Future Generation Computer Systems 113 (2020) 113–131

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Malicious changeload for the resilience evaluation of self-adaptive authorisation infrastructures

Christopher Bailey, Rogério de Lemos [*]

University of Kent, UK

ARTICLE INFO

Article history:
Received 27 July 2019
Received in revised form 17 December 2019
Accepted 24 June 2020
Available online 1 July 2020

Keywords:
Self-protecting systems
Authorisation infrastructures
Changeload
Insider threats
Autonomic computing
Access control

ABSTRACT

Self-adaptive systems are able to modify their behaviour and/or structure in response to changes that occur to the system, its environment, or even its goals. In terms of authorisation infrastructures, self-adaptation has shown to be a promising solution for enforcing access control policies and subject access privileges when mitigating insider threat. This paper describes the resilience evaluation of a self-adaptive authorisation infrastructure by simulating a case study related to insider threats. As part of this evaluation, a malicious changeload has been formally defined in order to describe scenarios of abuse in access control. This malicious changeload was then used to stimulate self-adaptation within a federated authorisation infrastructure. The evaluation confirmed the resilience of a self-adaptive authorisation infrastructure in handling abuse of access under repeatable conditions by consistently mitigating abuse under normal and high loads. The evaluation has also shown that self-adaptation had a minimal impact on the authorisation infrastructure, even when adapting authorisation policies while mitigating abuse of access.

1. Introduction

Self-adaptive systems are able to modify their behaviour and/or structure in response to changes that occur to the system, its environment, or even its goals [1]. A self-adaptive authorisation infrastructure is a self-adaptive system tailored to adapt, at run-time, access control policies and their enforcement [2]. An important aspect when evaluating the resilience of a self-adaptive authorisation infrastructure is to demonstrate its ability to mitigate abuse in access control.

In this paper, we present a simulation-based approach for evaluating the resilience of self-adaptive authorisation infrastructures under repeatable conditions of system and environmental changes. In our evaluation, in addition to observing performance as a measure of success, we also evaluate the impact of self-adaptation as a means to mitigate potential attacks. Although the goal of self-adaptation is to protect dynamically the authorisation infrastructures from attacks, self-adaptation measures may result in undesirable states, which may include the loss of access to critical resources.

For demonstrating the proposed approach, we evaluate the resilience of the Self-adaptive Authorisation Framework (SAAF) [3–5], whose goal is to make existing authorisation infrastructures self-adaptable. This is achieved by analysing potential attacks once they are detected, and synthesising appropriate mitigation actions depending on the operating conditions of the infrastructure. In terms of SAAF, this would also comprise the generation and deployment of new access control policies at run-time, without any human interference. The premise is that, an organisation can benefit from the properties of dynamic access control without the need to adopt new access control models.

A common way for evaluating self-adaptive systems is through case studies [6]. They are used to represent environment and system changes, and these are expected to stimulate self-adaptation, thus providing the basis for evaluating the impact of adaptation. An advantage of using case studies is that changes can be repeated to stimulate self-adaptation scenarios, thus allowing the evaluation of impact from adaptation in a more consistent way. For the evaluation of a self-adaptive authorisation infrastructure, we use a fictitious case study describing a set of insider attacks within a federated environment. The resilience evaluation of SAAF in a more realistic attack scenario was performed using an ethical on-line game to gather insights on how SAAF would react towards real malicious behaviour, and how malicious users would behave in the presence of self-adaptation [7]. The motivation for the study being reported in this paper is to evaluate whether the proposed solution affects the performance of the overall authorisation infrastructure, and to evaluate the effectiveness of the self-adaptive solution to handle malicious behaviour.

This paper provides two key contributions:

- the definition of a generic approach for evaluating the resilience of self-adaptive authorisation infrastructures. This

[*] Corresponding author.
E-mail address: r.delemos@kent.ac.uk (R. de Lemos).

https://doi.org/10.1016/j.future.2020.06.045
0167-739X/© 2020 Elsevier B.V. All rights reserved.

is demonstrated by deploying SAAF within a federated environment, thus showing how SAAF handles and mitigates malicious behaviour, in the form of insider threats, given the existence of non-cooperating third party organisations;
- the definition of *malicious changeload* that drives stimulation of adaptation in response to malicious behaviour (i.e., abuse of access control). The usefulness of malicious changeload in providing systematic means for specifying repeatable behaviour is demonstrated by evaluating the resilience of SAAF under various operational conditions.

The definition of malicious changeload in the context of self-protecting systems extends that of changeload, defined for the resilience evaluation of architectural-based self-adaptive systems [8], and which considered faults as the only undesirable type of change. In this paper, malicious changeload considers the abuse of access control in federated authorisation infrastructures. Regarding the resilience evaluation of self-protecting systems, to the best of our knowledge this is the first work that uses the notion malicious changeload from resilience benchmarking. This is an important concept if repeatable conditions of system and environmental changes are necessary in the benchmarking of self-protecting systems.

The rest of this paper is structured as follows. In Section 2, we present some basic concepts related to self-adaptive authorisation infrastructures and insider threats. Section 3 positions a motivating case study used as a basis for the evaluation. Section 4 specifies the malicious changeload derived from the case study. Section 5 describes a set of experiments and results that observes the run-time stimulation of malicious changeload, and adaptation of a target system. Section 6 reflects on the outcome of the experiments, along with the benefits and challenges of self-adaptive authorisation. Related work is presented in Section 7. Finally, in Section 8, a summary of the paper is provided in addition to some insights regarding future work.

2. Background

In this background section, we briefly describe the Self-adaptive Authorisation Framework (SAAF) to be used in the resilience evaluation of self-adaptive authorisation infrastructures, we provide some insight to insider threats that are representative of malicious behaviour and defined as part of malicious changeload, and then we provide a brief introduction to resilience benchmark.

2.1. Self-adaptive authorisation framework

The basis our work is the Self-adaptive Authorisation Framework (SAAF), whose goal is to make existing authorisation infrastructures self-adaptable [3–5]. The motivation is that, authorisation infrastructures maintained by organisations can benefit from the properties of dynamic access control without the need to adopt new access control models. SAAF is based on the MAPE-K feedback loop [9], which monitors the distributed services of an authorisation infrastructure, and builds a model of the whole system at run-time (i.e., deployed access control rules, assigned subject privileges, and protected resources). Malicious user behaviour observed by SAAF is mitigated through the generation and deployment of access control policies at run-time, preventing any identified abuse from continuing. Adaptation at the model level ensures that abuse can no longer continue. In addition, model transformation supports the generation of access control policies from an abstract access model. This enables the generation and deployment of policies that are specific to different implementations of access control.

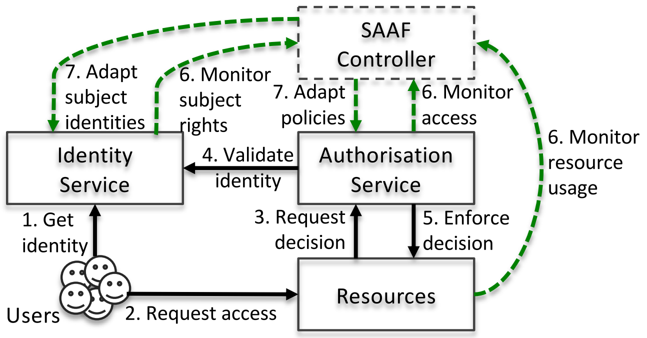

Fig. 1. SAAF conceptual design.

Fig. 1 presents a conceptual design of SAAF in which the services of the authorisation infrastructure, and their interactions, are represented as solid lines, while the autonomic controller and its interactions are represented as dash lines. The role of the SAAF controller is to monitor and adapt the services of the authorisation infrastructure. The interactions between the services are annotated with a sequence of events, and as it can be observed, the autonomic controller does not affect the sequence of events related to the functionality of the identity and authorisation services. The autonomic controller affects only the properties associated these services, respectively, identities and policies, as a means to mitigate malicious behaviour. A major challenge while implementing SAAF is that no single service provides a complete view of access control in terms of what users own in access rights, what access control rules exist, and finally, how users are utilising access rights.

Fig. 2 presents a detailed design of the SAAF controller. The SAAF controller comprises the analysis, which generates new access control models, and the plan, which selects the most appropriate access control model amongst the valid ones. For each identified attack, the SAAF controller selects a subset of solutions applicable to a particular attack, which depends on the type of attack and current access control model. At deployment-time, the SAAF controller is loaded with a set of predefined solutions that mitigate malicious events. The solutions match a finite set of actions that can be performed within the application domain, and are parametric in order to tailor the solutions to specific cases of insider attacks. Given a detected attack, a solution is selected from the following alternatives: (1) increasing, limiting or removing access rights owned by an individual, (2) increasing or limiting the scope of access defined by access control rules, (3) warning the individual(s) of their behaviour, and (4) monitoring the behaviour further. Associated with each solution there is a potential impact. Depending on the type of action invoked, it can cause either negligible or severe consequences to the system (which may be warranted given the severity of the attack detected). Which solution is selected depends on how severe the SAAF controller deems the identified malicious behaviour to be. For example, what is the number of non malicious users impacted negatively by the solution (thus losing access to resources). High severity may be justified for cases when many users are identified as being malicious in relation to specific resources or roles. In this cases, changing access control rules provides a more effective means to responding to attacks.

In its current form, SAAF ensures that whatever adaptations take place will not break conformance to the service's implemented access control methodology — in our case Attribute Based Access Control (ABAC), nor conflict with application domain requirements (e.g., ensure access to business critical systems). To implement ABAC, we provide an identity service referred as

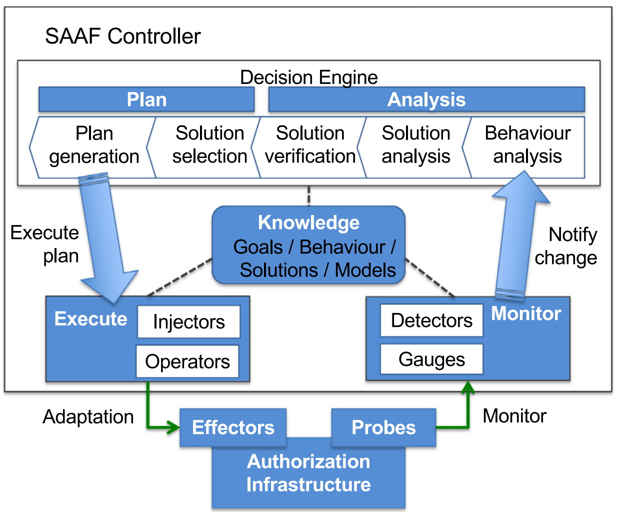

Fig. 2. SAAF controller design.

LDAP [10], which is a directory service commonly used to hold information (including user roles) about users within an organisation. To generate ABAC access control decisions, based on roles owned by users, we use a standalone service authorisation service, known as PERMIS [11].

2.2. Insider threats

Insider threat refers to an organisation's risk of attack by their own users or employees [12]. This is particularly relevant to access control, where the active management of authorisation has the potential to mitigate and prevent users from abusing their own access rights to carry out attacks.

A common characteristic of insider threat is that malicious insiders use their knowledge of their organisation's systems, and their assigned access rights, to conduct attacks. This places a malicious insider in a fortuitous position, whereby the insider (as an authorised user) can cause far greater damage than an external attacker, simply due to their access rights [13]. Such form of attack is representative of the attacks that many organisations consider to be most vulnerable from: the abuse of privileged access rights by the employees of an organisation [14]. Unless additional measures are put into place, malicious insiders can abuse existing security measures, where current approaches fail to robustly adapt and respond to the unpredictable nature of users. Whilst there are a number of novel techniques that enable the detection of insider threat [15–17], there is little research that uses such techniques within an automated setting, like our proposed approach that relies on self-adaptation.

2.3. Resilience benchmarking

A benchmark is a standard procedure that allows characterising and comparing systems or components according to specific characteristics (e.g., performance, dependability) [18]. Previous work on computer benchmarking can be divided in three main areas: performance benchmarking [19], dependability benchmarking [18], and security benchmarking [20].

In the context of self-protecting systems, resilience benchmarking represents a step further since it needs to consider system and environment dynamics, although it is bound to encompass techniques from these previous efforts due to its inherent relation to performance, dependability and security. Given an application domain (that specifies the target systems and its environment), a resilience benchmark should provide generic ways for characterising system behaviour in the presence of changes, allowing to compare similar systems quantitatively. If a system is effective and efficient in accommodating or adjusting to changes, avoiding successful attacks as much as possible and operating as close as possible to its defined goals, it is reasonable to consider the system to be resilient. This capability can be benchmarked by submitting the system to various types of changes, time and resources dedicated to mitigate them, as well as, the impact of this process in the fulfilment of the system goals. As the changes affecting the system may lead to the degradation of its performance, without leading necessarily to security breaches, we need to assess variations in the properties of interest when the system is under varying environmental conditions, in order to characterise its behaviour from a resilience perspective.

3. Case study: LGZLogistics

Given the challenges in obtaining detailed data on actual cases of insider threats, this fictitious case study draws upon several historical cases discussed in the CERT guide to insider threat [12]. In this paper, we consider data theft attacks that are performed to a fictitious logistics company, called *LGZLogistics*, representing a service provider and identity provider within a federated authorisation infrastructure. Malicious behaviour is conducted by disgruntled employees of the logistics company, as well as employees of an external *Trusted Business Partner* (TBP) [12]. The role of a TBP is key to this case study, as it is representative of the relationship a service provider organisation has with an identity provider organisation (e.g., *LGZLogistics* trusts the TBP to provide IT help desk services).

The case study focuses on two areas of insider threat that organisations are highly vulnerable to: the abuse of user access rights by employees of the organisation, and the abuse of access rights by TBP organisations [14].

3.1. Context and architecture

LGZLogistics portrays a small to medium sized company of 1000 employees, ten of which are IT staff that support and administer a set of protected resources. These resources are protected via an instantiated Attribute Based Access Control (ABAC) model, in the form of subject attribute assignments within identity services, and an access control policy within an authorisation service (see Fig. 3).

LGZLogistics maintains a SimpleSAML.php [21] identity service lgzIS to authenticate its subjects (employees), and issue access rights (as credentials). The organisation also maintains a PERMIS standalone authorisation service as [22], to authorise subject access to its resources. These resources include an employee database empDB, and a bespoke logistics tool lgT. The employee database contains personnel information about the logistic company's employees, which is required for general IT help desk enquires.

LGZLogistics uses the authorisation service as to authorise access for its own subjects, as well as subjects from a second offshore contractor organisation (a TBP). *LGZLogistics* trusts the contractor organisation to issue access rights to their subjects, as part of a business contract for providing IT help desk services. As such, the contractor organisation also operates a SimpleSAMLphp identity service conIS that manages its own employees' access rights to the requesting service providers (i.e., *LGZLogistics*). As part of their contract, subjects from the contractor organisation are permitted access to empDB to facilitate help desk duties. Access for subjects from either identity service is obtained as follows:

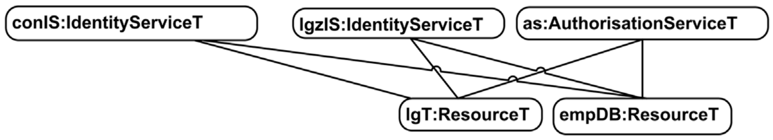

Fig. 3. *LGZLogistics* authorisation infrastructure architecture.

1. A subject attempts to perform an action on a resource;
2. The resource enacts a policy enforcement point (PEP) that requires the subject to authenticate with their identity service (i.e., lgzIS or conIS);
3. Upon authentication, a short term credential is released to the resource's PEP, denoting a signed set of subject attributes (e.g., a SAML assertion [23]);
4. The PEP forwards the subject's issued credential to the authorisation service as, which validates the contents of the credential to ensure attributes released have been issued by a trusted identity provider;
5. If valid, the attributes are used to request access via the authorisation service as; along with the resource, and action to be performed.
6. Lastly, the authorisation service as decides whether to grant access in accordance to its authorisation policy.

3.2. Access control model

LGZLogistics employ an ABAC methodology to protect its resources. As such, an instantiation of ABAC considers the subjects of *LGZLogistics* and the subjects of the contractor organisation. Each set of subjects have a permissible scope of access rights that can be assigned to them.

Fig. 4 defines access in the form of a class diagram. There are five 'permisRole type' attributes [24] (specific to the PERMIS standalone authorisation service) with corresponding values. Subjects are assigned these attributes, which can then be used to invoke permissions.

In addition to the subject attribute assignments and attribute permission assignments, *LGZLogistics* define a set of valid attribute assignment rules (within its authorisation policy). Fig. 5 specifies what attributes an identity provider is trusted to issue on behalf of its employees. For example, *LGZLogistics* identity provider lgzIS is trusted to assign attributes $\langle permisRole, SysAdmin\rangle$, $\langle permisRole, SysAnalyst\rangle$, and $\langle permisRole, Staff\rangle$ to its employees. The contractor organisation identity provider conIS can only assign attributes $\langle permisRole, ContractorSupervisor\rangle$ and $\langle permisRole, Contractor\rangle$.

3.3. Subject behaviour

This section identifies typical subject behaviour for the day to day operations of *LGZLogistics*, as well as a malicious behaviour scenario.

3.3.1. Typical behaviour

The following describes a base line of subject behaviour, detailing the average usage of the authorisation infrastructure likely to occur in the day to day operations of *LGZLogistics*:

- Each staff member requests 'access' to the lgT resource on average two times per day;
- Contractors receive on average fifty calls per day, each call requiring one 'read' access to empDB;
- On average, one in five calls require access to 'modify' the empDB, which can only be performed by a contractor supervisor, systems analyst, or system administrator;
- On average, each system analyst performs ten 'read' requests, and five 'modify' requests per day to the empDB;
- A system admin performs on average one 'read', 'modify', 'delete', and 'create' request per day to the empDB.

3.3.2. Malicious behaviour scenario

The logistics company is victim of an insider attack, largely as a result of a catalyst event [16]. The catalyst event refers to a notification to several key IT workers that they have been selected for job redundancy.[1]

A systems analyst that has been selected for redundancy is unhappy about the decision, and attempts to damage the company in three ways. The first is to attack the empDB resource by randomly corrupting employee records, invoking the permission 'modify' empDB. The second is an attempt to disrupt access to the lgT resource, essentially flooding the resource by initiating numerous authorised sessions. The final attempt is socially motivated, whereby the analyst, who works closely with employees of the contractor organisation, informs them that *LGZLogistics* is going to cancel their contract to cut costs.

A contractor supervisor, now fearing job redundancy, decides to steal data from the empDB resource. The supervisor has links with the internet underground [12], and is aware of anonymous buyers looking for data fit for identify theft. By persuading his peers, three other contractors decide to collaborate in stealing employee information from the empDB, to sell it to the internet underground.

4. Specification of malicious changeload

In this section, we define the changeload related specifically to malicious behaviour in the context of authorisation infrastructures. Essentially, it applies Cámara et al.'s definitions of a changeload model [8] (which is specific to architectural-based self-adaptation) to authorisation infrastructures. Cámara et al.'s changeload model was chosen in order to concretely define the scope of change within an authorisation infrastructure.

Cámara et al. formulated their changeload model primarily to classify system and environment changes that stimulate adaptation [8]. They have defined *changeload* as a set of change scenarios that demonstrates changes, which are: valid within a conventional operational profile, invalid thus stimulating adaptation, or as the result of adaptation.

A *malicious changeload*, in the context of authorisation infrastructures, drives stimulation of adaptation in response to the abuse of access control (i.e., places a system into a non-conventional operational state). It is considered that both environment and system stimulation are capable in generating non-conventional operational states (and are often a by-product of each other), whereby environment change leads to system change.

[1] Instead of generalising an attack as being "harmful", the labelling of an attack in the context of a case study is fundamental for specifying a meaningful malicious changeload.

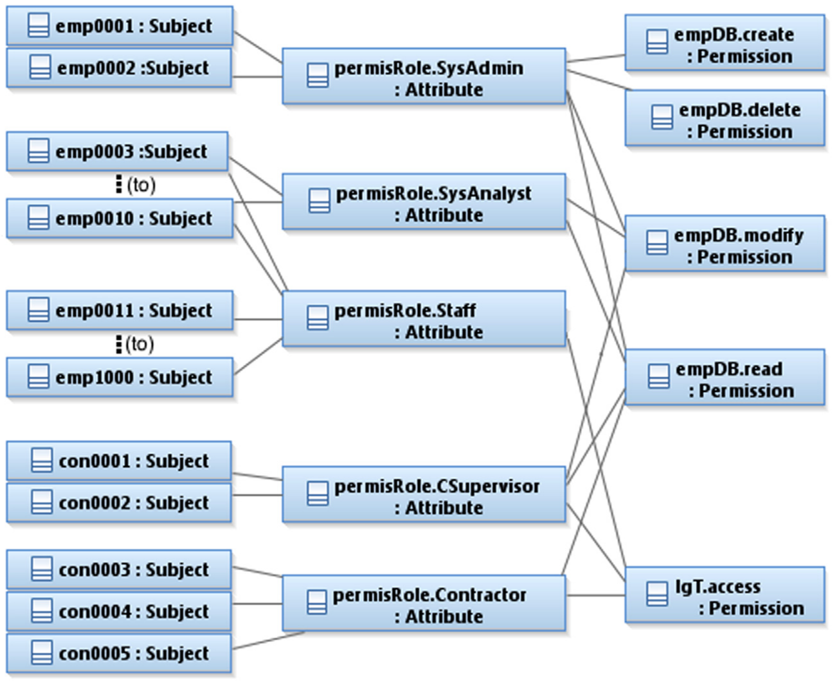

Fig. 4. *LGZLogistics* subject attribute permission assignments.

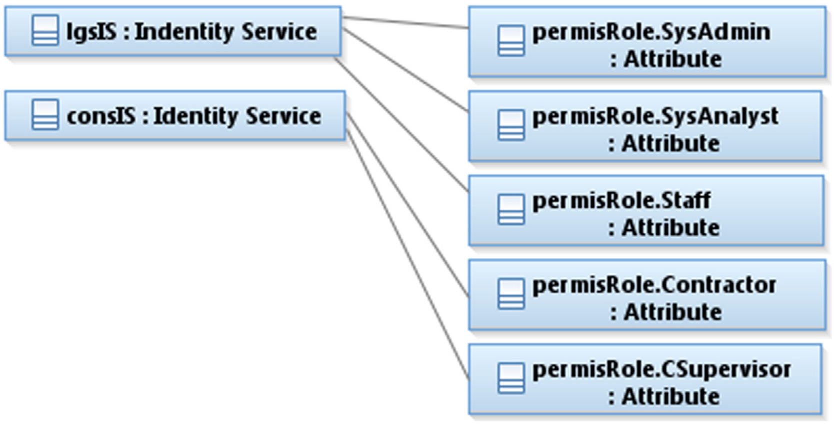
lgsIS : Indentity Service
consIS : Identity Service
permisRole.SysAdmin : Attribute
permisRole.SysAnalyst : Attribute
permisRole.Staff : Attribute
permisRole.Contractor : Attribute
permisRole.CSupervisor : Attribute

Fig. 5. *LGZLogistics* valid attribute assignments.

4.1. System and environment models

For the specification of system and the environment properties, we need to define, respectively, the *system model* and *environment model*. These models enable the specification of system properties that describe an authorisation infrastructure's run-time parameters and workload, and environment properties that characterise the operational conditions imposed on an authorisation infrastructure. The properties contained in both system and environment models are dependent on a given deployment of an authorisation infrastructure and its protected resources.

The *LGZLogistics* authorisation infrastructure is formally defined in terms of an architecture model (Fig. 6). For SAAF, on the other hand, the *access control model* provides the relations between components of an architectural model (i.e., how a subject of an identity service component can access a resource component). Despite this, the use of an architectural model is beneficial for defining properties of a system and the environment. It enables the specification of system properties that describe an authorisation infrastructure's run-time parameters and workload, and properties of the environment that characterises the operational conditions imposed on an authorisation infrastructure. These properties are said to be contained within a system model and environment model, derived from the architecture model.

Example 1. Fig. 6 displays an architecture model of the LGZLogistics authorisation infrastructure, where the set of architectural types is $\mathcal{T} = \{IdentityServiceT, AuthorisationServiceT, ResourceT\}$. Examples of system properties include $\Gamma_{sys}(AuthorisationServiceT) = \{policy, sub\_access\_rate\}$. Examples of environment properties (displayed inside the grey boxes) include $\Gamma_{env}(AuthorisationServiceT) = \{sub\_access\_req\_rate, lgT\_access\_request\_rate\}$, and $\Gamma_{env}(ResourceT) = \{activeSessions, latency\}$.

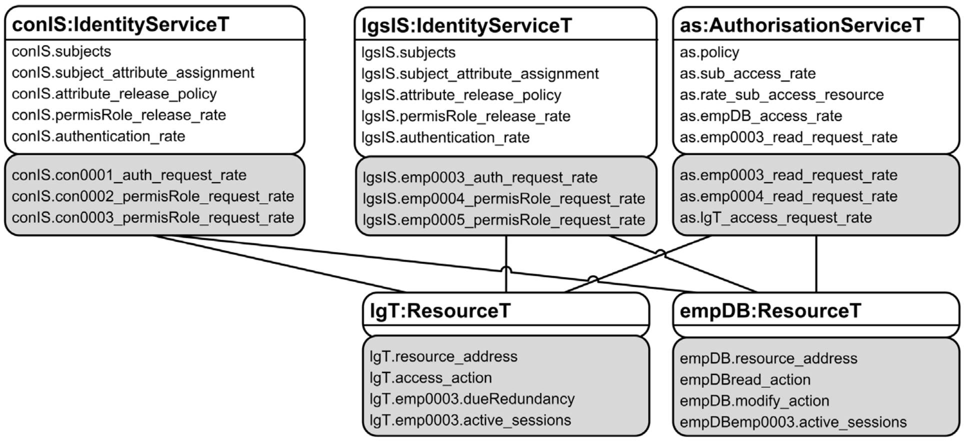
conIS:IdentityServiceT
conIS.subjects
conIS.subject_attribute_assignment
conIS.attribute_release_policy
conIS.permisRole_release_rate
conIS.authentication_rate
conIS.con0001_auth_request_rate
conIS.con0002_permisRole_request_rate
conIS.con0003_permisRole_request_rate
lgsIS:IdentityServiceT
lgsIS.subjects
lgsIS.subject_attribute_assignment
lgsIS.attribute_release_policy
lgsIS.permisRole_release_rate
lgsIS.authentication_rate
lgsIS.emp0003_auth_request_rate
lgsIS.emp0004_permisRole_request_rate
lgsIS.emp0005_permisRole_request_rate
as:AuthorisationServiceT
as.policy
as.sub_access_rate
as.rate_sub_access_resource
as.empDB_access_rate
as.emp0003_read_request_rate
as.emp0003_read_request_rate
as.emp0004_read_request_rate
as.lgT_access_request_rate
lgT:ResourceT
lgT.resource_address
lgT.access_action
lgT.emp0003.dueRedundancy
lgT.emp0003.active_sessions
empDB:ResourceT
empDB.resource_address
empDBread_action
empDB.modify_action
empDBemp0003.active_sessions

Fig. 6. Example architecture model for an authorisation infrastructure.

4.2. System and environment state

The *system state* captures the value of system properties and the execution of services at a given moment in time. For example, the authentication of subjects, the release of subject attributes, the validation of subject attributes, and the authorisation of subject access.

Example 2. In this example, two system attributes are identified that denote execution of the authorisation infrastructure: (i) rate of attribute releases (of any kind) from the identity service lgzIS, and (ii) rate of successful read requests per interval to empDB. For typical execution of the identity service lgzIS, there is a constant throughput of one attribute release per minute. For typical execution of the authorisation service as, there is a constant throughput of three successful access decisions to empDB.

- A = $\langle lgzIS.sub\_attr\_release\_rate,\ as.empDB\_read\_rate\rangle$
- B = $\langle constant\_function,\ constant\_function\rangle$
- $V_A = \langle 1,\ 3\rangle$
- $V_B = \langle \theta_{lgzIS}(t) = 1/min,\ \theta_{as}(t) = 3/min\rangle$

The *environment state* captures operational conditions of external systems and users that interact with the authorisation infrastructure. This includes conditions, such as, the rate of access requests by subjects, or number of active sessions in resources. Building a perception of environment state is essential to identifying states that exhibit malicious behaviour (e.g., subjects exhibiting excessive deviation from normal activity).

Example 3. In this example, three environment attributes are identified that denote operational conditions: (i) the number of active sessions in empDB, (ii) the rate of authentication requests made by all subjects against the identity service lgzIS, and (iii) the rate of access requests to access the resource empDB to authorisation service as. The values associated with these operational conditions are, respectively, five active sessions, a throughput of one authentication requests per minute, and a throughput of three access requests per minute.

- A = $\langle empDB.active\_sessions, lgzIS.sub\_authentications\_req\_rate, as.empDB\_read\_req\_rate\rangle$
- B = $\langle constant\_function,\ constant\_function,\ constant\_function\rangle$
- $V_A = \langle 5,\ 1,\ 3\rangle$
- $V_B = \langle \theta_{empDB}(t) = 5,\ \theta_{lgzIS}(t) = 1/min,\ \theta_{as}(t) = 3/min\rangle$

4.3. Operational profiles

An authorisation infrastructure can be in one of two types of states, a conventional operational state, or non-conventional operational state [8]. A *conventional operational state* refers to a state where there is no ongoing abuse of access rights.

Example 4. A conventional operational profile is described as a set of states that does not contain any known patterns of abuse of access (i.e., violations).

$COP = \{s \in S \mid s \models \neg(empDBViolation)\}$

A violation describes a predicate, that if true, denotes malicious behaviour within the environment state.

A *non-conventional operational state* refers to a state where there is ongoing abuse of access rights.

Example 5. A non-conventional operational profile is described as a set of states that contain one or more occurrences of malicious behaviours. In this case, a violation (empDBViolation) denotes a specific violation in access to the empDB.

$NCOP_{empDBViolation} = \{s \in S \mid s \models empDBViolation\}$

The violation empDBViolation is focused on determining if any particular subject is requesting access to the empDB resource in a rapid manner. A subject that requests access to empDB at a rate ($subAccessReqRate_{empDB}$) greater than a maximum prescribed rate ($maxSubAccessReqRate_{empDB}$) is considered to be malicious.

$$empDBViolation = subAccessReqRate_{empDB} > maxSubAccessReqRate_{empDB}$$

4.4. Change types and changes

Change types affect either identity services or authorisation services, which are characterised as part of an authorisation infrastructure, or its environment, consisting mainly of protected resources. *Change types* are defined as a vector of 'attributes'[2] that describe a change and the dynamics of a change.

In application to authorisation infrastructures, a change type describes an observable event within identity services, authorisation services, or protected resources. Essentially, the observation of such change will have a consequence on properties contained within the system and environment model.

[2] Note that the domain of authorisation infrastructures refer to 'attributes' as a piece of information that expresses something about the subject or the current conditions within an accessed resource. This is not to be confused with attributes of a formal model of change (i.e., changeload).

Example 6. In the following, several examples of low level environment change types are exemplified, depicting the process of a subject requesting access to a resource. The instantiation of these change types will have a consequence on one or more environment properties.

(i) **Authentication request type** captures (within an identity service) the identity service receiving a request for authentication of a user.

$$\textbf{auth\_request\_type} = \langle identity\_service, \langle authRequest(username, password)\rangle, \langle event\rangle\rangle$$

(ii) **Attribute release request type** captures a request received by the identity service made by a service provider for a subject's identity attributes.

$$\textbf{attr\_release\_request\_type} = \langle identity\_service, \langle attrRequest(identity, \langle iAttribute\_type_1, \dots, iAttribute\_type_n\rangle, target)\rangle, \langle event\rangle\rangle$$
$$\textbf{identity} = \langle identity\_type, identity\_value\rangle$$

It describes the request of a service provider (target) for a set of identity attributes (iAttribute_type) that have been issued to a subject (identity). The set of attributes requested can be a null set, therefore requesting all releasable attribute types for the subject identity. Note that an identity is referred to by a type of identifier and a value. For example, identity_type may be an LDAP distinguished name.

(iii) **Credential validation request type** is the receipt of a credential validation request within an authorisation service.

$$\textbf{cred\_validation\_request\_type} = \langle auth\_service, \langle valRequest(identity, issuer, \langle iCondition_1, \dots, iCondition_n\rangle, \langle iAttribute_1, \dots, iAttribute_n\rangle)\rangle, \langle event\rangle\rangle$$

It contains attributes issued by a given identity provider (issuer) for a requesting subject, detailing a request to validate a subject's attributes. A set of conditions specified by the issuer can also be contained, whereby a condition refers to a type/value tuple, such as a single use declaration, or validity time. A credential validation request can either push the subject's known attributes, or (given a null set) require the authorisation service to pull the subject's known attributes from the subject's identity provider. In the latter case, the authorisation service invokes an attribute release request.

(iv) **Access request type** is the request, received by an authorisation service, and made by a resource on behalf of a subject.

$$\textbf{access\_request\_type} = \langle auth\_service, \langle accessRequest(\langle iAttribute_1, \dots, iAttribute_n\rangle, resource, action, \langle rAttribute_1, \dots, rAttribute_n\rangle, identity)\rangle, \langle event\rangle\rangle$$
$$\textbf{iAttribute} = \langle iAttribute\_type, iAttribute\_value\rangle$$
$$\textbf{rAttribute} = \langle rAttribute\_type, rAttribute\_value\rangle$$

The request contains (1) the subject's identity attributes (iAttribute), (2) the resource and action to be carried out by the subject, (3) a set of resource environment attributes (rAttribute) provided by the resource (e.g., $\langle timeOfDay\_type, 11am\rangle$), and (4) the requesting subject's identity.

(v) **Resource action step type** models an action that has occurred within any protected resource. The type is generic as resources are generally unique to the organisation and their purpose, unlike with an authorisation service type that exists to fulfil access control requirements.

$$\textbf{resource\_action\_step\_type} = \langle resource, \langle rAttribute\rangle, \langle step\_function\rangle\rangle$$

The type identifies an attribute modification by means of a step function. The attribute modified (*rAttribute*) is a tuple of type/value, and can represent anything modelled within the resource type, be it generic or specific. For example, this type could be instantiated to increase the total amount of bandwidth consumed by a subject, within a given session.

Example 7. In the following, several examples of system change types are described, conveying the system's response to a subject requesting access.

(i) **Authentication decision type** captures the consequence (within an identity service) of an authentication request being responded to.

$$\textbf{auth\_decision\_type} = \langle identity\_service, \langle authDecision(auth\_request)\rangle, \langle event\rangle\rangle$$

(ii) **Attribute release type** is the consequence of an attribute release request, within an identity service.

$$\textbf{attr\_release\_type} = \langle identity\_service, \langle attrRelease(attr\_release\_request)\rangle, \langle event\rangle\rangle$$
$$\textbf{attrRelease(attr\_release\_request)} = \langle issuer, identity, \langle iCondition_1, \dots, iCondition_n\rangle \langle iAttribute_1, \dots, iAttribute_n\rangle\rangle$$

It details the releasable identity attributes (iAttribute) as a tuple stating the type of identity attribute and its value. Identity attributes are released along with the issuer of the attributes (i.e., an ID of the identity provider or individual whom assigned these attributes), the identity of the subject (i.e., a persistent ID), and a set of conditions. Conditions are a type value tuple, detailing the use of the released attributes. For example, a condition may assert the released attributes may only be used once, or can only be used in a given time frame.

(iii) **Credential validation type** is the consequence of a credential validation request, within an authorisation service.

$$\textbf{cred\_validation\_type} = \langle auth\_service, \langle valCredentials(cred\_validation\_request)\rangle, \langle event\rangle\rangle$$
$$\textbf{valCredentials(cred\_validation\_request)} = \langle viAttribute_1, \dots, viAttribute_n\rangle$$

It returns valid attributes ($viAttribute_i$) for a subject if the provided iAttributes conform to the authorisation service's credential validation policy. These are effectively the same as identity attributes, however, they are referred to as valid because an authorisation service has checked that the identity service is trusted to issue them.

(iv) **Access decision type** is the consequence of an access request, providing a decision based on the attributes within an access request, and an authorisation service's access control policy.

access_decision_type $= \langle auth\_service, \langle accessDecision(access\_request)\rangle, \langle event\rangle\rangle$

accessDecision(access_request) $=$ *decision*

A *change* is an instantiation of a change type. Once enacted, the perception of state (either system or environment) has changed.

Example 8. In this example, the environment change types, provided in Example 6, are instantiated into actual changes relevant to the *LGZLogistics* case study.

(i) **Authentication request** change defines the attributes received as part of a request for authentication within identity provider lgzIS.

emp0003_auth_request $= (auth\_request\_type, lgzIS, \langle authRequest(emp0003, password)\rangle, \langle event\rangle, 1373463234, 0)$

(ii) **Attribute release request** change could be requested by a resource's policy enforcement point at the time of authentication. However, it is also used by authorisation services as part of a credential validation request (depending on its configuration). The request states a set of identity attribute types (i.e., attribute types that can exist with an identity, such as e-mail), and an identity. The identity, shown as a set of numerical and alphabetical characters, is a privacy protected persistent id (PID), however, equally could denote a non-privacy protected identifier (e.g., an e-mail address).

emp0003_permisRole_request $= (attr\_release\_request\_type, lgzIS, \langle attrRequest(\langle pid, bxu915810faa4910\rangle, \langle permisRole\rangle, empDB)\rangle, \langle event\rangle, 1373463236, 0)$

(iii) **Credential validation request**

emp0003_cred_validation_request $= (cred\_validation\_request\_type, as, \langle valRequest(\langle PID, bxu915810faa4910\rangle, lgzIS, \langle\langle notOnOrBefore, 1373462240\rangle, \langle notOnOrAfter, 1373473240\rangle\rangle, \langle\langle permisRole, SysAnalyst\rangle\rangle)\rangle, \langle event\rangle, 1373463240, 0)$

This change portrays a credential validation request being received in authorisation service as. A privacy protected identity is provided, along with the authenticating identity service (lgzIS), in order for the authorisation service as to validate the subject's released attribute $\langle permisRole, SysAnalyst\rangle$. The two conditions ($notOnOrBefore$, $notOnOrAfter$) state the validity of the released attribute.

(iv) **Access request** change captures the subject emp0003, with assigned identity PID: $bxu915810faa4910$ and attribute $\langle permisRole, SysAnalyst\rangle$, requesting to execute 'read' action on the resource empDB (Payroll service). The change is observed as the receipt of request within the authorisation service as.

emp0003_empDB_request $= (access\_request\_type, as, \langle accessRequest(\langle\langle permisRole, SysAnalyst\rangle\rangle, empDB, read, \langle NULL\rangle, \langle pid, bxu915810faa4910\rangle)\rangle, \langle event\rangle, 1373463245, 0)$

(v) **Resource action step** is a change that increments the total bandwidth the subject emp0003 has used within an active session, specifically, to the empDB resource. This change indicates a step change to the attribute activeSessions[emp0003].bandwidth, which contains the subject's current used bandwidth for their active session; the change increases 200kb bandwidth to 800kb.

incr_emp0003_empDB_bandwitdh $= (resource\_action\_step\_type, empDB, \langle 200kb\rangle, \langle \theta_{empDB}(t) = activeSessions[emp0003].bandwidth + 600kb\rangle, 1373465245, 0)$

Example 9. This example provides instantiations of the system change types identified in Example 7. An instantiation of a change type is defined as (change_type, srcinst, V_A, V_B, time, duration).

(i) **Authentication decision** change indicates the subject emp0003 authenticating themselves against the identity service lgzIS, which is classified by an event. The change is coupled with the attributes of the request, in order to provide the decision. The decision generates a grant and the generation of a new session for the subject within the lgzIS identity service.

emp0003_auth $= (auth\_decision\_type, is, \langle authDecision(emp0003\_auth\_request)\rangle, \langle event\rangle, 1373463235, 0)$

authDecision(emp0003_auth_request) $=$ *success*

(ii) **Attribute release** change indicates a change observed at the lgzIS identity service, where a resource empDB has requested the attribute release of attribute type 'permisRole' of the subject emp0003. Identity service lgzIS releases a tuple of attributes that match the request from the resource for the required subject. In this case, it releases $\langle permisRole, SysAnalyst\rangle$. The time indicates the time and date of the attribute release, and as this is not associated to any session, the duration is instant (0).

emp0003_permisRole_release $= (attr\_release\_type, is, \langle attrReleasee(emp0003\_permisRole\_request)\rangle, \langle event\rangle, 1373463239, 0)$

attrRelease(emp0003_permisRole_request) $= \langle\langle permisRole, SysAnalyst\rangle\rangle$

(iii) **Credential validation** change indicates a change observed within the authorisation service as, being a credential validation. Credential validation either validates the provided attributes in the request, or pulls the subject's attributes from the identity provider. In this example, the authorisation service has validated the pushed attributes, asserting that $\langle permisRole, SysAnalyst\rangle$ is valid.

emp0003_cred_validation $= (cred\_validation\_type, as, \langle valCredentials(emp0003\_cred\_validation\_request)\rangle, \langle event\rangle, 1373463240, 0)$

valCredentials(emp0003_cred_validation_request) $= \langle\langle permisRole, SysAnalyst\rangle\rangle$

(iv) **Access decision** change indicates the authorisation servic as receiving a request and generating an authorisation decision based on the attributes of the request. The authorisation service has granted the request. The change is instant and is only relevant for the specific request, therefore there is no duration.

$$\begin{aligned}&\mathbf{emp0003\_empDB\_grant}\\&\quad = (access\_decision\_type, as,\\&\qquad \langle accessDecision(emp0003\_empDB\_request)\rangle,\\&\qquad \langle event\rangle,\\&\qquad 1373463245, 0)\\&\mathbf{accessDecision(emp0003\_empDB\_request)} = permit\end{aligned}$$

4.5. Scenarios and changeload

A *scenario* encompasses a set of changes over time, in light of a set of system goals, and a given state. It is used to formally describe malicious behaviour over time, such as a progression of violations. Key to a scenario is the definition of goals that should be fulfilled as the system undergoes change. In relation to detecting and mitigating malicious behaviour, a goal may refer to an error margin in detecting attacks, maximum response time to resolving attacks, and impact of attacks before required policy changes.

A *base scenario* defines a state that conforms to a system's conventional operational profile, i.e., a state where no known malicious behaviour is present. Such an assumption requires that only malicious behaviour intended to be stimulated against the base scenario can be evaluated, as it is not possible to rule out the existence of unknown malicious behaviour. The *LGZLogistics* case study has several valid base scenarios. For example, a base scenario could describe the typical workload during a normal business day within *LGZLogistics*. This includes a typical definition of criteria and assignment of access. Alternatively, it could represent the initial deployment of its authorisation infrastructure (i.e., no workload).

Example 10. A base scenario for the LGZLogistics case study portrays the authorisation infrastructure, and its expected system and environment properties, for a typical work day. For simplicity, only the system and environment properties relating to subject access are described. *DailyAccess* captures a base scenario of a typical system state relating to access decisions, and a typical environment state relating to access requests.

$$DailyAccess_{BaseScenario} = (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, \emptyset)$$

Each element of the base scenario tuple is expressed below. The system state combines properties that indicate the run-time parameters of services (e.g., authorisation policies), as well as system workload properties (e.g., rate of permitted access for a given subject). Both the system and environment states are defined in conformance to LGZLogistic's access control model (Section 3.2), and its definition of typical subject behaviour (Section 3.3).

$$\begin{aligned}&SysAccess^{t}_{state} =\\&\quad \langle\langle as.policy,\ lgzIS.emp0003.attr,\ conIS.con0003.attr,\\&\qquad as.emp0003.empDB.read,\ as.con0003.empDB.read\rangle,\\&\quad \langle constant\_function,\ constant\_function,\ constant\_function,\\&\qquad constant\_function,\ constant\_function\rangle,\\&\quad \langle AP_1,\ \{Staff, SysAnalyst\},\ \{Contractor\},\ 0.6,\ 1.25\rangle,\\&\quad \langle \theta_{as.policy}(t) = AP_1,\ \theta_{as}(emp0003.permisRole) = \{Staff, SysAnalyst\},\\&\qquad \theta_{as}(con0003.permisRole) = \{Contractor\},\ \theta_{as}(t) = 0.6/min,\\&\qquad \theta_{as}(t) = 1.25/min\rangle\\&\quad \rangle\end{aligned}$$

$SysAccess^{t}_{state}$ defines the state of access control, including policies and attribute assignments. For example, subject emp0003 from identity service lgzIS, is assigned attributes $\langle permisRole, \{Staff, SysAnalyst\}\rangle$. AP_1 denotes a PERMIS authorisation policy that implements the valid attribute assignment rules in Fig. 5, and attribute permission assignments in Fig. 4. Note, the system state defined is not exhaustive, rather it focuses only on: system properties that define the current state of access; provides an example of attribute assignment to a subject from each identity provider; and an example rate of permitted access to the empDB resource.

$$\begin{aligned}&EnvReqs^{t}_{state} =\\&\quad \langle\langle as.SysAdmin.empDB.Read,\ as.SysAdmin.empDB.Modify,\\&\qquad as.SysAdmin.empDB.Create,\ as.SysAdmin.empDB.Delete,\\&\qquad as.SysAnalyst.empDB.Read,\ as.SysAnalyst.empDB.Modify,\\&\qquad as.ContractorSupervisor.empDB.Read,\\&\qquad as.ContractorSupervisor.empDB.Modify,\\&\qquad as.Contractor.empDB.Read,\ as.Staff.logisticsTool.Access\rangle,\\&\quad \langle constant\_function,\ constant\_function,\ constant\_function\\&\qquad constant\_function,\ constant\_function,\ constant\_function\\&\qquad constant\_function,\ constant\_function,\ constant\_function\\&\qquad constant\_function\rangle,\\&\quad \langle 0.8,\ 0.4,\ 0.2,\ 0.2,\ 4.8,\ 1.6,\ 6.7,\ 6.7,\ 3.8,\ 20.0\rangle,\\&\quad \langle \theta_{as}(t) = 0.8/min,\ \theta_{as}(t) = 0.4/min,\\&\qquad \theta_{as}(t) = 0.2/min,\ \theta_{as}(t) = 0.2/min,\\&\qquad \theta_{as}(t) = 4.8/min,\ \theta_{as}(t) = 1.6/min,\ \theta_{as}(t) = 6.7/min,\\&\qquad \theta_{as}(t) = 6.7/min, \theta_{as}(t) = 3.8/min,\ \theta_{as}(t) = 20.0/min\rangle\\&\quad \rangle\end{aligned}$$

$EnvReqs^{t}_{state}$ defines the state of the environment with regards to subjects requesting access. The environment properties identified in the state condition refer to collective behaviour per attribute per permission. For example, for subjects requesting access to 'read' empDB, with attribute $\langle permisRole, SysAdmin\rangle$, a rate of 0.8 requests per minute is observed. As there are two subjects with this attribute (Fig. 4), it is assumed that each subject has an average rate of 0.4 requests per minute (i.e., one request every 150 s).

The fixed goals (G_f) define the conditions that must be maintained within the authorisation infrastructure, regardless of change. Ultimately, a goal requires a system to be brought out of a non-conventional operational state (once identified). However, goals also focus on a wider scope of conditions that attempt to ensure that only necessary adaptations are taken, once in a non-conventional state. The following describes a set of goals relevant to the LGZLogistics case study:

- Probability of 99% that all instances of known violation types are detected;
- Probability of 90% that violations are mitigated through subject adaptation;
- Probability of 99% that all adaptations performed exhibit a lower cost than current and unmitigated violations, to the organisation.

Probabilities cited are pseudo values that indicate LGZLogistics requirements for mitigation. However, an accurate probability can only be achieved through rigorous benchmarking of the scenario in an off-line environment [25]. In any case, probabilities defined are specific to the deployment environment, and configuration of the authorisation infrastructure.

Cámara et al. state that a *change scenario* represents a set of changes applied to a base scenario [8]. As such, a change scenario instantiates a set of changes within the authorisation infrastructure when it is in a particular state. Through the application of change scenarios, it is expected to bring the authorisation infrastructure into an non-conventional state, where the authorisation infrastructure's fixed goals can be evaluated.

Example 11. The following sequence of changes describes subject *emp0003* accessing the empDB resource.

$$\begin{aligned}\mathbf{c_1} &= (\mathit{access\_request\_type}, \mathit{as}, \\ &\quad \langle \mathit{request}(\langle\langle \mathit{permisRole}, \mathit{SysAnalyst}\rangle\rangle, \mathit{empDB}, \\ &\quad \mathit{read}, \langle \mathit{NULL}\rangle, \mathit{pid} = \mathit{bxu}915810\mathit{faa}4910)\rangle, \langle \mathit{event}\rangle, 5, 0) \\ \mathbf{c_2} &= (\mathit{access\_request\_type}, \mathit{as}, \\ &\quad \langle \mathit{request}(\langle\langle \mathit{permisRole}, \mathit{SysAnalyst}\rangle\rangle, \mathit{empDB}, \\ &\quad \mathit{read}, \langle \mathit{NULL}\rangle, \mathit{pid} = \mathit{bxu}915810\mathit{faa}4910)\rangle, \langle \mathit{event}\rangle, 10, 0) \\ \mathbf{c_3} &= (\mathit{access\_request\_type}, \mathit{as}, \\ &\quad \langle \mathit{request}(\langle\langle \mathit{permisRole}, \mathit{SysAnalyst}\rangle\rangle, \mathit{empDB}, \\ &\quad \mathit{read}, \langle \mathit{NULL}\rangle, \mathit{pid} = \mathit{bxu}915810\mathit{faa}4910)\rangle, \langle \mathit{event}\rangle, 15, 0)\end{aligned}$$

$\mathbf{c_1}$ describes a single access request for emp0003, identified by privacy protected id (PID) bxu915810faa4910, using attribute $\langle \mathit{permisRole}, \mathit{SysAnalyst}\rangle$, to access empDB. Thereafter, at 5 s intervals, new changes are instantiated, whereby the request for access to empDB is repeated. As a result, the subject affects a number of environment properties associated to the system, namely, the subject's rate of access to empDB.

The sequence of changes describes a rapid rate of access to the empDB resource (labelled as $C_{rapidAccess}$). With the sequence of changes defined, it is applied to the base scenario with the following notation:

$$\begin{gathered} \mathit{ExpectedToRapidUsage}_{ChangeScenario} = \\ (\mathit{SysAccess}^t_{state}, \mathit{EnvReqs}^t_{state}, G_f, C_{rapidAccess}) \\ C_{rapidAccess} = \{c_1, c_2, c_3\} \end{gathered}$$

Cámara et al. formulated their changeload model primarily to classify system and environment change that stimulates adaptation. As such, a malicious changeload, in the context of authorisation infrastructures, is one that drives stimulation of adaptation in response to the abuse of access control (i.e., places a system into a non-conventional operational state). It is considered that both environment and system stimulation are capable in generating non-conventional operational states (and are often a by-product of each other), whereby environment change leads to system change.

4.6. Violations

A set of violations are defined as the upper bounds of abnormal behaviour, based on the normal behaviour described in the *LGZLogistics* case study. It is assumed that historical data of subject behaviour (if present), coupled with an expert approach, is used to define relevant violations. With reference to the Self-Adaptive Authorisation Framework (SAAF), each violation is defined as a trigger rule (with an associated cost).

The following violations detail patterns of access against *LGZLogistic's* resources, regarding short term and long term rates of access invoking certain permissions. For each violation a maximum rate of access is defined, whereby a short term rate refers to subject access within a minute interval, and a long term rate refers to subject access within a 10 min interval (to simulate a scaled measure of prolonged change).

For example, violation empDBShortRead and violation empDBLongRead classifies malicious behaviour as any subject successfully requesting access to invoke the 'read' action on empDB, at a greater rate than a max allowable. A constraint is applied to the violation, whereby this violation only applies to subjects whom do not have the attribute $\langle \mathit{permisRole}, \mathit{SysAdmin}\rangle$.

$$\begin{aligned}&\texttt{empDBShortRead} = \\ &\quad (\texttt{subAccessReqRate}_{\texttt{empDB.read}} > \\ &\quad \texttt{MaxAccessReqShortRate}_{\texttt{empDB.read}}) \wedge \\ &\quad (\texttt{subAttribute} <> \langle \mathit{permisRole}, \mathit{SysAdmin}\rangle)\end{aligned}$$

$$\begin{aligned}&\texttt{empDBLongRead} = \\ &\quad (\texttt{subAccessReqRate}_{\texttt{empDB.read}} > \\ &\quad \texttt{MaxAccessReqLongRate}_{\texttt{empDB.read}}) \wedge \\ &\quad (\texttt{subAttribute} <> \langle \mathit{permisRole}, \mathit{SysAdmin}\rangle)\end{aligned}$$

For violations empDBShortModify and empDBLongModify, malicious behaviour is classified in terms of any subject successfully requesting access to invoke the 'modify' action on empDB, at a greater rate than a max allowable. As with the aforementioned violations, a constraint is applied meaning the violation is only applicable to subjects who do not have the attribute $\langle \mathit{permisRole}, \mathit{SysAdmin}\rangle$.

$$\begin{aligned}&\texttt{empDBShortModify} = \\ &\quad (\texttt{subAccessReqRate}_{\texttt{empDB.modify}} > \\ &\quad \texttt{MaxAccessReqShortRate}_{\texttt{empDB.modify}}) \wedge \\ &\quad (\texttt{subAttribute} <> \langle \mathit{permisRole}, \mathit{SysAdmin}\rangle)\end{aligned}$$

$$\begin{aligned}&\texttt{empDBLongModify} = \\ &\quad (\texttt{subAccessReqRate}_{\texttt{empDB.modify}} > \\ &\quad \texttt{MaxAccessReqLongRate}_{\texttt{empDB.modify}}) \wedge \\ &\quad (\texttt{subAttribute} <> \langle \mathit{permisRole}, \mathit{SysAdmin}\rangle)\end{aligned}$$

Violation empDBShortDelete classifies malicious behaviour in a subject rapidly gaining access to delete entries within the emphDB resource.

$$\begin{aligned}&\texttt{empDBShortDelete} = \\ &\quad (\texttt{subAccessReqRate}_{\texttt{empDB.delete}} > \\ &\quad \texttt{MaxAccessReqShortRate}_{\texttt{empDB.delete}})\end{aligned}$$

Violation lgTShortAccess classifies malicious behaviour of subjects rapidly accessing the lgT (logistic tool) resource.

$$\begin{aligned}\texttt{lgTShortAccess} = (&\texttt{subAccessReqRate}_{\mathsf{lgT}} > \\ &\texttt{MaxAccessReqFastRate}_{\mathsf{lgT}})\end{aligned}$$

Violation empDBTransaction is slightly different, whereby it classifies a transaction of non-conventional change. This type of violation denotes a pattern whereby a rate of transactional requests are compared against a maximum rate. The violation requires an environment property that measures the rate of access requests, by a subject, in performing a read action succeeded by a modify action against empDB. Basically, it aims to identify subjects who rapidly read and write to the empDB resource.

$$\begin{aligned}&\texttt{empDBTransaction} = \\ &\quad (\texttt{subAccessReqRate}_{\texttt{empDB.readModifyTransaction}} > \\ &\qquad \texttt{MaxAccessReqLongRate}_{\texttt{empDB.readModifyTransaction}})\end{aligned}$$

A final violation, albeit by contrast does not capture subject activity directly, is dueRedundancy. This violation is a consequence of a change made within the empDB resource, indicating that a subject has been marked for job redundancy. A subject facing the prospect of redundancy is seen as a potential risk, and as such, a violation is used to increase the impact a subject has on an organisation. This is viewed as a motivator for adaptation, as when combined with previously identified violations, the subject's activity may now warrant adaptation.

$$\texttt{dueRedundancy} = (\texttt{subDueRedundancy} == \mathit{true})$$

4.7. Identifying change types and change

To stimulate violations within the context of the *LGZLogistics* case study, it is necessary to identify properties of interest and the change types that will impact such properties. For this specific case study, only environment properties are considered. These are properties that concern subject activity that cannot be directly controlled (e.g., a subject's rate of access requests).

4.7.1. Environment properties

For each violation, and for each subject, there exists a set of environment properties that measure the extent of change in the environment. Many environment properties represent composite properties of subject-related changes over time. In reference to SAAF, these properties are dynamically created as mutable attributes within SAAF's behaviour model, and updated through the observation of environment change via probes.

For example, `empDBShortRead` asserts that if a subject's access rate in requesting a read on `empDB` (who is not a *SysAdmin*) goes beyond a maximum number of requests within a minute interval, a violation has occurred. To measure against this violation, an environment property of $\texttt{as.subject.AcessReqRate}_{\texttt{empDB.read}}$ is required (e.g., $\texttt{as.emp0003.AccessReqRate}_{\texttt{empDB.read}}$).

4.7.2. Change types and changes

Once environment properties are identified it is necessary to select relevant change types (and changes) that result in a non-conventional operational state. For example, a non-conventional operational profile that contains the violation `empDBShortRead` is realised through a succession of changes, whereby a single subject successfully requests access to 'read' `empDB`. The violation occurs when a subject, e.g., emp0003, has performed a number of **Access request** change type, and is permitted by an **Access decision** change type.

The **Access request** change type is the result of a number of sequential changes, such as the subject first authenticating with their identity provider, requesting a release of attributes as credentials, and validation of attributes. In this instance, these changes need to be realised before a subject performs an **Access request** change type.

All but one violation described for the *LGZLogistics* case study is triggered by an **Access request** change type. The violation `dueRedundancy` is triggered by a **Resource action step** change, whereby an environment property for a given subject indicates a subject is due for job redundancy (e.g., `empDB.emp0003.isSetForRedundancy`).

4.8. Malicious changeload

Using the *LGZLogistics* malicious behaviour scenario, the following set of change scenarios are defined. Together they represent the malicious changeload for the case study. There are seven change scenarios defined within the malicious changeload, representative of the case study's malicious behaviour scenario, and each change scenario is applicable to the base scenario.

The first change scenario ($setSubjectRedundancies_{ChangeScenario}$) considers a set of resource changes relevant to `empDB`, where changes identify four system analysts are to be made redundant (`dueRedundancy`).

$$
\begin{aligned}
&setSubjectRedundancies_{ChangeScenario} = \\
&\quad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{setRedundancies}) \\
&C_{setRedundancies} = \{ \\
&\quad \mathbf{c_1} = (resource\_action\_step\_type, empDB, \\
&\qquad \langle emp0003.Redundancy \rangle, \\
&\qquad \langle emp0003.Redundancy = true \rangle, 0, 0) \\
&\quad \mathbf{c_2} = (resource\_action\_step\_type, empDB, \\
&\qquad \langle emp0004.Redundancy \rangle, \\
&\qquad \langle emp0004.Redundancy = true \rangle, 0, 0) \\
&\quad \mathbf{c_3} = (resource\_action\_step\_type, empDB, , \\
&\qquad \langle emp0005.Redundancy \rangle, \\
&\qquad \langle emp0005.Redundancy = true \rangle, 0, 0) \\
&\quad \mathbf{c_4} = (resource\_action\_step\_type, empDB, \\
&\qquad \langle emp0006.Redundancy \rangle, \\
&\qquad \langle emp0006.Redundancy = true \rangle, 0, 0)\}
\end{aligned}
$$

The second scenario ($emp0003ReadModify_{ChangeScenario}$) describes a malicious change scenario resulting in violations `empDBLongReadModify`, `empDBLongRead`, and `empDBLongModify`, whereby subject emp0003 persistently reads and modifies records in the `empDB` resource, every four seconds. An arbitrary function δ is defined in order to calculate the time at which a change is executed within the change scenario (for each scenario, we assume a different function δ). For the following change scenario, δ is defined as $\delta(n) = \frac{1}{2}(4n - (-1)^n + 1)$, where n refers to the *nth* change in the change scenario.

$$
\begin{aligned}
&emp0003ReadModify_{ChangeScenario} = \\
&\quad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{maliciousTransactions}) \\
&C_{maliciousTransactions} = \{ \\
&\quad \mathbf{c_1} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, SysAnalyst \rangle\rangle, empDB, \\
&\qquad read, \langle NULL \rangle, pid = emp0003)\rangle, \langle event \rangle, 3, 0) \\
&\quad \mathbf{c_2} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, SysAnalyst \rangle\rangle, empDB, \\
&\qquad modify, \langle NULL \rangle, pid = emp0003)\rangle, \langle event \rangle, 4, 0) \\
&\quad \ldots \\
&\quad \mathbf{c_n} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, SysAnalyst \rangle\rangle, empDB, \\
&\qquad read, \langle NULL \rangle, pid = emp0003)\rangle, \langle event \rangle, \delta(n), 0) \\
&\quad \mathbf{c_{n+1}} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, SysAnalyst \rangle\rangle, empDB, \\
&\qquad modify, \langle NULL \rangle, pid = emp0003)\rangle, \\
&\qquad \langle event \rangle, \delta(n+1), 0)\}
\end{aligned}
$$

The third scenario ($con0002FastRead_{ChangeScenario}$) describes a malicious change scenario by subject con0002, resulting in violations `empDBShortRead` and `empDBLongRead`. In this scenario, a contractor supervisor persistently accesses the `empDB` resource at a rate of 2 s, in order to obtain employee data. The scenario begins 150 s relative to the start of malicious changeload. The function δ denotes the progression in time (seconds) between changes, defined as $\delta(n) = 2n$.

$$
\begin{aligned}
&con0002FastRead_{ChangeScenario} = \\
&\quad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{con0002FastRead}) \\
&C_{con0003FastRead} = \{ \\
&\quad \mathbf{c_1} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, ContractorSupervisor \rangle\rangle, \\
&\qquad empDB, read, \langle NULL \rangle, pid = emp0003)\rangle, \\
&\qquad \langle event \rangle, 150, 0) \\
&\quad \ldots \\
&\quad \mathbf{c_n} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, ContractorSupervisor \rangle\rangle, \\
&\qquad empDB, modify, \langle NULL \rangle, pid = emp0003)\rangle, \\
&\qquad \langle event \rangle, \delta(n), 0)\}
\end{aligned}
$$

For contractors con0003, con0004, and con0005, similar change scenarios exist based on $con0002FastRead_{ChangeScenario}$. However, the scenarios are introduced in stages of 30 s intervals (i.e., con0003 begins at 3 min from the start of the malicious changeload, con0004 begins at 3.5 min, etc.). The rate of changes is defined as $\delta(n) = 2.5n$, where subjects utilise their $\langle permisRole, Contractor \rangle$ attribute, in accessing `empDB`.

Lastly, $emp0003FastAccess_{ChangeScenario}$ describes a further malicious change scenario by emp0003, resulting in violation `lgTShortAccess`. In this scenario, emp0003 persistently requests access every 370 ms and gains access to `lgT` resource (logisticsTool resource), to disrupt the performance of the resource. The scenario begins 900 s relative to the start of malicious changeload. δ denotes a function whereby progression in time (seconds) is defined as $\delta(n) = 0.37n$.

$$
\begin{aligned}
&emp0003FastAccess_{ChangeScenario} = \\
&\quad (SysAccess^{t}_{state}, EnvReqs^{t}_{state}, G_f, C_{emp0003FastAccess}) \\
&C_{emp0003FastAccess} = \{ \\
&\quad \mathbf{c_1} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, Staff\rangle\rangle, lgT, \\
&\qquad read, \langle NULL\rangle, pid = emp0003)\rangle, \\
&\qquad \langle event\rangle, 900, 0) \\
&\quad \ldots \\
&\quad \mathbf{c_n} = (access\_request\_type, as, \\
&\qquad \langle request(\langle\langle permisRole, SysAnalyst\rangle\rangle, lgT, \\
&\qquad read, \langle NULL\rangle, pid = emp0003)\rangle, \\
&\qquad \langle event\rangle, \delta(n), 0)\}
\end{aligned}
$$

This malicious changeload (consisting of the seven change scenarios) concisely describes the *LGZLogistics* malicious behaviour scenario. It is the intention that the changeload can be repeated under various operational conditions, and also used to compare future approaches to self-adaptive authorisation. As such, it can be exploited to execute simulation of changes within an authorisation infrastructure, in order to evaluate the impact of violations, and trigger self-adaptive responses. However, one limitation is that no parser currently exists to execute a defined changeload. Therefore, a changeload can only be viewed as a model of change, which must be manually transformed into an executable script (e.g., JMeter simulation scripts).

5. Experiments

The *LGZLogistics* case study is simulated within a live self-adaptive authorisation infrastructure. This self-adaptive authorisation infrastructure is instantiated across four individual machines. Two machines running **Debian Linux** (512 MB of memory) are deployed hosting an LDAP directory and an installation of SimpleSAMLphp [21]. These are configured to operate as the lgzIS and conIS identity services, respectively. A single machine running **Ubuntu Linux** (2048 MB of memory) is deployed hosting an installation of the PERMIS standalone service, instantiating authorisation service as, and a prototype of the SAAF controller. Lastly, a single 'client' machine running **Windows** (2048 MB) is deployed to simulate activity between subjects accessing a resource, and communicating with services of the authorisation infrastructure.

The rest of this section details a brief overview of the deployment of the prototype of the SAAF controller, a description of how the malicious changeload is simulated within the environment, the execution of experiments, and lastly, a summary of results.

5.1. Deploying the SAAF controller

For the *LGZLogistics* case study, SAAF is deployed as a federated authorisation infrastructure in order to facilitate the adaptation process.

Fig. 7 portrays *LGZLogistic's* federated authorisation infrastructure, based on the architectural model described in Section 2.1. Here, the infrastructure is distributed across multiple management domains (identity provider and service provider domains). *LGZLogistics* operates a service provider domain (to handle authorisation and provision access to resources), and their own identity provider domain (to handle identity management of their own employees). In addition, the *contractor* organisation is said to operate their own identity provider domain (to handle identity management of their own employees).

SimpleSAMLphp [21] is used as the enabling technology to facilitate communication between these management domains. It provides a layer of control over 'what' information can be released or requested (in regards to subjects), and how subjects can be authenticated. Deployments of SimpleSAMLphp are capable of exchanging information via signed or unsigned SAML assertions [23], such as messages containing a set of subject attributes and the subject's unique identifier.

A prototype of the SAAF controller is deployed within *LGZLogistics* service provider domain, whereby it is expected to manage authorisation assets across both management domains. However, self-adaptation over multiple management domains is a challenging and non-trivial problem. Identity providers often do not release uniquely identifiable personal information to service providers, and use transient (TID) or persistent (PID) IDs to allow service providers to identify subjects. In addition, identity providers may not be as forthcoming to accepting adaptations from an organisation outside of their management domain, meaning the SAAF controller can only 'request' adaptation.

A solution to enabling adaptation across multiple management domains is the deployment of an effector managed by the identity provider domain [26]. Here an effector can map a service provider's view of a subject (i.e., from a subject TID/PID to subject LDAP entry), and govern which adaptations to perform. Instantiations of this effector are deployed on each of the identity services (*subject identity adaptation*), as well as an effector capable of deploying and activating policies within the PERMIS authorisation service (*policy adaptation*).

A resource probe is deployed on the empDB resource to observe changes to the state of an employee's job redundancy property (*resource change*). In addition, a probe is deployed on the PERMIS authorisation service to detect *access change* and *policy change*. A probe is not deployed on the contractor's identity service (conIS) simulating a limitation in federated authorisation infrastructures, where third party organisations may prevent immediate access to their subject's attributes (*subject change*). This limits the SAAF prototype's view of the state of access, whereby the SAAF prototype must infer its perception of subjects from the observation of access requests (via the authorisation service as).

The SAAF controller is configured to detect and mitigate the set of violations described in Section 4.6. Here, a solution policy exists containing a set of solutions applicable to mitigating instances of these violations. Each solution contains a weighting of cost to the deploying organisation (e.g., the cost in removing subject access, or removing the trust in an identity provider). A minimum subject impact weighting, on a scale of 0 to 1, is also defined, which is used to constrain a subset of solutions relevant to resolving differing scales of malicious subject behaviour. These weightings are used as part of solution analysis and solution selection. The following solutions are configured for this deployment:

- `S0 - noAdaptation` default solution for when all other solutions cause greater impact over an observed behaviour;
- `S1 - removeSubjectAttribute` removal of an individual abused attribute from a subject (i.e., the cause of a violation);
- `S2 - removeAllSubjectAttributes` removal of all attributes from a subject, typical for when subjects are persistently abusing access;
- `S3 - removeAttributeAssignment` removal of trust in an identity provider in issuing valid attributes (policy change);
- `S4 - removeAllAttributeAssignments` removal of all trust in an identity provider in issuing valid attributes (policy change);
- `S5 - deactivatePolicy` removal of all access to all of *LGZLogistic's* resources.

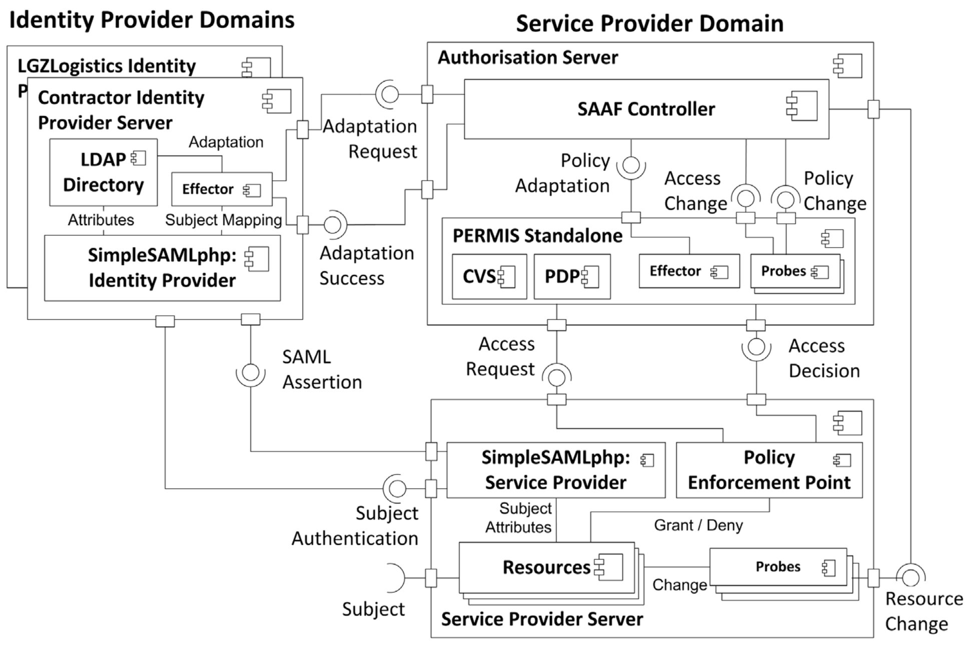

Fig. 7. Self-adaptive federated authorisation infrastructure.

A limitation in this deployment is the inability to use the integrated rbacDSML tool,[3] preventing solution verification from taking place. This is due to the fact that our deployment operates within a federated environment that conforms to ABAC, which rbacDSML is unable to accommodate for. It was decided that evaluating SAAF within a federated environment provided greater contributions as opposed to enhancing rbacDSML to operate within a federated ABAC environment [5]. As a consequence, we assume all adaptation solutions result in acceptable implementations of the access control model.

5.2. Executing LGZLogistics changeload

The execution of the *LGZLogistics* malicious changeload (Section 4.8) is achieved through enacting environment change via a number of protocols:

- LDAP binds [10] - for authenticating subjects within identity providers;
- SAML assertions [23] - for requesting and deliverance of released attributes as signed credentials (to and from identity provider services);
- SOAP messages [28] - for credential validation requests, credential validation responses, access requests, and access decisions (to and from protected resources and authorisation services).

An installation of the JMeter testing tool [29] (deployed on the Windows 'client' machine) automates each of the change types applicable to an authorisation infrastructure, using the aforementioned protocols. Here, subjects are simulated in authenticating, requesting, and obtaining access to protected resources.

Using the experimentation profile proposed by Cámara et al. [8], the malicious changeload is executed across multiple runs as part of four experiments. The two experiments are designed to evaluate the SAAF prototype. Exp1 and Exp2 evaluate the prototype in mitigating malicious changeload under normal and high loads, respectively.

Each experiment is executed six times (referred to as 'runs'), where each run follows the set of stages stated in Fig. 8. The stages of a run are:

1. steady state time — the realisation of a base scenario for the *LGZLogistics* case study that portrays the authorisation infrastructure and its expected system properties, and environment properties, for a typical work day. Steady state time is maintained for a period of 30 min in order to ensure the controller and authorisation infrastructure is evaluated in a warmed up state. During this time, the baseline scenario is simulated within the authorisation infrastructure;
2. environment stimulation — the execution of the malicious changeload. From this period on there is a set of staggered violations in which several periods of 'time to react' and 'time to adapt' overlap environment stimulation. This is necessary in order to evaluate the prototype's ability to detect and mitigate multiple attacks that have been conducted collaboratively;
3. time to react — the detection of malicious behaviour and decision to act;
4. time to adapt — time it takes to perform adaptations;
5. keep time — time needed to observe system recovery post adaptation. In this post adaptation the baseline scenario resumes and no further adaptation takes place.
6. check time — the post analysis of each run. It remains the same for each run within an experiment.

At the end of each run the system and environment states are reset before performing the next run. For consistency, each run (and experiment) observes the same steady state, malicious changeload, and keep/check time.

5.3. Experiments execution

Experiments Exp1 and Exp2 demonstrate adaptation under increasing loads on the controller (in terms of processing environment change).

[3] rbacDSML allows the verification of RBAC access control models, enabling organisations to manage their access control models with greater accuracy, efficiency, and assurance [27].

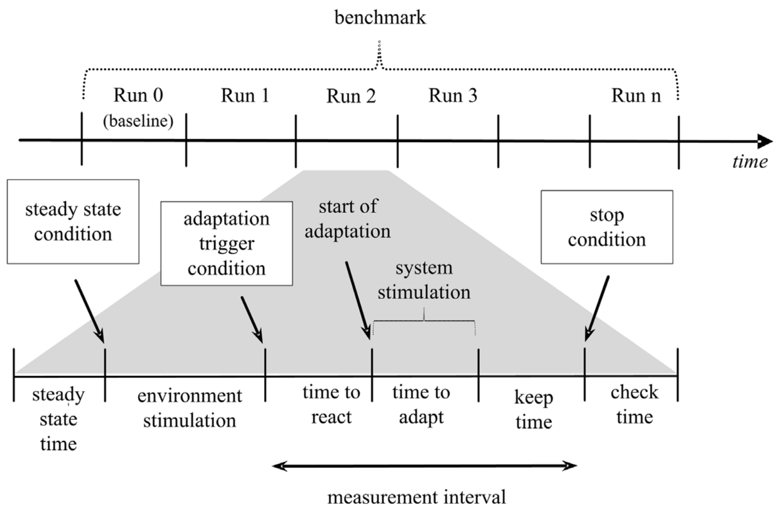

Fig. 8. Executing changeload experimentation profile [8].

5.3.1. Baseline execution

Baseline runs identify the impact of the malicious changeload whereby no adaptation takes place. During these runs, the prototype controller is active, yet limited to only detecting the number and types of violations that have occurred.

Fig. 9(i) and (ii) describe the rate of access of key subjects within the *LGZLogistics* authorisation infrastructure (taken at minute intervals). Note that 'all.Staff' indicates an aggregate rate for all subjects with attribute ⟨*permisRole*, *staff*⟩, whereas, all others represent the access requests of an individual. Fig. 9(i) depicts execution of malicious changeload under normal load, simulated as the continuation of the base scenario throughout environment stimulation. Fig. 9(ii) depicts execution of malicious changeload under high load, simulated as an increase to the base scenario's 'staff' rate of access, from 20req/min to 600req/min.

The normal load baseline (i) is representative of a baseline run for Exp1 and Exp2 since the experiments undergo the same steady state and malicious changeload scenarios for their corresponding runs.

Comparing the baseline runs portrayed minimal difference in violations observed. Point A indicates the start of the malicious changeload (1800 s into the run), where the *setSubjectRedundancies* change scenario is executed, sending the controller several resource change events. It also indicates the start of *emp0003ReadModify* change scenario, at point B, where a system analyst begins to persistently read and modify records in empDB at a rate of 15req/min. At point C a contractor supervisor (*con0002FastRead*) begins to persistently read the empDB resource at a rate of 33req/min. This is followed by D, where three contractors also begin malicious behaviour, exhibiting a slightly lower request rate of 24req/min. Lastly, at point E, *emp0003FastAccess* change scenario is stimulated, representing a system analyst attempting to disrupt the performance of resource lgT, accessing at a rate of 160req/min.

The only exception between the two baselines is indicated at point F (high load baseline). As a result of the client machine being pushed to its limits (overloaded by *emp0003FastAccess*), a slowdown in load occurred after 3000 s into the run. Whilst this presented an anomaly to the baseline, adaptation runs were not impacted, as shown in Fig. 10.

Regarding the detection of malicious behaviour, the controller detected 275 violations in normal load (i), and 260 violations in high load (ii). These were confined to six types of violations: `dueRedundancy`, `empDBTransaction`, `empDBShortRead`, `empLongRead`, `lgTShortAccess`, `empLongModify`. The high load baseline had fewer detections due to the slowdown of client requests at 3000 s into the run.

5.3.2. Adaptation execution

Experiments Exp1 and Exp2 undergo the same malicious changeload as the baseline execution, albeit against a normal and high load, respectively. The experiments and results to be discussed in the following take as a basis Fig. 10 and Table 1.

Both experiments resulted in the consistent identification and selection of solutions to violations, where 14 attack steps were identified and responded to. Table 1 details these attack steps, where each *step* describes:

- A malicious *subject* who has carried out a violation;
- The calculated *impact* of the *subject* to the organisation;
- The *violation* observed;
- A set of *identifiedsolutions* in which to mitigate the violation;
- The *selectedsolution* used to mitigate the violation;
- The response time (R_{Time}) in which to select the solution;
- The time to carry out an adaptation A_{Time};
- The result as to whether a solution was successful (1), failed (0), or not applicable (i.e., no adaptation performed).

Reviewing Table 1, steps 1 to 4 identify a resource change event (at point A), which triggered the violation `dueRedundancy`. For each subject, an impact level was calculated based on past behaviour observed. The controller calculated a low impact for these subjects (0.07), as none of these subjects have any previous violations. However, the result of this means that the controller is less tolerable to future violations.

Step 5 portrays the controller's first adaptation in response to subject *emp0003* triggering a second violation (`empDBTransaction` at point B). As a result, the subject's impact level was recalculated from 0.07 to 0.4. Solution S1 was identified (as being within scope of the subject's level of impact), and realised in the form of an adapted ABAC model (whereby the subject's ⟨*permisRole*, *SysAnalyst*⟩ attribute is removed). The adapted ABAC model was then assessed by the controller's planning stage, ensuring the identified solution does not cause a greater cost to the organisation over observed violations. As the solution only impacts the malicious subject, and no other solution is applicable to the impact of the subject, solution S1 is selected. Solution S1 is enacted as a SOAP message [26] sent to the subject's relevant identity provider (lgzIS) effector. The effector accepts the request and removes *emp0003's SysAnalyst* attribute. The consequence of this adaptation is that *emp0003* is no longer able to gain future access to empDB, as the employee lacks the necessary access rights.

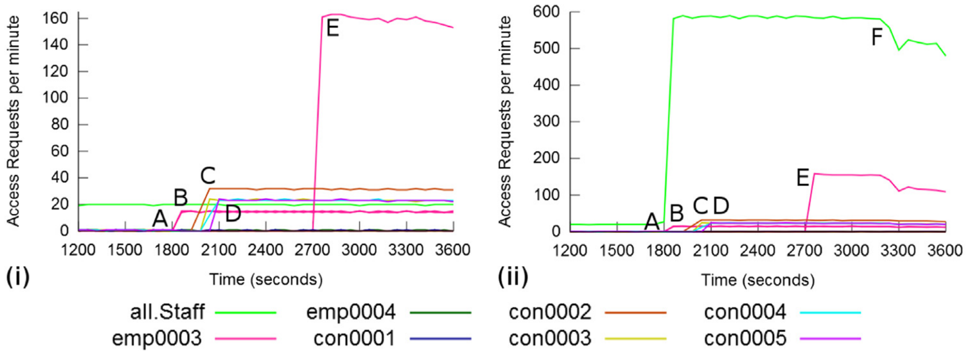

Fig. 9. (i) Baseline Exp1 normal load, (ii) baseline Exp2 high load.

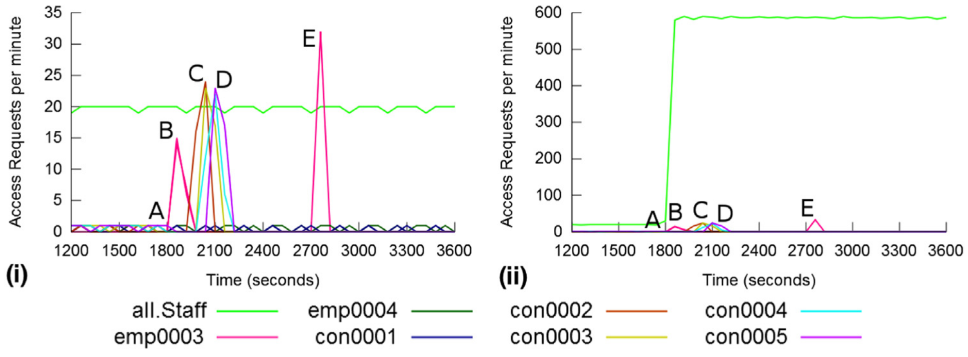

Fig. 10. (i) Exp1 normal load, (ii) Exp2 high load.

Table 1
Exp1: Adaptation with normal load (time in milliseconds)

Step	Subject	Impact	Violation	Identified solutions	Selected solutions	R_{Time} (Avg, Std)	A_{Time} (Avg, Std)	Result
1	emp03	0.07	dueRedundancy	S0	S0	4.6, 0.5	N/A	N/A
2	emp04	0.07	dueRedundancy	S0	S0	2, 0	N/A	N/A
3	emp05	0.07	dueRedundancy	S0	S0	2, 0	N/A	N/A
4	emp06	0.07	dueRedundancy	S0	S0	1.8, 0.4	N/A	N/A
5	emp03	0.4	empDBTransaction	S1	S1	291.6, 59.6	182.2, 48.6	1
6	con02	0.07	empShortRead	S0	S0	1.4, 0.5	N/A	N/A
7	con02	0.27	empShortRead	S1	S1	186.6, 77.5	153.4, 22	1
8	con03	0.07	empShortRead	S0	S0	2.4, 1.1	N/A	N/A
9	con04	0.07	empShortRead	S0	S0	1.6, 0.5	N/A	N/A
10	con03	0.27	empShortRead	S1	S1	94.8, 33.8	165.2, 63.2	1
11	con05	0.07	empShortRead	S0	S0	4.6, 2.8	N/A	N/A
12	con04	0.27	empShortRead	S1	S1	139.8, 23.1	146.2, 38.9	1
13	con05	0.27	empShortRead	S1	S1	70.2, 9.7	120.4, 27.7	1
14	emp03	0.8	lgTShortAccess	S2, S3, S4, S5	S2	297.8, 35.8	189.4, 63.8	1

In steps 6 to 13, the contractor supervisor (*con0002*) and three other contractors are detected at points C and D respectively, triggering violation `empDBShortRead`. Detection results in a similar process to that of the first attack by *emp0003* (step 5), where each subject's impact is recalculated and appropriate solutions are identified and enacted. Eventually, each contractor's relevant attributes in accessing the *empDB* resource are identified and removed via a SOAP message sent to the contractor's identity provider (conIS) effector.

In the final step, *emp0003* rapidly accesses the lgT resource, this time using their remaining attribute $\langle$*permisRole*, *Staff*$\rangle$. This triggered the violation `lgTShortAccess`, whereby the controller calculates the subject's impact level as 0.8. Several solutions are now applicable given this impact weighting, including solutions that result in policy adaptation. However, as the subject has been identified as the source of two previous violations, but is the only subject that has abused their *permisRole* attribute of *SysAnalyst* and *Staff*, solution S2 is enacted. This results in the complete removal of access for subject *emp0003*.

As a result of subject identity adaptation, malicious behaviour by subjects were mitigated given the abuse of access rights. Moreover, subject identity adaptation ensured that non-malicious subjects remained able to request and gain access to protected resources, as evident by the rate of access maintained for *all.staff*, as shown in Fig. 10.

5.4. Summary of results

In the experiments performed, adaptation resulted in preventing the detected malicious subject(s) from gaining further access. This was achieved through removing the abused access right (assigned attribute), removing all of a subject's access rights at identity provider level, or removing varying degrees in trust of the contractor identity provider.

In Exp1 and Exp2, no impact was made to authorisation services in terms of the service being able to perform its duties. This reflects the fundamental design of SAAF, which promotes separation of concerns between adaptation and authorisation.

In these experiments, the impact on identity provider services was negligible. There was no observable rise in latency in subject authentication and attribute release as a result of identity provider adaptation. However, malicious subjects were impacted at identity provider level, in terms of attribute removal, yet this was the desired consequence of adaptation.

Further experiments have been performed using different malicious changeloads that trigger other self-adaptive strategies for mitigating malicious behaviour [30], and all of them have demonstrated the effectiveness of the SAAF prototype in handling the violations identified in the malicious changeloads.

5.4.1. Subject identity versus policy adaptation

The experiments portray scenarios that exemplify subject identity adaptation and policy adaptation. A scenario where a controller is capable of performing subject identity adaptation against all identity providers, and a scenario where the controller is limited in performing subject identity adaptation (requiring policy adaptation).

Subject identity adaptation is seen as the economical choice, whereby malicious behaviour can be mitigated with no impact to non-malicious subjects. When subject identity adaptation was possible, the malicious subjects' behaviour was mitigated almost immediately, preventing future violations. However, where subject identity adaptation was not possible, subjects were capable of repeating violations until the controller identified that the cost of unresolved violations warranted policy adaptation.

Policy adaptation has far greater consequence in comparison to subject identity adaptation, which is calculated (in part) by the number of non-malicious subjects that will lose access to resources as a result of an adaptation.

Regardless of type, each adaptation results in a concrete adjustments to the authorisation infrastructure. Adaptations ultimately control the outcome of future access decisions, and whether or not subjects can be authorised in accessing resources.

5.4.2. Performance

Whilst benchmarking the performance is not the main objective for this paper, the performance observed in the experiments requires some explanation. Of particular interest is the performance of different types of adaptation. Performance is directly related to the number of violations the controller can identify, the size of its access control model, the number of previously identified violations, the number of solutions applicable to an identified violation, and the type of adaptation to be performed. For each experiment, these factors remained persistent, relative to the given experiment step.

A concern was the high standard deviation observed (max 99 ms) between experiment runs for some adaptations, specifically in regards to the time it took the controller to react and decide upon solutions. Steady state time was used to place the controller in a warmed up state. However, due to a mix of factors the standard deviation failed to improve. Some of these factors include network fluctuation between communication of the prototype controller and its effectors, the triggering of Java garbage collection and Java's code optimisation, and that the controller prototype is yet to be optimised. To compensate for this, further experiment runs are required, but were limited to 6 runs per experiment (due to the hour long run-time of each run).

6. Discussion

The *LGZLogistics* case study has provided a scenario for demonstrating and evaluating the detection and mitigation a malicious changeload. This has been achieve through self-adaptation in the context of a federated environment consisting of multiple management domains. Probes and effectors have shown to facilitate automated adaptation across these management domains, where there is arguably a greater need for automation given the fact that federations contain large and unknown user bases.

6.1. Evaluation approach

The experiment was designed to demonstrate the resilience of the SAAF prototype in mitigating malicious behaviour under repeatable conditions. This required simulating a predefined malicious changeload for triggering self-adaptation, and capture the mitigating responses from the SAAF prototype.

The evaluation of SAAF prototype was performed using a simulation of a large scale deployment, akin to a small to medium sized organisation. This was critical to providing evidence of the SAAF prototype's feasibility in operating within the real world, and that the prototype would consistently mitigate violations in a resilient manner. As such, it was observed that the SAAF prototype was capable of mitigating violations when operating under high loads, and when faced with non-cooperating management domains.

6.2. Detection and adaptation

The goal of this paper was not to improve existing techniques for detection malicious behaviour, rather, to demonstrate a new process for handling insider threat. With that said, detection within the SAAF prototype is worth discussing. The SAAF prototype uses detectors to identify known types of attacks, typically focused on thresholds, which is a common approach in detection of malicious behaviour [31,32].

Adopting a threshold approach for detecting malicious behaviour has the advantage of clearly identifying extremes in user behaviour These rules are then incorporated into rules defined by experts, knowledgeable in the differences between normal and abnormal behaviour. However, if a rule is incorrect or inappropriate for the current state of the system, there is the potential for many false positives. Clearly a challenge for SAAF is to employ detection techniques that can evolve and accommodate such legitimate changes in behaviour.

In SAAF, the decision whether to adapt uses cost sensitive modelling [33] to assess subject impact and impact of solutions. This approach has allowed the aggregation of multiple violations before enacting the appropriate solution. Multiple occurrences of violations arguably strengthens the perception in the subject being malicious,[4] as well as adjudicating the extent of appropriate adaptation. Lastly, through this approach, the organisation in which the SAAF prototype is being deployed has the ability to fine tune the enactment of the alternative solutions, in terms of their costs.

In the experiments discussed, the SAAF prototype considers the metric of rate of access requests as the primary environment property in identifying malicious behaviour. Whilst using this metric has shown to be successful in identifying attacks, for it to be efficient, the level of access control must be fine grained. In addition, a subject's ability to access a resource should be determined by short term (or one time use) credentials issued by their identity provider.

The experiments demonstrated the selection and escalation of solutions in response to detected violations. Whilst this was successful and ultimately viewed as enacting 'appropriate' solutions to violations, the cost sensitive modelling approach employed has

[4] One exception to this is if the behaviour rules specified are incorrect, which is addressed as part of SAAF's limitations.

several limitations. One of the limitation is the fact the approach relies upon weighting solutions by a perceived cost of negative impact to an organisation, which is then compared to a perceived cost of subject activity. Although not observed within the experiment itself, there is potential for multiple solutions in conjunction with observed behaviour to present identical costs (i.e., benefits) to an organisation.

One issue that was not addressed in the experiments is the presence of bottlenecks. Given this implementation of SAAF is a prototype, a notable deficiency in its design is its inability to consider multiple violations during a single iteration of its feedback loop. If violations are detected during the prototype's current analysis of behaviour, multiple violations are queued, analysed, planned and executed in a sequential manner. This may result in an increase of the response times when mitigating behaviour identified in the aforementioned manner, due to failed or redundant adaptations if a previous adaptation has already resolved the violation.

6.3. Threats to validity

This paper has presented the resilience evaluation of the Self-adaptive Authorisation Framework (SAAF) prototype in handling and mitigating malicious behaviour by simulating a case study related to insider threats. However, there are assumptions that may affect the validity of the results.

Regarding internal validity, there is the nature of using case studies and simulation. Specifically, simulation presents a certain amount of bias whereby the violations performed are known, and the prototype controller can be configured in an optimum way to best handle such violations. This is the case in the deployment of the SAAF prototype in which the specification of the malicious changeload and the development of SAAF was done by the same person. Therefore, the simulation approach can only be seen as a means to demonstrate the prototype's resilience in handling known violations. What we have not evaluated is how the prototype handles unknown malicious behaviour, and in particular, unpredictable changes that might violate known behavioural patterns. In our understanding, the key threat to validity are the unknown malicious behaviour, and not so much whether a third party should be responsible for the specification of either SAAF or the malicious changeload. Since unpredictable changes are challenging to simulate, it is vital to evaluate SAAF in a live environment in which real users carry out malicious behaviour. This has been done [7], and that study complements the outcomes of this paper that uses case studies, and simulation of a malicious changeload.

Another threat to internal validity is related to the parameters adopted for the definition of scenarios, malicious changeloads and violations (see Sections 4.5 and 4.6). In this paper, these were selected in order to stimulate the SAAF prototype under normal and high load conditions. However, since there are no clear indicators how these should be selected because it depends on several factors, such as, application domain and deployment environment, the key criterion adopted for the selection of these parameters was to maximise stress conditions when evaluating the resilience of the SAAF prototype.

Regarding external validity, although we have identified the malicious changeload for a specific case study, the general model of malicious changeload, briefly presented in Section 4, could be easily instantiated into different case studies. However, the major challenge would be related to the deployment of SAAF prototype on a new environment, and this could introduce new vulnerabilities at the level of probes and effectors, for example. Moreover, changeload does not consider potential attacks to the infrastructure in which the SAAF prototype is deployed. Since the defined changeload is restricted to activities related to access control policies and their enforcement, direct attacks to the deployment infrastructure were not considered because they are outside the scope of the study. For these changes to be considered, a new changeload definition is required.

While several validity threats exist, the results obtained have shown the value of using a simulation of a malicious changeload when evaluating the resilience of self-adaptive authorisation infrastructures.

7. Related work

In this section, we present related work on self-protecting systems, and on how resilience benchmarking provides a practical way for characterising and comparing self-adaptive authorisation infrastructures.

Self-protecting systems are a specialisation of self-adaptive systems with a goal to mitigating malicious behaviour, and as such, the SAAF prototype can be considered a self-protecting system. In the following, we discuss few of the works that have demonstrated self-protection within the context of mitigating insider attacks. In particular, we discuss two self-protection approaches based on the state of access control, and one approach based on the state system architecture.

One of the approaches to self-protection via access control is SecuriTAS [34]. SecuriTAS is a tool that enables dynamic decisions in awarding access, which is based on a perceived state of the system and its environment. SecuriTAS is similar to dynamic access control approaches, such as RADac [35], in that it has a notion of risk (threat) to resources, and changes in threat leads to a change in access control decisions. However, it furthers the concepts in RADac to include the notion of utility. The main difference between SecuriTAS and SAAF, is that SecuriTAS authorisation infrastructure incorporates self-adaptation by design, and it is based on its own bespoke access control model. SAAF, on the other hand, is a framework that describes how existing access control models, like ABAC, and legacy authorisation infrastructures, like PERMIS [11,24], can be made self-adaptive, so that it can be configured to actively mitigate insider threat. With that said, both approaches demonstrate an authorisation infrastructure's resilience in mitigating insider attacks, by ensuring that authorisation remains relevant to system and environment states (and preventing continuation of attacks by adaptation of security controls).

In contrast to self-protection via access control, architectural-based self-protection (ABSP) [36] presents a general solution to detection and mitigation of security threats via run-time structural adaptation. Rather than reason at the contextual layer of 'access control', ABSP uses an architectural model of the running system to identify the extent of impact of identified attacks. Once attacks or security threats have been assessed, a self-adaptive architectural manager (Rainbow [6]) is used to perform adaptations to mitigate the attack. ABSP shares a number of similarities with intrusion response and prevention systems, particularly, with the scope of adaptations that ABSP can perform (e.g., structural adaptation against network devices and connections). However, because ABSP maintains a notion of 'self', it is able to reason about the impact of adaptations and provide assurance over adaptation before adapting its target system.

Another similar example of self-protection is one proposed by Morin et al. which takes a novel approach in managing access control, through the use of architectural adaptation [37]. When access control policies are changed, the architectural model is updated, resulting in the running system being reconfigured through adaptation. Morin et al.'s approach shows the effective deployment of access control across an entire system, where unlike a top down approach proposed by XACML, there is no centralised point of failure. A limitation in this approach is that this form of architectural adaptation is expensive, requiring all resources that need access control to be engineered in a particular manner, lowering the usefulness of the approach in industry. In addition, it lacks the ability to automatically evolve and reflect changes to access control, once malicious behaviour has occurred. However, the technique poses a novel and viable means of realising a change to access control, once such a change has been formulated.

Although there has been several publications on self-adaptive security [38,39], there are few contributions on how these systems should mitigate in a dynamic way cyber attacks, and in particular, insider threats. However, some of the existing work complements the work of SAAF by providing means in order to manage access control policies [40], and looking into selection of policies based on risk [41]. On other hand, there is similar work to ours, whose goal is to enabling legacy systems to incorporate dynamic security policies, which in the case of Al-Ali et al. is by adapting the system architecture [42].

Compared to other established benchmarks (see Section 2.3), a resilience benchmark can be specified following the same generic principles of benchmarking, but changeload needs to be revised and expanded, and should support a risk-based approach for evaluating by comparison the adaptation mechanisms of a self-adaptive software system [8]. An example of such changeload is the one applied to the resilience evaluation of an architectural-based self-adaptive system [43]. In this paper, we have tailored the original changeload model [8] to the context of self-adaptive authorisation infrastructures and focusing on insider threats. There is no other work, to be best of our knowledge, that has attempted to evaluate the resilience of self-protecting system in a systematic way, not even providing a generic framework that could be instantiated into a wide range of systems.

8. Conclusions

This paper presented an approach for the resilience evaluation of self-adaptive authorisation infrastructures. For demonstrating the overall approach for handling and mitigating malicious behaviour, we have defined a fictitious case study of insider threat, defined a repeatable malicious changeload, and deployed a Self-Adaptive Authorisation Framework (SAAF) prototype. Changes on the operational conditions included: changes to the run-time load of the authorisation infrastructure, changes to the autonomic controller, and changes in the availability of probes and effectors.

The evaluation demonstrated that SAAF was resilient in handling abuse in access control under repeatable conditions, and that SAAF consistently mitigated abuse under normal and high loads. In addition, when faced with restrictions in enacting adaptation, SAAF was able to escalate its selection of policy adaptation in order to overcome failures in subject identity adaptation. It was shown that, whilst subject identity adaptation created minimal impact on non-malicious subjects, it was authorisation policy adaptation that was effective in halting abuse of access. Finally, the experiments based on SAAF prototype have also demonstrated its effectiveness in mitigating abuse of access control in federated environments.

Several limitations could be associated with the proposed approach. In the security domain, case studies, compared with real systems, are not able to achieve the same level of coverage regarding the range of attacks the system may encounter. On the other hand, the use of case studies and malicious changeload are fundamental for obtaining the repetitive conditions necessary for evaluating the resilience of a self-adaptive system. However, in order to balance out this limitation, we have demonstrated the resilience of SAAF in the context of a honeypot based deployment that made use of a game [7]. Another limitation when evaluating the resilience of self-adaptive authorisation infrastructures using use cases is the inability of dealing with a wide range of changes that are representative of unexpected subject behaviour. This may include, for example, subjects changing their behaviour when reacting to self-adaptation. This has confirmed that simulation does not consider the run-time consequences of mitigation based on self-adaptation.

As future work, the plan is to use mutation when specifying a malicious changeload in order to uncover vulnerabilities that are particular to certain deployments of self-adaptive authorisation infrastructures.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] R. de Lemos, et al., Software engineering for self-adaptive systems: A second research roadmap, in: R. de Lemos, H. Giese, H. Müller, M. Shaw (Eds.), Software Engineering for Self-Adaptive Systems II, in: Lecture Notes in Computer Science, vol. 7475, Springer Berlin Heidelberg, 2013, pp. 1–32, http://dx.doi.org/10.1007/978-3-642-35813-5_1.

[2] L. Montrieux, R. de Lemos, C. Bailey, Challenges in engineering self-adaptive authorisation infrastructures, in: Y. Yu, A. Bandara, S. Honiden, Z. Hu, T. Tamai, H. Muller, J. Mylopoulos, B. Nuseibeh (Eds.), Engineering Adaptive Software Systems: Communications of NII Shonan Meetings, Springer Singapore, Singapore, 2019, pp. 57–94, http://dx.doi.org/10.1007/978-981-13-2185-6_3.

[3] C. Bailey, D.W. Chadwick, R. de Lemos, Self-adaptive authorization framework for policy based RBAC/ABAC models, in: Proceedings of the 2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing, DASC '11, IEEE Computer Society, Washington, DC, USA, 2011, pp. 37–44, http://dx.doi.org/10.1109/DASC.2011.31.

[4] C. Bailey, D.W. Chadwick, R. de Lemos, Self-adaptive federated authorization infrastructures, J. Comput. System Sci. 80 (5) (2014) 935–952, URL http://www.sciencedirect.com/science/article/pii/S0022000014000154.

[5] C. Bailey, L. Montrieux, R. de Lemos, Y. Yu, M. Wermelinger, Run-time generation, transformation, and verification of access control models for self-protection, in: Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2014, ACM, New York, NY, USA, 2014, pp. 135–144, URL http://doi.acm.org/10.1145/2593929.2593945.

[6] D. Garlan, S.-W. Cheng, A.-C. Huang, B. Schmerl, P. Steenkiste, Rainbow: Architecture-based self-adaptation with reusable infrastructure, Computer 37 (10) (2004) 46–54, http://dx.doi.org/10.1109/MC.2004.175.

[7] C.M. Bailey, R. de Lemos, Evaluating self-adaptive authorisation infrastructures through gamification, in: 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2018, Luxembourg City, Luxembourg, June 25–28, 2018, 2018, pp. 502–513, http://dx.doi.org/10.1109/DSN.2018.00058.

[8] J. Cámara, R. de Lemos, M. Vieira, R. Almeida, R. Ventura, Architecture-based resilience evaluation for self-adaptive systems, Computing 95 (8) (2013) 689–722, http://dx.doi.org/10.1007/s00607-013-0311-7.

[9] J.O. Kephart, D.M. Chess, The vision of autonomic computing, Computer 36 (1) (2003) 41–50, http://dx.doi.org/10.1109/MC.2003.1160055.

[10] V. Koutsonikola, A. Vakali, LDAP: Framework, practices, and trends, IEEE Internet Comput. 8 (5) (2004) 66–72, http://dx.doi.org/10.1109/MIC.2004.44.

[11] D.W. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, T.A. Nguyen, PERMIS: A modular authorization infrastructure, Concurr. Comput. : Pract. Exper. 20 (11) (2008) 1341–1357, http://dx.doi.org/10.1002/cpe.v20:11.

[12] D.M. Cappelli, A.P. Moore, R.F. Trzeciak, The CERT Guide to Insider Threats: How to Prevent, Detect, and Respond to Information Technology Crimes, first ed., Addison-Wesley Professional, 2012.

[13] D. Caputo, M. Maloof, G. Stephens, Detecting insider theft of trade secrets, IEEE Secur. Priv. 7 (6) (2009) 14–21, http://dx.doi.org/10.1109/MSP.2009.110.

[14] J. Oltsik, The 2013 Vormetric insider threat report [Online], 2013, Available from: http://www.vormetric.com/sites/default/files/vormetric-insider-threat-report-oct-2013.pdf. (Accessed 30 September 2018).

[15] IBM, IBM security intelligence with big data [Online], 2018, Available from: http://www-03.ibm.com/security/solution/intelligence-big-data/ (Accessed 30 September 2018).

[16] J.R. Nurse, O. Buckley, P.A. Legg, M. Goldsmith, S. Creese, G.R. Wright, M. Whitty, Understanding insider threat: A framework for characterising attacks, in: Workshop on Research for Insider Threat (WRIT) Held As Part of the IEEE Computer Society Security and Privacy Workshops (SPW14), in Conjunction with the IEEE Symposium on Security and Privacy (SP), IEEE, 2014, pp. 214–228, URL http://www.sei.cmu.edu/community/writ2014/.

[17] L. Spitzner, Honeypots: Catching the insider threat, in: Proceedings of the 19th Annual Computer Security Applications Conference, IEEE, 2003, pp. 170–179.

[18] K. Kanoun, L. Spainhower, Dependability Benchmarking for Computer Systems, Wiley-IEEE Computer Society Pr, 2008.

[19] J. Gray, Benchmark Handbook: For Database and Transaction Processing Systems, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1992.

[20] H. Madeira, Towards a security benchmark for database management systems, in: Proceedings of the 2005 International Conference on Dependable Systems and Networks, DSN '05, IEEE Computer Society, Washington, DC, USA, 2005, pp. 592–601, http://dx.doi.org/10.1109/DSN.2005.93.

[21] SimpleSAMLphp, [Online]. Available from: http://simplesamlphp.org/. (Accessed 30 September 2018).

[22] PERMIS Standalone Authorisation Server, [Online]. Available from: http://sec.cs.kent.ac.uk/permis/. (Accessed 30 September 2018).

[23] OASIS, Security Assertion Markup Language (SAML) Version 2.0, Tech. Rep., OASIS, 2005.

[24] D.W. Chadwick, A. Otenko, The PERMIS X.509 role based privilege management infrastructure, in: Proceedings of the Seventh ACM Symposium on Access Control Models and Technologies, SACMAT '02, ACM, New York, NY, USA, 2002, pp. 135–140, URL http://doi.acm.org/10.1145/507711.507732.

[25] J. Cámara, P. Correia, R. de Lemos, M. Vieira, Empirical resilience evaluation of an architecture-based self-adaptive software system, in: Proceedings of the 10th International ACM Sigsoft Conference on Quality of Software Architectures, QoSA '14, ACM, New York, NY, USA, 2014, pp. 63–72, URL http://doi.acm.org/10.1145/2602576.2602577.

[26] C. Bailey, D.W. Chadwick, R. de Lemos, K.W.S. Siu, Enabling the autonomic management of federated identity providers, in: Proceedings of the 7th IFIP WG 6.6 International Conference on Autonomous Infrastructure, Management, and Security: Emerging Management Mechanisms for the Future Internet - Volume 7943, AIMS'13, Springer-Verlag, Berlin, Heidelberg, 2013, pp. 100–111, http://dx.doi.org/10.1007/978-3-642-38998-6_14.

[27] L. Montrieux, Model-Based Analysis of Role-Based Access Control (Ph.D. thesis), The Open University, 2013.

[28] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H.F. Nielsen, S. Thatte, D. Winer, Simple object access protocol (SOAP) 1.1, 2000.

[29] Apache Software Foundation, Jmeter [online], 2018, Available from: http://jmeter.apache.org/. (Accessed 30 September 2018).

[30] C.M. Bailey, Self-Adaptive Authorisation Infrastructures (Ph.D. thesis), University of Kent, 2015.

[31] C.Y. Chung, M. Gertz, K. Levitt, DEMIDS: A misuse detection system for database systems, in: M.E. van Biene-Hershey, L. Strous (Eds.), Integrity and Internal Control Information Systems, Kluwer Academic Publishers, Norwell, MA, USA, 2000, pp. 159–178, URL http://dl.acm.org/citation.cfm?id=342030.342078.

[32] K. Wang, S. Stolfo, Anomalous payload-based network intrusion detection, in: E. Jonsson, A. Valdes, M. Almgren (Eds.), Recent Advances in Intrusion Detection, in: Lecture Notes in Computer Science, vol. 3224, Springer Berlin Heidelberg, 2004, pp. 203–222, http://dx.doi.org/10.1007/978-3-540-30143-1_11.

[33] C. Strasburg, N. Stakhanova, S. Basu, J.S. Wong, A framework for cost sensitive assessment of intrusion response selection, in: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01, COMPSAC '09, IEEE Computer Society, Washington, DC, USA, 2009, pp. 355–360, http://dx.doi.org/10.1109/COMPSAC.2009.54.

[34] L. Pasquale, C. Menghi, M. Salehie, L. Cavallaro, I. Omoronyia, B. Nuseibeh, SecuriTAS: A tool for engineering adaptive security, in: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE '12, ACM, New York, NY, USA, 2012, pp. 19:1–19:4, URL http://doi.acm.org/10.1145/2393596.2393618.

[35] R. McGraw, Risk-Adaptable Access Control (RADac), Tech. Rep., National Institute of Standards and Technology (NIST), 2009.

[36] E. Yuan, S. Malek, B. Schmerl, D. Garlan, J. Gennari, Architecture-based self-protecting software systems, in: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures, ACM, 2013, pp. 33–42.

[37] B. Morin, T. Mouelhi, F. Fleurey, Y. Le Traon, O. Barais, J.-M. Jézéquel, Security-driven model-based dynamic adaptation, in: Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, ASE '10, ACM, New York, NY, USA, 2010, pp. 205–214, URL http://doi.acm.org/10.1145/1858996.1859040.

[38] E. Yuan, N. Esfahani, S. Malek, A systematic survey of self-protecting software systems, ACM Trans. Auton. Adapt. Syst. 8 (4) (2014) 17:1–17:41, URL http://doi.acm.org/10.1145/2555611.

[39] G. Tziakouris, R. Bahsoon, M.A. Babar, A survey on self-adaptive security for large-scale open environments, ACM Comput. Surv. 51 (5) (2018) 100:1–100:42, URL http://doi.acm.org/10.1145/3234148.

[40] M. Hummer, M. Kunz, M. Netter, L. Fuchs, G. Pernul, Adaptive identity and access management—contextual data based policies, EURASIP J. Inf. Secur. 2016 (1) (2016) 19, http://dx.doi.org/10.1186/s13635-016-0043-2.

[41] D. Díaz-López, G. Dólera-Tormo, F. Gómez-Mármol, G. Martínez-Pérez, Dynamic counter-measures for risk-based access control systems: An evolutive approach, Future Gener. Comput. Syst. 55 (2016) 321–335, URL http://www.sciencedirect.com/science/article/pii/S0167739X14002052.

[42] R. Al-Ali, P. Hnetynka, J. Havlik, V. Krivka, R. Heinrich, S. Seifermann, M. Walter, A. Juan-Verdejo, Dynamic security rules for legacy systems, in: Proceedings of the 13th European Conference on Software Architecture - Volume 2, ECSA '19, ACM, New York, NY, USA, 2019, pp. 277–284, URL http://doi.acm.org/10.1145/3344948.3344974.

[43] J. Cámara, R. de Lemos, N. Laranjeiro, R. Ventura, M. Vieira, Robustness-driven resilience evaluation of self-adaptive software systems, IEEE Trans. Dependable Secure Comput. 14 (1) (2017) 50–64, http://dx.doi.org/10.1109/TDSC.2015.2429128.

Rogério de Lemos is a senior lecturer in the School of Computing at the University of Kent since 1999. In the past, he was a Senior Research Associate at the Centre for Software Reliability (CSR) at the University of Newcastle upon Tyne, and invited assistant professor at the University of Coimbra in Portugal. His research interests are on self-adaptive software systems, architecting dependable and secure systems, and resilient AI. He is a member of the IEEE Computer Society.