, vs. policy enforcement with EPICS for one specific composition pattern, a service chain consisting of three services $S1$, $S2$ and $S3$, where $S1$ invokes $S2$ and $S2$ invokes $S3$ to complete the request. Figure 23 shows the round-trip interactions between the client and the chain of services for the baseline case, and figure 24 shows the interactions for the same service chain with EPICS. In the baseline case, data access authorization and authentication have to go through the client for every service needing access to a data item owned by the client[14], whereas in EPICS each service in the chain receives the entire active bundle shared by the client and authentication and authorization take place at the service domain through the active bundle API. This results in fewer interactions between the client and the services to complete a service request, i.e. in the baseline case there is

14. Note that here we make the assumption that the services are able to directly interact with the client.

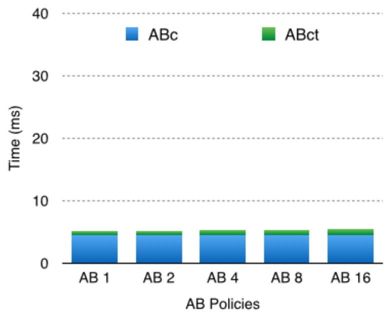


Fig. 20b: Tamper resistance overhead without XACML.

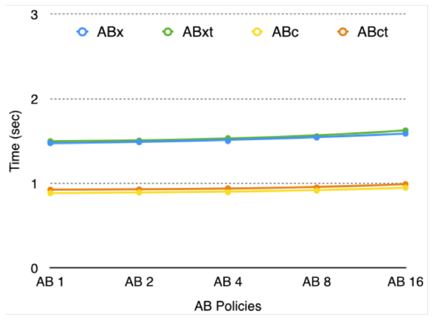


Fig. 22: Round-trip scenario time on EC2 Large.

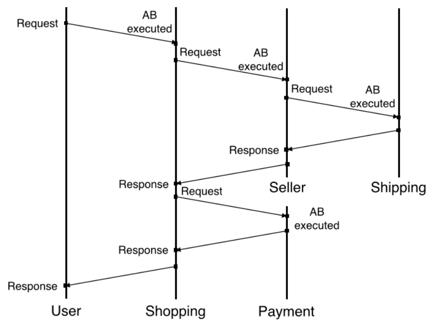


Fig. 21: Round trip interactions between client and composite service in EPICS.

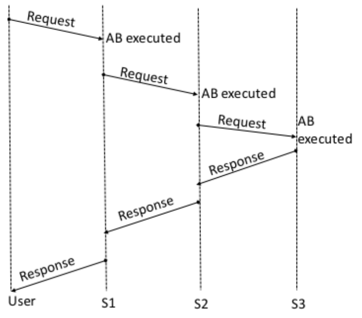


Fig. 23: Round-trip interaction between client and chain of services with policy enforcement at data owner.

an extra data request-response interaction between the client and each service requesting access to a data item.

In these experiments, each active bundle contains 6 data items, each of which has a single applicable access policy, and each service in the chain requests access to 1 data item in the bundle. The flavor of AB used in the EPICS experiments is ABxt, i.e. XACML-based policies and tamper resistance. The experiments were run using C3 Large EC2 instances, 5 times with 100 interactions per run. The average round-trip response time for the baseline case is 908 ms, and the average round-trip response time for EPICS is 1184 ms, which is a reasonable overhead for enhanced security. As the number of services in the chain grows, the baseline approach may not scale due to the extra interactions involved for data access, whereas EPICS will not have the same scalability issues.

6 Related Work

Access control is a well-studied topic in the context of Web services. Various Web service standards have been proposed to address security in SOAP-based services, including WS-Security to provide specifications for credential exchange, message integrity, and message confidentiality during service interactions, and WS-Policy to provides specifications for advertising policies of services and policy requirements of clients. These standards could be sufficient for point-to-point security and policy enforcement in static service invocations, but they fall short of the policy transmission and enforcement in dynamic service invocations because of the involvement of multiple services in a request. Single sign-on and shared authentication solutions such as OpenID[15] and OAuth[16] provide centralized and federated access management, but do not consider specification and enforcement of client policies.

Security Assertion Markup Language (SAML) is an open-standard specification and a framework for exchanging authentication and authorization information between parties in XML [24]. Cross-domain implementation of SAML is a problem, because browser cookies used to maintain the authentication state information cannot be transferred across different DNS. Shibboleth defines an architecture based on SAML and provides an open-source implementation that allows federated identity management, authentication, and authorization [25], which enables cross-domain single sign-on. The solution is prone to the TTP related issues because of its dependence on an identity provider.

DataSafe is a software-hardware architecture that supports data confidentiality throughout their lifecycle [26]. It is based on additional hardware and uses a trusted hypervisor to enforce policies, track data flow, and prevent data leakage. The hosts without DataSafe can only access encrypted data, but it is unable to track data if they are disclosed to non-DataSafe hosts. The use of a special architecture limits the solution to well-known hosts that already have the required setup.

A privacy-preserving information brokering (PPIB) system has been proposed by Li et al. for secure information access and sharing via an overlay network of brokers, coordinators, and a central authority (CA) [27]. To provide protection, this approach proposes a mechanism to divide and allocate the responsibility and processing among multiple brokers so that no single component has sufficient control to

15. http://openid.net/
16. http://oauth.net/

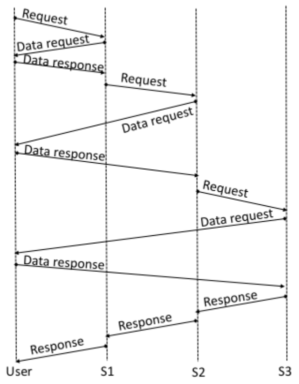


Fig. 24: Round-trip interaction between client and chain of services with EPICS.

make meaningful inference from the information disclosed to it. However, the ownership, distribution, and management of components are a challenge in a large cross-domain system. Also the uses of a centralized TTP to manage metadata, joining/leaving of brokers, and key management creates a single point of failure. Other solutions have been proposed that address secure data dissemination when the recipients are not known in advance. Pearson et al. present a case study of the EnCoRe project that uses sticky policies to manage the privacy of shared data across different domains [28]. The main idea of sticky policies is to make data and policies inseparable so that an unauthorized recipient cannot access the data without satisfying the policies. In the EnCoRe project, the sticky policies are enforced by a TTP, making the approach prone to TTP-related issues.

Another class of solutions for ensuring proper access control in Web services is based on tracking of data flow in applications. These approaches rely on modified language interpreters keeping track of the flow of user data to ensure that access control checks are carried out properly, such as the approach proposed by Dalton et al. [29], which carries out shadow checks before operations execute on data resources in a system. Static program analysis techniques discovering application code paths with missing access control checks are also utilized to discover access control vulnerabilities. However, such approaches require accessing the source code of the applications, are language-specific and make assumptions about the architecture of the application [30], which makes them inapplicable in the case of composite Web services with dynamically determined service compositions. Approaches such as CloudFence [31] and SilverLine [32] enforce access control by associating access control state with data flows, using taint analysis. Data flow tracking solutions in general suffer from significant performance overheads and tight language binding, limiting their applicability in Web services.

7 Conclusion

Composite Web services are generally composed of cross-domain heterogeneous services. The clients interact with the primary service, which can outsource their requests (including their data) to secondary services from different ownership domains. In this case, it is very difficult for a client to determine how their data will be shared and who will access it. This invisible sharing exposes the data to new risks that are otherwise avoidable if data stays within a trusted domain. Existing solutions provide point-to-point secure data transmission and ensure security within a single domain, but are insufficient for distributed data dissemination because of the involvement of multiple cross-domain services.

In this paper, we proposed the EPICS framework based on the notion that data can reside anywhere but are always accompanied by the owner's access policies and a policy enforcement mechanism that protects and controls their disclosure. The novelty of EPICS lies in the transformation of data, traditionally a passive entity, into an active entity protecting itself against unauthorized disclosure and tampering. In case of attacks detected by AB engine, EPICS makes it possible to destroy the data dynamically to prevent disclosure, which is not possible with models treating data as passive entities. We described the design and implementation of the framework and provided a performance and security evaluation using a realistic e-commerce scenario. The proposed framework is compatible with existing service infrastructure and meets the real-time constraints of Web service interactions. The main benefits of using the EPICS framework for data dissemination in composite Web services are as follows:

- It ensures policy-based access control of a client's data based on the enforcement of policies.
- It provides privacy-preserving controlled data dissemination by minimizing the data disclosure.
- It provides context-based adaptable data dissemination based on the use of external environment information (such as trust values, an emergency or an attack context) and the flexibility of the policies.
- It allows consumers sharing data to define access control policies without knowing the data disclosure path and ensure policy enforcement at each endpoint in the interaction.
- It is independent of TTPs and does not require the availability of the data owner to disseminate data once the data is shared with the primary service.
- It reduces the liability of a service for managing a client's data by disclosing to the service only the data authorized by the client's policies.
- It is compatible with the existing service infrastructure such as HTTP-based RESTful services.
- It can be incorporated in any data dissemination application as it is policy language agnostic and supports a wide range of standard authentication and authorization mechanisms.

Future work will include extending the framework implementation with stateful and mutable versions of active bundles, enhancing security of the framework against execution hijack attacks using mechanisms such as code obfuscation and secure enclave execution (cf. Intel SGX [33]), and applying the framework for data dissemination and policy enforcement in different application domains.

References

[1] M. P. Papazoglou and W.-J. Van Den Heuvel, “Service oriented architectures: approaches, technologies and research issues,” *The*

International Journal on Very Large Data Bases, vol. 16, no. 3, pp. 389–415, 2007.

[2] R. Ranchal, A. Mohindra, J. Manweiler, and B. Bhargava, "Radical strategies (RADS) for engineering web-scale cloud solutions," *IEEE Cloud Computing*, vol. 2, no. 5, pp. 20–29, 2015.

[3] M. Azarmi, B. K. Bhargava, P. Angin, R. Ranchal, N. Ahmed, A. Sinclair, M. Linderman, and L. B. Othmane, "An end-to-end security auditing approach for service oriented architectures." in *IEEE Symposium on Reliable Distributed Systems*, 2012, pp. 279–284.

[4] R. Fernando, R. Ranchal, B. An, L. Othmane, and B. Bhargava, "Consumer oriented privacy preserving access control of electronic health records in the cloud," in *Proc. IEEE Conference on Cloud Computing*, 2016.

[5] "Owasp top 10 2013," https://www.owasp.org/index.php/Top_10_2013-Top_10, accessed: Mar 2017.

[6] "Whitehat security. website security statistics report," https://www.whitehatsec.com/resources/, 2014, accessed: Mar 2017.

[7] "Target data breach," https://corporate.target.com/about/shopping-experience/payment-card-issue-FAQ, accessed: Mar 2017.

[8] "Anthem data breach," http://www.anthemfacts.com, accessed: Mar 2017.

[9] D. Muthukumaran, D. O'Keeffe, C. Priebe, D. M. Eyers, B. Shand, and P. R. Pietzuch, "Flowwatcher: Defending against data disclosure vulnerabilities in web applications." in *ACM Conference on Computer and Communications Security*, 2015, pp. 603–615.

[10] "Web Services Policy Framework 1.5," W3C, Tech. Rep., Aug. 2007. [Online]. Available: http://www.w3.org/TR/ws-policy/

[11] F. B. Schneider, "Enforceable security policies," *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 1, pp. 30–50, Feb. 2000.

[12] D. Basin, V. Jugé, F. Klaedtke, and E. Zălinescu, "Enforceable security policies revisited," *ACM Transactions on Information and System Security (TISSEC)*, vol. 16, no. 1, pp. 3:1–3:26, Jun. 2013.

[13] J. H. Saltzer and M. D. Schroeder, "The protection of information in computer systems," *IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.

[14] L. B. Othmane and L. Lilien, "Protecting privacy of sensitive data dissemination using active bundles," in *World Congress on Privacy, Security, Trust and the Management of e-Business*, 2009, pp. 202–213.

[15] R. Ranchal, "Cross-domain data dissemination and policy enforcement," Ph.D. dissertation, West Lafayette, IN, USA, 2015.

[16] R. Ranchal, B. Bhargava, R. Fernando, H. Lei, and Z. Jin, "Privacy preserving access control in service-oriented architecture," in *Proc. IEEE International Conference on Web Services*, 2016, pp. 412–419.

[17] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[18] L. Wang, D. Wijesekera, and S. Jajodia, "A logic-based framework for attribute based access control," in *ACM Workshop on Formal Methods in Security Engineering*, 2004, pp. 45–55.

[19] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *IEEE Symposium on Security and Privacy (S&P'07)*, 2007, pp. 321–334.

[20] M. Lorch, S. Proctor, R. Lepro, D. Kafura, and S. Shah, "First experiences using xacml for access control in distributed systems," in *ACM workshop on XML security*, 2003, pp. 25–37.

[21] H. Krawczyk, "Cryptographic extraction and key derivation: The hkdf scheme," in *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, pp. 631–648.

[22] M. Wand, G. Kiczales, and C. Dutchyn, "A semantics for advice and dynamic join points in aspect-oriented programming," *ACM Transactions on Programming Languages and Systems*, vol. 26, no. 5, pp. 890–910, 2004.

[23] P. Angin, B. Bhargava, and R. Ranchal, "Tamper-resistant autonomous agents-based mobile-cloud computing," in *IEEE/IFIP Network Operations and Management Symposium (NOMS'16)*, 2016.

[24] "Security assertion markup language (SAML)," https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, accessed: June 2015.

[25] S. Cantor and T. Scavo, "Shibboleth architecture," *Protocols and Profiles*, vol. 10, p. 16, 2005.

[26] Y.-Y. Chen, P. A. Jamkhedkar, and R. B. Lee, "A software-hardware architecture for self-protecting data," in *ACM Conference on Computer and Communications Security*, 2012, pp. 14–27.

[27] F. Li, B. Luo, P. Liu, D. Lee, and C.-H. Chu, "Enforcing secure and privacy-preserving information brokering in distributed information sharing," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 6, pp. 888–900, 2013.

[28] S. Pearson and M. C. Mont, "Sticky policies: An approach for managing privacy across multiple parties," *IEEE Computer*, no. 9, pp. 60–68, 2011.

[29] M. Dalton, C. Kozyrakis, and N. Zeldovich, "Nemesis: Preventing authentication & access control vulnerabilities in web applications," in *18th USENIX Security Symposium*, 2009, pp. 267–282.

[30] M. Monshizadeh, P. Naldurg, and V. N. Venkatakrishnan, "Mace: Detecting privilege escalation vulnerabilities in web applications," in *ACM Conference on Computer and Communications Security*, 2014, pp. 690–701.

[31] V. Pappas, V. P. Kemerlis, A. Zavou, M. Polychronakis, and A. D. Keromytis, "Cloudfence: Data flow tracking as a cloud service," in *16th International Symposium on Research in Attacks, Intrusions, and Defenses (RAID'13)*, 2013, pp. 411–431.

[32] Y. Mundada, A. Ramachandran, and N. Feamster, "Silverline: data and network isolation for cloud services," in *3rd USENIX conference on Hot topics in cloud computing (HotCloud'11)*, 2011, p. 13.

[33] M. Hoekstra, R. Lal, P. Pappachan, V. Phegade, and J. Del Cuvillo, "Using innovative instructions to create trustworthy software solutions." in *HASP@ ISCA*, 2013, p. 11.

Rohit Ranchal is with IBM Watson Health, Cambridge, MA. He received his Ph.D. in Computer Science from Purdue University. He is a member of IEEE. (ranchal@us.ibm.com)

Bharat Bhargava is with Computer Science at Purdue University, West Lafayette, IN. He received his Ph.D. in Electrical Engineering from Purdue University. He is a fellow of the IEEE and IETE. (bbshail@purdue.edu).

Pelin Angin is with Computer Science at Purdue University, West Lafayette, IN. She received her Ph.D. in Computer Science from Purdue University. (pangin@purdue.edu)

Lotfi ben Othmane is with Electrical and Computer Engineering at Iowa State University, Ames, IA. He received his Ph.D. in computer science from Western Michigan University. (lbenothmane@gmail.com)